**飞腾平台性能测试指导手册**

**服务器篇**

|  |
| --- |
| **天津飞腾信息技术有限公司**  **二零一九年十二月** |

**文档修订历史**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **版本** | **作者** | **日期** | **变化内容描述** | **审核人** | **批准人** | **批准日期** |
| 1.0 | 许高峰 | 2019.10.5 | 初稿 |  |  |  |
| 1.1 | 任利峰 | 2019.12.17 | 调整文档结构并增加测试工具使用说明 |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |

**版权所有© 天津飞腾信息技术有限公司2019。**

天津飞腾信息技术有限公司对此文档内容拥有版权，并受法律保护。

目录

[第一章. 概述](#_Toc25370)

[1. 编写目的](#_Toc9659)

[2. 性能测试项](#_Toc21354)

[第二章. SPEC使用说明](#_Toc18713)

[1. SPEC CPU®2017简介](#_Toc3463)

[2. SPEC CPU®2017安装](#_Toc10943)

[2.1挂载光盘](#_Toc17087)

[2.2安装SPEC CPU®2017](#_Toc24115)

[2.3配置phytium.cfg文件](#_Toc2608)

[2.4安装gfortran](#_Toc20767)

[2.5配置shrc环境](#_Toc2513)

[3. SPEC CPU®2017用例测试](#_Toc11438)

[3.1单核测试](#_Toc15595)

[3.2多核测试](#_Toc17065)

[4. SPEC CPU®2006简介](#_Toc27454)

[5. SPEC CPU®2006安装](#_Toc27777)

[6. SPEC CPU®2006用例测试](#_Toc15971)

[6.1多核测试](#_Toc13229)

[6.2 speed测试](#_Toc3884)

[6.3 常用测试组合](#_Toc5642)

[7. 系统负载状态查询](#_Toc4878)

[第三章. STREAM使用说明](#_Toc3609)

[1. STREAM简介](#_Toc4574)

[2. STREAM安装](#_Toc1618)

[3. STREAM测试](#_Toc20064)

[3.1单线程测试](#_Toc22499)

[3.2多线程测试](#_Toc24709)

[第四章. LMBENCH使用说明](#_Toc14483)

[1. LMBENCH简介](#_Toc30271)

[2. LMBENCH安装](#_Toc22695)

[2.1下载与解压](#_Toc22616)

[2.2编译配置](#_Toc14583)

[2.3查询结果](#_Toc15030)

[2.4再次运行](#_Toc17042)

[3. LMBENCH结论分析](#_Toc23526)

[3.1系统基本参数](#_Toc13475)

[3.2处理器Process性能](#_Toc21997)

[3.3数学运算](#_Toc25214)

[3.4上下文切换](#_Toc21163)

[3.5本地通讯延时](#_Toc19188)

[3.6文档、内存延时](#_Toc10329)

[3.7本地通信带宽](#_Toc7006)

[3.8内存操作延时](#_Toc18146)

[第五章. LTP使用说明](#_Toc1032)

[1. Ltp简介](#_Toc25904)

[1.1 软件测试设计](#_Toc12509)

[1.2 软件测试方法](#_Toc5531)

[1.3系统监控](#_Toc4225)

[2. Ltp安装](#_Toc25376)

[2.1下载与ltp源码包](#_Toc9538)

[2.2编译安装ltp](#_Toc25869)

[3. Ltp初始测试](#_Toc13957)

[3.1运行初始测试](#_Toc25920)

[3.2查看初始测试结果](#_Toc11927)

[4. Ltp压力测试](#_Toc21805)

[4.1执行压力测试](#_Toc4105)

[4.2查看压力测试结果](#_Toc28743)

[5. Ltp其他常规测试命令](#_Toc1802)

[第六章. IOzone使用说明](#_Toc7808)

[1. IOzone简介](#_Toc32336)

[1.1软件简介](#_Toc32764)

[1.2测试读写模式](#_Toc7315)

[1.3主要参数说明](#_Toc4909)

[2. IOzone安装](#_Toc30382)

[3. IOzone测试](#_Toc12525)

[3.1 测试用例](#_Toc25559)

[3.2 结论与分析](#_Toc13910)

[4. Iostat查询](#_Toc3218)

[4.1 iostat用法](#_Toc12900)

[4.2显示所有设备负载情况](#_Toc21069)

[4.3常用iostat命令](#_Toc23976)

[4.4重点关注性能监控指标](#_Toc23748)

[第七章. SPECjvm2008使用说明](#_Toc5271)

[1. SPECjvm2008简介](#_Toc17311)

[2. SPECjvm2008安装](#_Toc4401)

[3. 配置JAVA环境变量](#_Toc15850)

[4. SPECjvm2008测试](#_Toc18207)

[第八章. UnixBench使用说明](#_Toc20051)

[1. UnixBench简介](#_Toc7538)

[2. UnixBench安装](#_Toc23121)

[3. UnixBench测试](#_Toc23467)

[第九章. Netperf使用说明](#_Toc19370)

[1. Netperf简介](#_Toc5866)

[2. Netperf安装](#_Toc24822)

[3. Netperf测试](#_Toc5832)

[3.1批量网络流量的性能测试](#_Toc29858)

[3.1.1 TCP\_STREAM测试](#_Toc27539)

[3.1.2 UDP\_STREAM测试](#_Toc27126)

[3.2测试请求/应答网络流量的性能](#_Toc18198)

[3.2.1 TCP\_RR 测试](#_Toc24844)

[3.2.2 TCP\_CRR 测试](#_Toc4248)

[3.2.3 UDP\_RR 测试](#_Toc5463)

[第十章. iPerf使用说明](#_Toc27109)

[1. iPerf简介](#_Toc11537)

[2. iPerf安装](#_Toc9594)

[3. iPerf测试](#_Toc10535)

[3.1 iPerf工作模式和参数](#_Toc32596)

[3.2 iPerf案例介绍](#_Toc8545)

[第十一章. FIO使用说明](#_Toc13551)

[1. FIO简介](#_Toc5843)

[2. FIO安装](#_Toc15987)

[3. FIO测试](#_Toc14046)

[3.1 FIO参数说明](#_Toc14889)

[3.2 FIO测试](#_Toc31570)

[3.3 测试场景](#_Toc25768)

# 概述

## 编写目的

本次文档编写主要目的为飞腾平台性能测试上提供测试方法，测试项目分为：综合性能测试、压力测试、内存测试、网络测试、磁盘 IO 测试及其它测试类。更好地指导各厂商高质、高效、有序完成测试任务，特制定本手册。

## 性能测试工具

系统性能测试关注CPU、内存、系统综合性能、网络、I/O等方面的测试，通过标准的基准测试工具，评价服务器硬件的性能指标。

|  |  |  |  |
| --- | --- | --- | --- |
| **序号** | **工具/软件名称** | **版本** | **用 途** |
| 1 | SPEC CPU | 2017/2006 | CPU性能测试工具 |
| 2 | STREAM | v5.10 | 内存性能测试工具 |
| 3 | Lmbench | V3.0 | 系统性能测试工具 |
| 4 | LTP | V20160510 | 稳定性能测试工具 |
| 5 | IOzone | V3\_487 | 磁盘I/O性能测试工具 |
| 6 | SPECjvm2008 | V1.01 | JAVA虚拟机测试工具 |
| 7 | Unixbench | V5.1.3 | 操作系统综合性能测试工具 |
| 8 | Netperf | V2.7.0 | 网络性能测试 |
| 9 | iPerf | V3.7 | 网络性能测试工具 |
| 10 | FIO | V3.16 | 磁盘压力测试工具 |

# SPEC使用说明

## SPEC CPU®2017简介

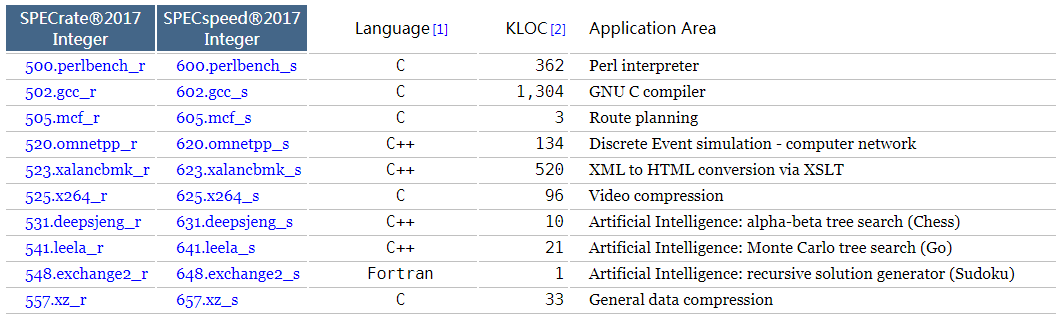
SPEC CPU 2017是一套CPU子系统测试工具，包括4大种类共43个测试，包括在测试整型速度、浮点运算速度的性能测试SPECspeed 2017、SPECspeed 2017 Floating Point以及测试整型并发速率和浮点并发速率SPECrate 2017 Integer和SPECrate 2017 Floating Point 。

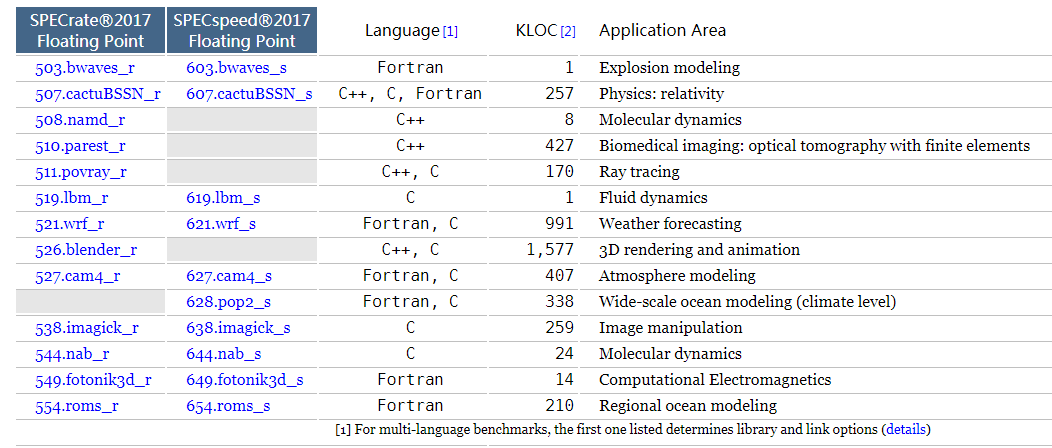
* \_SPECspeed 2017 Integer，有10个基准;
* \_SPECrate 2017 Integer，有10个基准;
* \_SPECspeed 2017浮点，有10个基准;
* \_SPECrate2017年浮点，有13个基准;

借助于OpenMP多线程API，SPEC CPU 2017还可以在测试过程中监控功耗。

新的基准测试套件移除了libquantum项目（libquantum是模拟量子计算机的库文件，由于测试项目被认为仅对Intel有利）。

SPEC CPU 2017基准测试工具可在该网站购买下载，正式授权版本为1000美元。





## SPEC CPU®2017安装

### 2.1挂载光盘

root@ISG\_test01#mount -oloop cpu2017-1\_0\_5.iso /mnt

mount: /dev/loop0 is write-protected, mounting read-only

### 2.2安装SPEC CPU®2017

root@ISG\_test01:/mnt# ./install.sh -d /opt/SPEC/CPU2017

SPEC CPU2017 Installation

Top of the CPU2017 tree is '/mnt'

Installing FROM /mnt

Installing TO /opt/SPEC/CPU2017

Is this correct? (Please enter 'yes' or 'no')

**Yes #这里输入yes**

The following toolset is expected to work on your platform. If the

automatically installed one does not work, please re-run install.sh and

exclude that toolset using the '-e' switch.

The toolset selected will not affect your benchmark scores.

linux-aarch64 For 64-bit AArch64 systems running Linux.

Built with GCC 4.8.4 (Ubuntu/Linaro

4.8.4-2ubuntu1~14.04.1) on an HPE

Moonshot running Linaro v14.04.1 LTS.

=================================================================

Attempting to install the linux-aarch64 toolset...

Unpacking CPU2017 base files (44 MB)

Unpacking CPU2017 tools binary files (150.2 MB)

Unpacking 500.perlbench\_r benchmark and data files (102 MB)

Unpacking 502.gcc\_r benchmark and data files (240.3 MB)

Unpacking 503.bwaves\_r benchmark and data files (0.2 MB)

Unpacking 505.mcf\_r benchmark and data files (8.5 MB)

Unpacking 507.cactuBSSN\_r benchmark and data files (12.5 MB)

Unpacking 508.namd\_r benchmark and data files (8.3 MB)

Unpacking 510.parest\_r benchmark and data files (25.6 MB)

Unpacking 511.povray\_r benchmark and data files (23.3 MB)

Unpacking 519.lbm\_r benchmark and data files (4.3 MB)

Unpacking 520.omnetpp\_r benchmark and data files (56.6 MB)

Unpacking 521.wrf\_r benchmark and data files (217.2 MB)

Unpacking 523.xalancbmk\_r benchmark and data files (212 MB)

Unpacking 525.x264\_r benchmark and data files (57.9 MB)

Unpacking 526.blender\_r benchmark and data files (215.7 MB)

Unpacking 527.cam4\_r benchmark and data files (348.6 MB)

Unpacking 531.deepsjeng\_r benchmark and data files (0.5 MB)

Unpacking 538.imagick\_r benchmark and data files (80.7 MB)

Unpacking 541.leela\_r benchmark and data files (3.8 MB)

Unpacking 544.nab\_r benchmark and data files (38.6 MB)

Unpacking 548.exchange2\_r benchmark and data files (0.1 MB)

Unpacking 549.fotonik3d\_r benchmark and data files (5.2 MB)

Unpacking 554.roms\_r benchmark and data files (11.4 MB)

Unpacking 557.xz\_r benchmark and data files (104.1 MB)

Unpacking 600.perlbench\_s benchmark and data files (3.1 MB)

Unpacking 602.gcc\_s benchmark and data files (0.9 MB)

Unpacking 603.bwaves\_s benchmark and data files (0 MB)

Unpacking 605.mcf\_s benchmark and data files (0.1 MB)

Unpacking 607.cactuBSSN\_s benchmark and data files (0.1 MB)

Unpacking 619.lbm\_s benchmark and data files (30.4 MB)

Unpacking 620.omnetpp\_s benchmark and data files (0.1 MB)

Unpacking 621.wrf\_s benchmark and data files (0.3 MB)

Unpacking 623.xalancbmk\_s benchmark and data files (0.1 MB)

Unpacking 625.x264\_s benchmark and data files (0.2 MB)

Unpacking 627.cam4\_s benchmark and data files (0.5 MB)

Unpacking 628.pop2\_s benchmark and data files (283.8 MB)

Unpacking 631.deepsjeng\_s benchmark and data files (0.2 MB)

Unpacking 638.imagick\_s benchmark and data files (0.3 MB)

Unpacking 641.leela\_s benchmark and data files (0 MB)

Unpacking 644.nab\_s benchmark and data files (0.1 MB)

Unpacking 648.exchange2\_s benchmark and data files (0 MB)

Unpacking 649.fotonik3d\_s benchmark and data files (0.1 MB)

Unpacking 654.roms\_s benchmark and data files (1.1 MB)

Unpacking 657.xz\_s benchmark and data files (0.2 MB)

Unpacking 996.specrand\_fs benchmark and data files (0 MB)

Unpacking 997.specrand\_fr benchmark and data files (0 MB)

Unpacking 998.specrand\_is benchmark and data files (0 MB)

Unpacking 999.specrand\_ir benchmark and data files (6.4 MB)

Checking the integrity of your source tree...

Checksums are all okay.

Unpacking binary tools for linux-aarch64...

Checking the integrity of your binary tools...

Checksums are all okay.

Testing the tools installation (this may take a minute)

...................................................................................................................................................................................................................................................................................................................-.......

Installation successful. Source the shrc or cshrc in

/opt/SPEC/CPU2017

to set up your environment for the benchmark.

### 2.3配置phytium.cfg文件

root@ISG\_test01:cp Example-gcc-linux-aarch64.cfg phytium.cfg

配置文件phytium.cfg如下：



root@ISG\_test01:cd /opt/SPEC/CPU2017/config

修改的配置文件主要内容如下：

root@ISG\_test01# diff Example-gcc-linux-aarch64.cfg phytium.cfg

|  |
| --- |
| 80c80,81  < tune = base  ---  > #tune = base  > tune = all  106c107  < copies = 1 # EDIT to change number of copies (see above)  ---  > copies = 64 # EDIT to change number of copies (see above)  130c131,132  < % define gcc\_dir /opt/rh/devtoolset-6/root/usr # EDIT (see above)  ---  > #% define gcc\_dir /opt/rh/devtoolset-6/root/usr # EDIT (see above)  > % define gcc\_dir /usr # EDIT (see above)  333a336,635 |

### 2.4安装gfortran

root@ISG\_test01:vi /etc/apt/source-list

注释原文件内容，然后添加如下内容

deb http://archive.kylinos.cn/kylin/KYLIN-ALL 4.0.2sp2-server-ft2000 main restricted universe multiverse

root@ISG\_test01:# apt-get update

root@ISG\_test01:# apt install gfortran

### 2.5配置shrc环境

root@ISG\_test01# cd /opt/SPEC/CPU2017

root@ISG\_test01#source shrc

## SPEC CPU®2017用例测试

### 3.1单核测试

#cd /opt/SPEC/CPU2017/bin

#./runcpu --config=/opt/SPEC/CPU2017/config/phytium.cfg --size=ref --tune=all --iterations=3 intrate fprate intspeed fpspeed

--size | -i为规模选择：分为test(简单规模模式，测试时间最短)、train(中级规模模式，测试时间居中)、ref(高级规模模式，测试时间最长)

--tune | -T为测试基准:分为base(基准值)和peak(峰值)，all：表示SPEC CPU2017将运行基准测试程序中的所有测试项目；

--iterations=3，--iterations | -n为测试遍数，官方建议为3次

### 3.2多核测试

#cd /opt/SPEC/CPU2017/bin

#./runcpu --config=/opt/SPEC/CPU2017/config/phytium.cfg --size=ref --tune=all --iterations=3 --copies=64 --thread=64 intrate fprate intspeed fpspeed

--size | -i为规模选择：分为test(简单规模模式，测试时间最短)、train(中级规模模式，测试时间居中)、ref(高级规模模式，测试时间最长)

--tune | -T为测试基准:分为base(基准值)和peak(峰值)，all：表示SPEC CPU2017将运行基准测试程序中的所有测试项目；

--iterations=3，--iterations | -n为测试遍数，官方建议为3次

**官方链接**

<https://www.spec.org/cpu2017/Docs/>

## SPEC CPU®2006简介

SPEC CPU 2006 benchmark是SPEC行业标准化的CPU测试基准套件。重点测试系统的处理器，内存子系统和编译器。这个基准测试套件包括的SPECint基准和SPECfp基准。

其中SPECint2006基准包含12个不同的基准测试和SPECfp2006年基准包含19个不同的基准测试。

SPEC设计了这个套件提供了一个比较标准的计算密集型，高性能的跨硬件的CPU测试工具。在SPEC CPU 2006基准有几种不同的方法来衡量计算机性能。一种方式是测量计算机完成单一任务的速度；另一种方式吞吐量，容量或速率的测量。

## SPEC CPU®2006安装

#mkdir /opt/spec2006

#tar xvf cpu2006-ft.tar.gz –C /opt/spec2006

#cd /opt/spec2006/cpu2006-ft/cpu2006

#./install.sh

SPEC CPU2006 Installation

Top of the CPU2006 tree is '/opt/spec2006/cpu2006-ft/cpu2006'

These appear to be valid toolsets:

ft-spec2006-tool

aarch64-linux

Enter the architecture you are using:

**aarch64-linux** #输入aarch64-linux架构

Checking the integrity of your source tree...

Checksums are all okay.

Removing previous tools installation

Unpacking binary tools for aarch64-linux...

Checking the integrity of your binary tools...

Checksums are all okay.

Top of SPEC benchmark tree is '/opt/spec2006/cpu2006-ft/cpu2006'

Everything looks okay. cd to /opt/spec2006/cpu2006-ft/cpu2006,

source the shrc file and have at it!

#source shrc

## SPEC CPU®2006用例测试

### 6.1多核测试

#cd /opt/spec2006/cpu2006-ft/cpu2006/bin

#./runspec –c ../config/linux64-arm64-gcc52.cfg –T base –n 3 –r 64 –I –i ref all

### 6.2 speed测试

#./runspec –c linux64-arm64-gcc52.cfg –T base –n 3 –speed –I –i ref all

**参数说明：**

--config | -c为执行时加载的配置文件，文件存放在cpu2006-ft/cpu2006/config中

--tune | -T为测试基准:分为base(基准值)和peak(峰值)

--size | -i为规模选择：分为test(简单规模模式，测试时间最短)、train(中级规模模式，测

试时间居中)、ref(高级规模模式，测试时间最长)

--iterations | -n为测试遍数，结果要生成pdf，n=3

--rate | -r rate,多线程测试

--output\_format 输出文件格式 默认为html text 可以选择html、mail、pdf、ps、printer、raw、screen、text 等多种方式。

all：SPEC CPU2006将运行基准测试程序中的所有测试项目。

**注意事项：**

* 确保此时的系统中包含以下编译器：g++、gcc、gfortran编译器
* --tune参数中base和peak的区别在于：base是简单的标准的配置（一般跑个base就可以衡量性能了），peak是可以提供更多个性化的配置编译选项。 默认值就是"--tune base". （--tune 也简写为 -T，如"-T base"、 --tune all （包括base和peak））
* rate和speed的区别：rate是系统的能达到的总体能力的衡量；speed是系统完成一个任务的速度的衡量，一般选择rate来看看一段时间内，系统到底能处理多少任务。默认值就是使用rate这种度量方式。
* 选择copy数量，一般来说copy数量等于CPU核的个数，参数为“-C num”（或--copies num）。注意：如果选择speed度量方式，其copy就是1，是不可以更改的。
* -c file"等于"--config file"，用于指定本次运行的配置文件。
* "-o formate"等于“--output\_format format”，指定生成输出报告、文件的格式。

### 6.3 常用测试组合

1．runspec -c test.cfg -i test -I all 基于最小测试数据集快速执行所有的测试，测试过程中如果某个用例发生错误，则跳过错误用例，继续执行其他用例。

2．runspec -c test.cfg -i ref -n 3 -I all 基于最大测试数据集全面执行所有的测试，用于测试单核CPU，测试过程中如果某个用例发生错误，则跳过错误用例，继续执行其他用例。

3．runspec -c test.cfg –r ref –n 3 fp 基于最大测试数据集，只运行fp测试

4．runspec -c test.cfg –r ref –n 3 int 基于最大测试数据集，只运行int测试

5．runspec -c test.cfg -i ref 473.astar 基于最大测试数据集只执行473.astar单个测试。

6．runspec –c test.cfg –i ref --rate 4 int 基于最大数据测试集进行rate测试，运行4线程测试的分值

## 系统负载状态查询

Linux系统中的uptime命令主要用于获取主机运行时间和查询Linux系统负载等信息。uptime命令可以显示系统已经运行了多长时间，信息显示依次为：现在时间、系统已经运行了多长时间、目前有多少登录用户、系统在过去的1分钟、5分钟和15分钟内的平均负载。uptime命令用法十分简单，直接输入uptime即可。

|  |
| --- |
| root@isgtest04-os:~# uptime  17:34:21 up 7 days, 1:01, 5 users, load average: 63.94, 58.35, 58.5 |

* 17:34:21表示系统当前时间
* up 7 days主机已运行时间，时间越大，说明的机器越稳定。
* 5 users表示用户连接数，是总连接数而不是用户数。
* load average表示系统平均负载，统计最近1、5、15分钟的系统平均负载。系统平均负载是指在特定时间间隔内运行队列中的平均进程数。

# STREAM使用说明

## STREAM简介

STREAM 是业界广为流行的综合性内存带宽实际性能测量工具之一。随着处理器处理核心数量的增多，内存带宽对于提升整个系统性能越发重要，如果某个系统不能够足够迅速地将内存中的数据传输到处理器当中，若干处理核心就会处于等待数据的闲置状态，而这其中所产生的闲置时间不仅会降低系统的效率还会抵消多核心和高主频所带来的性能提升因素。

STREAM 具有良好的空间局部性，是对 TLB 友好、Cache友好的一款测试。STREAM支持Copy 、Scale 、 Add、 Triad四种操作，下面分别介绍四种操作的含义：

Copy操作最为简单，它先访问一个内存单元读出其中的值，再将值写入到另一个内存单元。  
 Scale操作先从内存单元读出其中的值，作一个乘法运算，再将结果写入到另一个内存单元。  
 Add操作先从内存单元读出两个值，做加法运算， 再将结果写入到另一个内存单元。  
 Triad的中文含义是将三个组合起来，在本测试中表示的意思是将Copy、Scale、Add三种操作组合起来进行测试。具体操作方式是：先从内存单元中中读两个值a、b，对其进行乘加混合运算（a + 因子 \* b ） ，将运算结果写入到另一个内存单元。

**小页面模式（i.e. 普通的4K页面大小模式）：**Add > Copy > Triad > Scale，这是为什么？一次Add操作需要访问三次内存（两个读操作，一个写操作），Triad操作也需要三次访问内存，Copy和Scale操作需要两次访问内存。单位操作内，访问内存次数越多，越能够掩盖访存延迟，带宽越大。单位操作内，操作越复杂，操作完成时间越长，导致整个操作循环完成的时间越长。Add操作简单且访存次数多，故而带宽最大，Scale操作复杂且访存次数少，故而带宽最小。Copy操作简单但访存次数少，Triad操作复杂但访存次数多，考虑到编译器循环展开的作用，Copy操作能够更快地执行，最终导致Copy带宽比Triad略大。

**大页面模式：**基本规律一致。由于使用了大页面，每访问16M页面才会出现一次物理缺页，相比每访问4K页面就出现一次缺页来说，访存缺页的概率大大降低。这里需要注意的是，TLB缺失开销在本实验中是无法观测到的。这是因为测试中缺页开销占主导地位，访问过的内存不会被重新访问（时间局部性几乎没有），TLB缺失开销几乎可以不计。

## STREAM安装

下载地址

#cd /data

#git clone <https://github.com/jeffhammond/STREAM>

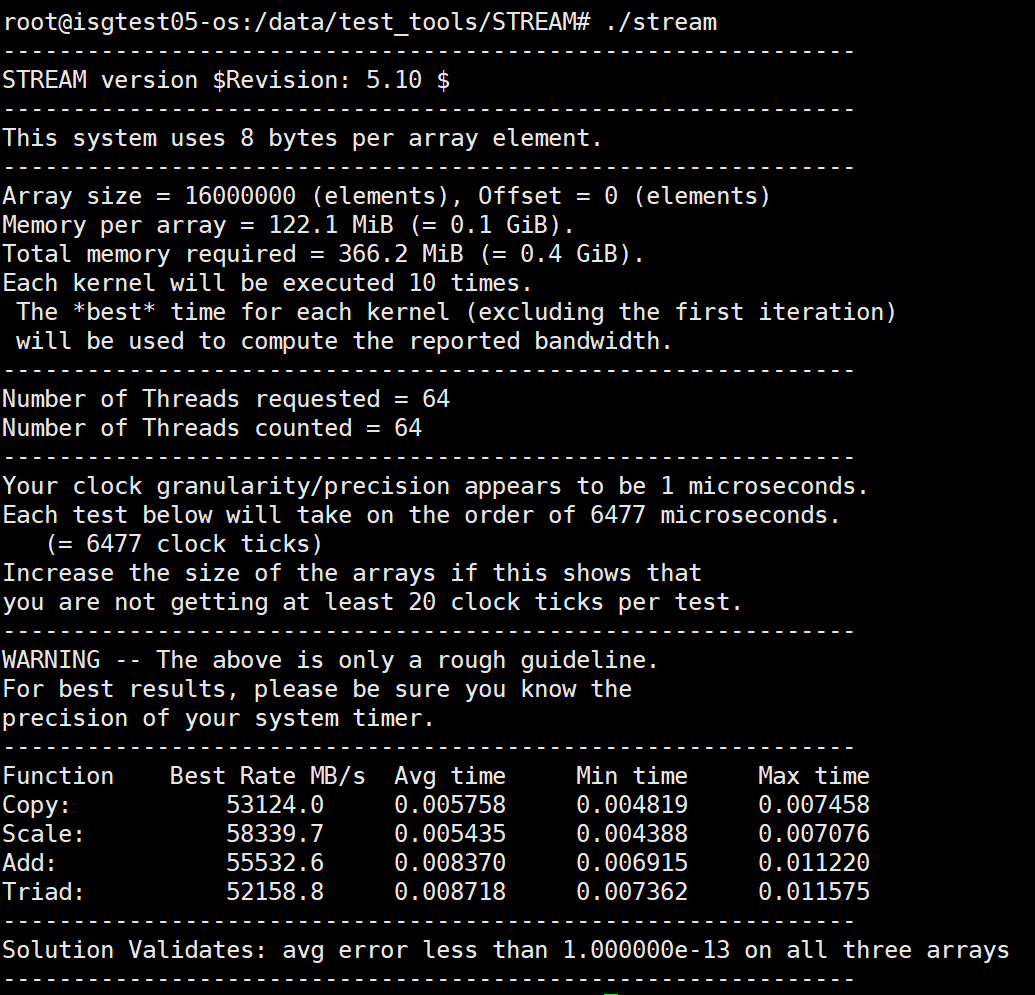
#cd /data/STREAM

## STREAM测试

### 3.1单线程测试

#gcc -O -fopenmp -DSTREAM\_ARRAY\_SIZE=16000000 -DNTIME=20 -mcmodel=large stream.c -o stream

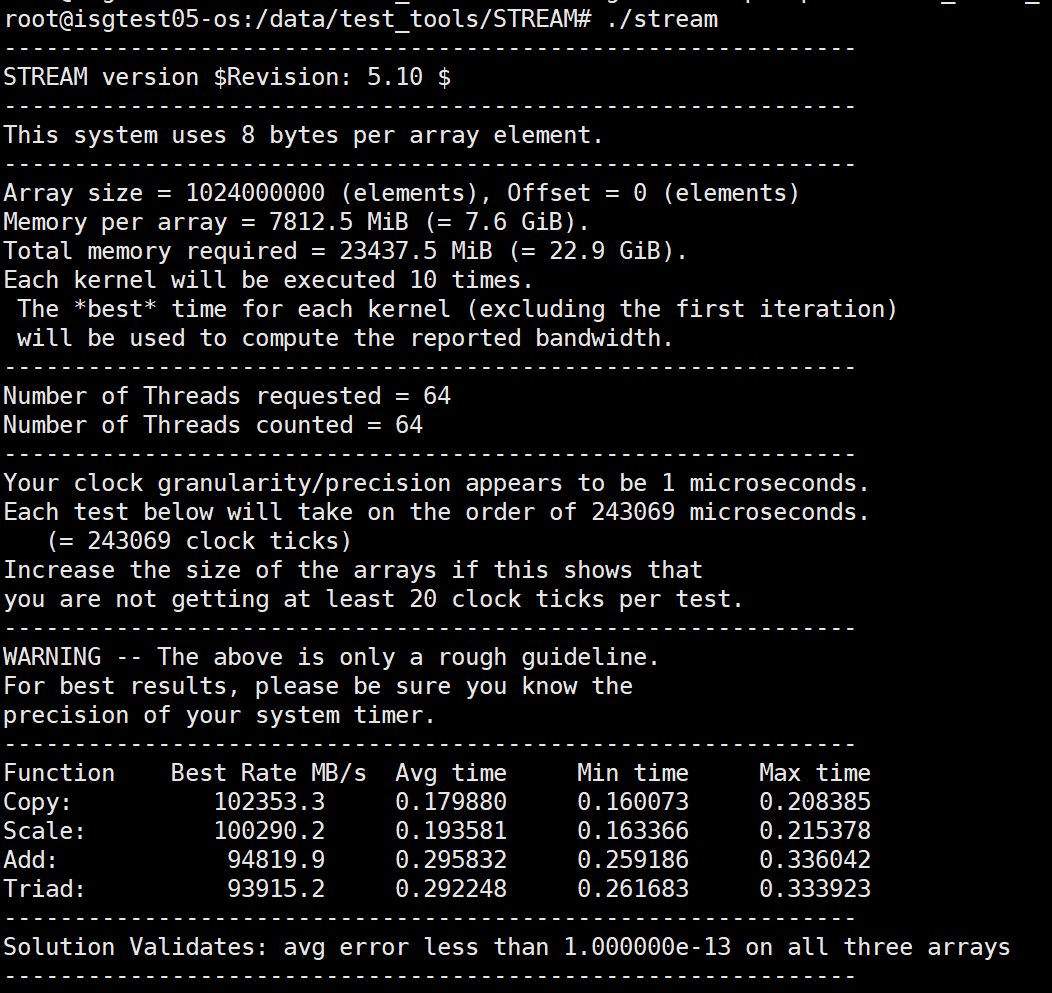
#./stream



### 3.2多线程测试

#gcc -O -fopenmp -DSTREAM\_ARRAY\_SIZE=1024000000 -DNTIME=20 -mcmodel=large stream.c -o stream

#./stream

STREAM参数说明

* **STREAM\_ARRAY\_SIZE**

STREAM\_ARRAY\_SIZE。这个是测试数据集的大小，该大小应该遵循以下两条规则。  
        a、数据集大小应不小于L3 cache大小的4倍。举例来说某10核机器中L3 cache为8MB/core，共80MB L3 cache，因此数据集的大小至少为80MB\*4=320MB。由于数据集中每个元素大小为64bits，即8B。故，数据集大小应设置为不小与320MB/8B=40M （40million或40000000）。32\*4==12800000  
        b、数据集大小应能确保程序输出时间大于20个时钟周期。该时钟周期可在程序输出信息中看到，如“Your clock granularity/precision appears to be 1 microseconds.“ 表示时钟周期为1微秒，20个时钟周期为20微秒。如果你的测试机器有200GB/s的带宽，那你的数据集大小应不小于4MB，即0.5million个元素。

* **NTIME**

NTIME。该参数为kernel执行的次数，程序将输出除第一次外其他结果中最好的结果，所以NTIME必须要大于1。该值默认为10，通常不需要修改。

* **OFFSET**

OFFSET。该值为数组的偏移量，修改此值可改变数组的对齐，从而在一定程度上改变输出的性能结果。一定程度在这指的是也许会改变，也许不会改变。如果需要修改该参数，通常将其设置为靠近2^n的数，例如使用-DOFFSET=1022 （靠近2^10=1024）。

* **STREAM\_TYPE**

我们可以通过修改该参数设置测试集的数据类型，默认是double(8B)。如果将其改为float则数据集大小减少一半。

* **Mcmmodel**

如果设置的STREAM\_ARRAY\_SIZE太大（大于2G），可以通过增加编译使用-mcmodel=large参数

-mcmodel=large

-mcmodel=small

-mcmodel=tiny

# LMBENCH使用说明

## LMBENCH简介

Lmbench用于测试OS提供的基本系统调用的性能，主要衡量两个关键特征：延迟（反应时间）和带宽。

Lmbench的主要功能

带宽测评工具：读取缓存文件、拷贝内存、读内存、写内存、管道、TCP。

延时测评工具：上下文切换、网络（连接的建立，管道，TCP，UDP和RPC hot potato）、文件系统的建立和删除、进程创建、信号处理、上层的系统调用、内存读入反应时间 。

其他：处理器时钟比率计算。

## LMBENCH安装

### 2.1下载与解压

#wget <http://www.bitmover.com/lmbench/lmbench3.tar.gz>

#tar xvf lmbench3.tar.gz

#cd lmbench3

#mkdir SCCS

#touch  SCCS/s.ChangeSet

### 2.2编译配置

# make results OS=arm-linux

**运行测试，此时会进行测试配置**

相关参数配置如下：

* MULTIPLE COPIES,同时运行并行测试数量，使用默认值
* Job placement selection,作业调度控制方法，选1允许作业调度，选择默认值
* MB(使用内存) 测试时需要使用的内存大小，建议不超过1G
* SUBSE，所要运行的子集，包括ALL/HARWARE/OS/DEVELOPMENT，选择默认值
* 用默认值
* FASTMEM,用默认值
* SLOWFS(文件系统延迟)，用默认值
* DISKS(磁盘测试)，用默认值
* REMOTE(远程网络测试)，用默认值
* Processor Mhz(CPU频率)，用默认值
* FSDIR(存放测试文件的路径)，用默认值
* Status output file(测试输出)，/dev/tty，用默认值
* Mail results,设置为no

### 2.3查询结果

#make see

cd results && make summary >summary.out 2>summary.errs

cd results && make percent >percent.out 2>percent.errs

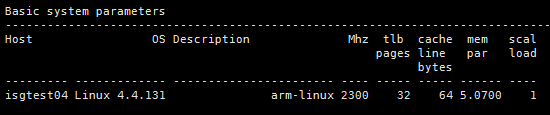
### 2.4再次运行

**#make rerun(不需要重新配置)**

## LMBENCH结论分析

测试结果包含主机各种测试的速度或者延迟，单位-微秒。

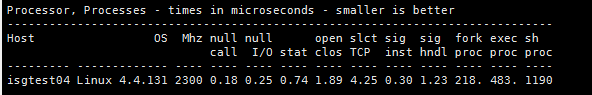
### 3.1系统基本参数



* Tlb pages: 转换后备缓存的页面数；
* Cache line bytes: 高速缓存行字节数
* mem par: 存储器分层并行化
* scal load：并行执行的Lmbench数目

### 3.2处理器Process性能

性能指标：【单位：μs，值越小性能越好】

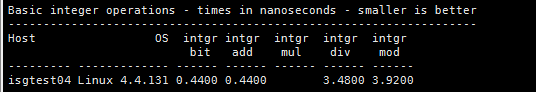


* null call: 执行getppid需要的时间；
* null I/O： 从/dev/zero读一个字节的时间长A，写一个字节到/dev/null需要的时间长B,A和B取平均得本值；
* stat：stat一个文件（即得到一个文件的信息）需用的时间；
* open close： open一个文件然后再close它总共需用的时间（不包括读目录和节点的时间）； selct TCP：通过TCP网络连接选择100个文件描述符所耗用的时间；
* sig inst： install signal handler所耗用的时间；
* sig hndl： catch signal 所耗用的时间；
* fork proc: fork一个完全相同的process，并把原来的process关掉所耗用的时间。
* exec proc：模拟一个shell进程的工作过程：fork一个新进程执行新命令，所耗用时间。
* sh proc：模拟最常见情况：fork一个新进程，同时询问系统shell来找到并运行一个新程序所耗用时间。

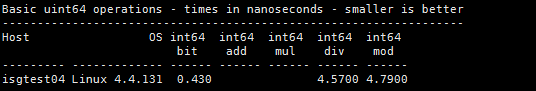
### 3.3数学运算

性能指标：【单位：ns，值越小性能越好】

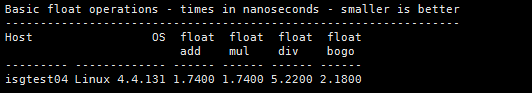
整数



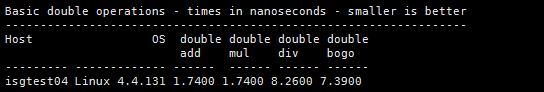
无符号整形



浮点型

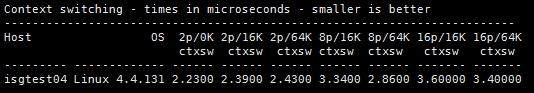


双精度型



### 3.4上下文切换

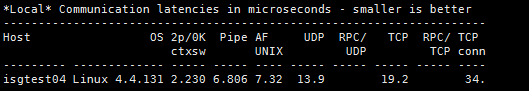
性能指标：【单位：μs，值越小性能越好】



* 多个进程用unix pipe环连接起来，每个进程从自己的管道中读取token，执行任务，然后将token写给下一个进程。
* context swithing时间包括：切换进程的时间，加上恢复进程所有状态所用时间（包括恢复cache状态）。
* 2p/0K： 每个进程的size为0（不执行任何任务），进程数为2时上下文切换耗用的时间；
* 2p/16K：每个进程的size为16K（执行任务），进程数为2时上下文切换耗用的时间；

### 3.5本地通讯延时

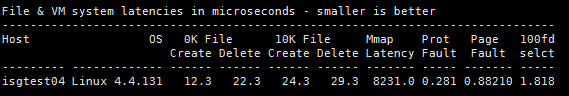
性能指标：【单位：μs，值越小性能越好】



* 2p/0K： 每个进程的size为0（不执行任何任务），进程数为2时上下文切换耗用的时间；
* Pipe：所谓的hot potato测试：两个没有具体任务的进程用unix pipe通信，一个token在两个进程间来回传递，传递一个来回所耗用的平均时间；
* AF UNIX：同Pipe，不同的是两个进程采用unix socket通信。
* UDP：同Pipe，不同的是两个进程采用UDP/IP 通信；
* RPC/UDP：同Pipe，不同的是两个进程采用sun RPC 通信；默认情况下，RPC通过udp协议传送。
* TCP：同Pipe，不同的是两个进程采用TCP/IP；
* RPC/TCP：同Pipe，不同的是两个进程采用sun RPC 通信；指定RPC通过tcp协议传送。
* TCP conn：创建一个AF\_INET (aka TCP/IP) socket，并连接到远程主机所耗用的时间，这个时间仅指创建socket和建立连接本身，不包括解析主机名等等其他动作所用时间。

### 3.6文档、内存延时

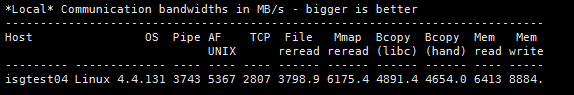
性能指标：【单位：μs，值越小越好】



* 0k create: 0k文件创建所花的时间；
* 0k Delete: 0k文件删除所花的时间；
* Mmap Latency: 将指定文件的开头n个字节map到内存，然后umap，并记录每次map和umap共耗用的时间；记录的是每次耗用时间的最大值；
* Prot Fault: 保护页延时时间；
* Page Fault: 缺页延时时间；
* 100fd selct: 对100个文档描述符配置select的时间；

### 3.7本地通信带宽

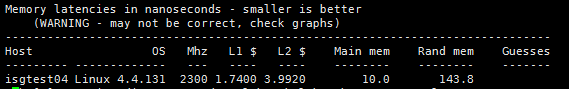
性能指标：【单位：MB/S，值越大越好】



* Pipe：在两个进程间建立一个unix pipe，pipe的每个chunk为64K，通过该管道移动50M数据所用的时间。
* AF UNIX：在两个进程间建立一个unix stream socket，每个chunk为64K，通过该socket移动10M数据所用的时间。
* TCP：同Pipe，不同的是进程间通过TCP/IP socket 通信，传输的数据为3MB；
* File reread：读文件并把他们汇总起来所用的时间；
* Mmap reread：将文件map到内存中，从内存中读文件并把他们汇总起来所用的时间；
* Bcopy(libc)： do bw\_mem $i bcopy，从指定内存区域拷贝指定数目的字节内容到指定的另一个内存区域的速度；
* Bcopy(hand)：do bw\_mem $i fcp，把数据从磁盘上一个位置拷贝到另一个位置所用的时间；
* Mem read：bw\_mem $i frd，累加数组中的整数值，测试把数据读入processor的带宽；
* Mem write：do bw\_mem $i fwr，把整数数组的每个成员设置为1，测试写数据到内存的带宽；

### 3.8内存操作延时

性能指标：【单位：ns，值越小性能越好】



* L1 $：输出结果中以 stride=128 开头的段中，首字段（size）为 0.00098 行的第二字段；
* L2 $：输出结果中以 stride=128 开头的段中，首字段（size）为 0.12500行的第二字段；
* Main mem：最后行的第二字段值；
* Rand mem：系统内存随机访问操作延时；
* Guesses：判断前面得到的L1和L2值差值占其中最大值的百分比，如果大于20%，且L2和Main mem差值占其中最大值的百分比小于20%，则输出”No L2 cache“；否则，判断L1和L2差距与其中最大值的百分比，如果小于20%，则输出”No L1 cache“；否则什么也不输出。

# LTP使用说明

## Ltp简介

LTP（LinuxTest Project）是SGI、IBM、OSDL和Bull合作的项目，目的是为开源社区提供一个测试套件，用来验证Linux系统可靠性、健壮性和稳定性。LTP测试套件是测试Linux内核和内核相关特性的工具的集合。该工具的目的是通过把测试自动化引入到Linux内核测试，提高Linux的内核质量。LTP提供了验证linux系统稳定性的标准，设计标准的压力场景，通过对linux系统进行压力测试，对系统的功能、性能进行分析，并以此确定linux系统的可靠性、健壮性和稳定性。

压力测试是一种破坏性的测试，即系统在非正常的、超负荷的条件下的运行情况 。用来评估在超越最大负载的情况下系统将如何运行，是系统在正常的情况下对某种负载强度的承受能力的考验。

使用LTP测试套件对Linux操作系统进行超长时间的测试，重点在于Linux用户环境相关的工作负荷。而并不是致力于证明缺陷。

### 1.1 软件测试设计

**压力测试的4个设计重点：测试选择、评价系统资源利用率、分析内核代码覆盖率、评价最终压力测试**

**1.测试选择**

测试应该可以得到 CPU（s）、内存、I/O 和网络等主要内核区域的高水平的资源利用率。

测试应该充分地覆盖内核代码，以帮助支持自其结果中生成的稳定性声明。

**2.评价系统资源利用率**

　　所选择的测试的组合必须给系统的资源带来足够的压力。Linux内核的 四个主要方面可以影响系统的响应和执行时间：

* CPU：用于在机器的CPU（s）上处理数据的时间。
* Memory：用于真实存储器中读写数据的时间。
* I/O：用于磁盘存储器读写数据的时间。
* Networking：用于网络读写数据的时间。

系统资源利用率评价阶段通常需要多次尝试才能得到合适的测试组合，并得到期望水平的利用率。当确定测试组合时，过度利用总是一个至关重要的问题。例如，如果选择的组合过于受 I/O 所限，可能会导致CPU的测试结果不好，反之亦然。测试主要是大量的试验和出错，直到所有资源达到期望水平。

当选定一个组合后，测试必须长时间运行以准确评价资源的利用率。测试运行的时间长短取决于每个测试的长度。假如多个测试同时运行，则时间必须足够长以使得这些测试中最长的那个可以完成。在这个评价过程中，sar工具也应该在运行。在评价运行的结论中，您应该收集并评价所有四种资源的利用率水平。

**3.分析内核代码覆盖率**

获得足够的内核覆盖率是系统压力测试的另一个职责。尽管所选的测试组合充分地利用了四种主要资源，它也有可能只是执行了内核的一小部分。因而，应该对覆盖率进行分析以确保组合可以成为一个系统压力测试，而不是一个系统负载生成器。

**4. 评价最终压力测试**

之所以要执行方法中的最后一步，是为了对系统压力测试进行核实。在一个被认为是稳定的内核上执行压力测试，通常发行版本中的内核可以满足这一要求，但不总是如此。要长时间地执行压力测试，同时运行sar工具，原因有以下两点：

* 长时间运行有助于发现组合中的所有问题，否则，在短时间的“取样测试（sniff test）”中这些问题可能会被忽略。
* sar 生成的数据构成以后测试运行中进行比较的基线。

长时间运行结束后，现在可以基于收集的所有数据来决定这个测试组合是否是系统压力测试的合适候选者。

### 1.2 软件测试方法

**测试方法有两个阶段:初始测试、压力测试**

**1.初始测试**

这一步是开始测试的必要条件。初始测试包括LTP测试套件在硬件和操作系统上成功运转，这些硬件和操作系统将用于可靠性运转。LTP测试套件包附带的驱动程序脚本runalltests.sh用于验证内核。这个脚本串行地运行一组成包的测试，并报告全部结果。也可以选择同时并行地运行几个实例。在执行runltp脚本的时候，可以指定参数添加需要测试的项目（在testscripts内），初始测试的测试脚本是runalltests.sh或runltp(runltp默认执行的内容与runalltests相同)，默认这个脚本执行如下测试内容：

* 文件系统压力测试。
* 硬盘 I/O 测试。
* 内存管理压力测试。
* IPC 压力测试。
* SCHED 测试。
* 命令功能的验证测试。
* 系统调用功能的验证测试。

**2.压力测试**

这一步可以验证产品在系统高使用率时的健壮性。作为runalltests.sh的补充，特别设计了一个名为ltpstress.sh的测试场景,在使用网络与内存管理的同时并行地运行大范围的内核组件,并在测试系统上生成高压力负荷。

ltpstress.sh也是LTP测试套件的一部分。这个脚本并行地运行相似的测试用例，串行地运行不同的测试用例，这样做是为了避免由于同时访问同一资源或者互相干扰而引起的间歇性故障。测试内容与runltp相同，不同点在于runltp可以指定测试项进行组合测试，而runalltests.sh则会全部执行。默认这个脚本执行如下测试内容：

* NFS压力测试。
* 内存管理压力测试。
* 文件系统压力测试。
* 数学(浮点)测试。
* 多线程压力测试。
* 硬盘I/O测试。
* IPC (pipeio、semaphore)测试。
* 系统调用功能的验证测试。
* 网络压力测试。

LTP工作组在设计Linux 内核压力测试脚本 ltpstress.sh 时使用了这一设计方法，为给系统提供足够的压力，LTP工作组对这个组合测试进行了分析，以确定 Linux 内核的哪些部分在测试执行中得到了使用。然后，修改了组合测试，在保持期望的高强度系统压力的同时提高代码覆盖率的百分比。最终得到的压力测试涵盖了Linux 内核的足够多部分，有助于稳定性声明，并且有系统使用情况和内核代码覆盖情况的数据来支持它。

有两个开放源代码工具可以帮助进行 Linux 内核的代码覆盖率分析：

* gcov：一个由 LTP 维护的开放源代码工具。这个工具分析内核代码的覆盖率，并报告哪些行、函数和分支被覆盖以及它们被访问了多少次。
* lcov：另一个由 IBM 开发，由 LTP 维护的开放源代码工具。 这个工具由一组构建于基于文本的 gcov 输出之上的 Perl 脚本构成，以实现基于 HTML 的输出。输出包括覆盖率百分比、图表以及概述页，可以快速浏览覆盖率数据。可以自LTP主页找到这两个工具。

lcov 工具会生成一棵完整的HTML 树，其中包含有内核中代码的每一行以及关于每一行执行了多少次的数据（如果有的话）。这个工具会量化覆盖率数据并生成关于内核中每一部分和文件覆盖率的百分比数字。

内核的代码覆盖率分析只是在ltpstress.sh的设计和开发过程中用到，目的是保证ltpstress.sh的可用性，我们在实际测试的时候就不需要再做内核的代码覆盖率分析了。

### 1.3系统监控

LTP测试套件附带的top工具是经过修改的，用作系统监控工具。使用 top 可以实时地观察处理器的行为。改进的top工具附加的功能可以将top结果的快照保存到文件中，并给出结果文件的平均总结，包括 CPU、内存和交换空间利用率等信息。

在我们的测试中，sar工具每10秒钟截取一次系统利用率的快照，并保存到结果文件。

测试之前所有选定的测试系统的硬件配置尽可能相同。去掉额外的硬件以减少潜在的硬件故障。在映像安装过程中选择最低的安全选项。预留至少2GB的硬盘空间以保存top数据文件和LTP日志文件。

在测试期间系统不要受到干扰。偶尔访问一下系统以确认测试仍在进行是可以接受的。确认的手段包括使用ps命令、检查top数据和检查LTP日志数据。

**源安装包目录列表说明：**

doc：该目录是说明文件和帮助文档的所在地，这个目录中对LTP的内容和每个工具都有详细的说明。

testscripts：该目录中存储的是可执行的测试脚本,不同方面的测试脚本的集合。

testcases：该目录存储了所有LTP测试套件中所使用的测试用例的源码。

runtest：该目录中的每个文件都是要执行的测试用例的命令集合,每个文件针对测试的不同方面（用于链接testscripts内的测试脚本和testcases测试项目）。

include：LTP测试套件的头文件目录,定义了LTP自身的数据结构和函数结构。

lib：LTP测试套件运行时自身需要的库文件,定义了LTP自身的各种函数。

bin：存放LTP测试的一些辅助脚本。

results：测试结果默认存储目录。

output：测试日志默认存储目录。

share：脚本使用说明目录。

pan：该目录存储的是LTP测试套件的测试驱动程序。

pan工作原理：LTP测试套件有一个专门的测试驱动程序，具体的测试用例的执行都是由pan来调用执行，它可以跟踪孤儿进程和抓取测试的输出信息。它的工作方式是这样的：

从一个测试命令文件中读取要测试的条目和要执行的命令行，然后等待该项测试的结束，并记录详细的测试输出。默认状态下pan会随机的选择一个命令行来运行，可以指定在同一时间要执行测试的次数。

pan会记录测试产生的详细的格式复杂的输出，但它不进行数据的整理和统计，数据整理统计的工作由scanner来完成，scanner是一个测试结果分析工具，它会理解pan的输出格式，并输出成一个表格的形式来总结那些测试passed或failed。

LTP测试套件通过执行测试脚本runalltests.sh(或runltp或runltplite.sh)或testscripts内的测试脚本调用驱动程序pan执行testcases内的测试项目。

**源安装包文件列表说明:**

IDcheck.sh：检查系统是否缺少执行LTP测试套件所需的用户和用户组，如果缺少则为LTP测试套件创建所需的用户和用户组。

runltplite.sh：这个脚本用来测试LTP安装，也可用来对测试套件的子项目进行测试。

ver\_linux：这个脚本是获取硬件、软件、环境信息。

## Ltp安装

### 2.1下载与ltp源码包

#wget [https://github.com/linux-test-project/ltp/archive/ltp-20160510.tar.gz](https://github.com/linux-test-project/ltp/archive/20190930.tar.gz)

#tar xvf ltp-20160510.tar.gz

#cd ltp-20160510

### 2.2编译安装ltp

#make autotools

#./configure

#make

#make install

不指定安装路径的话,将会默认安装到/opt/ltp目录（在这个目录运行文件就可以了）

## Ltp初始测试

### 3.1运行初始测试

#./runltp -p -l /tmp/resultlog.20191121 -d /tmp/ -o /tmp/ltpscreen.20191121 -t 1h

或者

#./runalltests.sh

**参数说明：**

-p:人为指定日志格式,保证日志为可读格式

-l:记录测试日志的文件

-d:指定临时存储目录，默认为/tmp

-o:直接打印测试输出到/tmp/ltpscreen.20191121

-t:指定测试的持续时间

-t 60s = 60 seconds

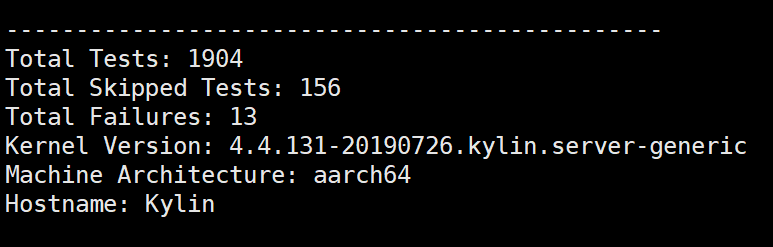
      -t 45m = 45 minutes

      -t 24h = 24 hours

      -t 2d  = 2 days

### 3.2查看初始测试结果

#cat /tmp/resultlog.20191121



## Ltp压力测试

### 4.1执行压力测试

#cd /opt/ltp/testscripts

#./ltpstress.sh -d /tmp/ltpstress.data -l /tmp/ltpstress.log -I /tmp/ltpstress.iostat  -i 5 -t 1 -S

**参数说明：**

-d:指定sar或top记录文件,默认/tmp/ltpstress.data

-l:记录测试结果到/tmp/ltpstress.log

-I:记录"iostat"结果到iofile,默认是/tmp/ltpstress.iostat

-i:指定sar或top快照时间间隔,默认为10秒

-m:指定最小的内存使用,默认为: RAM + 1/2 swap

-n:不对网络进行压力测试

-S:用sar捕捉数据

-T:利用LTP修改过的top工具捕捉数据

-t: 指定测试时间，默认为小时

### 4.2查看压力测试结果

默认情况下，测试结果放在 /tmp

ltpstress.log ---- 记录相关日志信息，主要是测试是否通过(pass or fail)

ltpstress.data ---- sar工具记录的日志文件，包括cpu,memory,i/o等

ltpstress.5010.output1 ---- 对应stress.part1，测试命令的一些输出信息

ltpstress.5010.output2 ---- 对应stress.part2，测试命令的一些输出信息

ltpstress.5010.output3 ---- 对应stress.part3，测试命令的一些输出信息

## Ltp其他常规测试命令

#cd /opt/ltp

1. ./runltp –f commands(测试常规命令)
2. ./runltp –f admin\_tools(测试常用管理工具是否正常稳定运行)
3. ./runltp –f dio(测试直接IO是否正常稳定)
4. ./runltp –f  dma\_thread\_diotest(测试直接存储器访问线程直接IO是否正常稳定)
5. ./runltp –f  fcntl-locktests(测试NFS网络文件系统锁是否正常稳定)
6. ./runltp –f  filecaps(测试filecaps是否正常)

预制条件:在/etc/sysctl.conf文件中加一行：CONFIG\_SECURITY\_FILE\_CAPABILITIES=y后重启电脑

1. ./runltp –f fs(测试文件系统是否正常)
2. ./runltp –f  fs\_bind(测试fs\_bind是否正常)
3. ./runltp –f fs\_ext4(测试fs\_ext4是否正常)
4. ./runltp –f fs\_perms\_simple(简单测试文件系统权限)
5. ./runltp –f  fs\_readonly(测试文件系统只读)
6. ./runltp –f fsx(对文件系统进行压力测试)
7. ./runltp –f hyperthreading(CPU超线程技术测试)
8. ./runltp –f io(异步IO测试)
9. ./runltp –f  io\_cd(对CD光驱进行压力测试)

预制条件:将光盘放入光驱

1. ./runltp –f  io\_floppy(对软盘进行压力测试)

预制条件:将软盘放入软驱中

1. ./runltp –f  lvm.part1(测试文件系统MSDOS、Reiserfs、EXT2、NFS、Ram Disk、MINIX)
2. ./runltp –f  math(数学库测试)
3. ./runltp –f  nfs(nfs网络文件系统测试)

预制条件:在本机配置nfs文件系统服务

1. ./runltp –f  lvm.part2(测试EXT3、JFS文件系统是否正常使用)

预制条件:安装EXT3、JFS文件系统

21../runltp –f pipes(对管道进行压力测试)

22. ./runltp –f syscalls(测试内核系统调用)

23. ./runltp –f syscalls-ipc(进程间通信测试)

24. ./runltp –f can(测试控制器区域网络的稳定性)

25. ./runltp –f connectors(测试Netlink Connector的功能性及稳定性)

26. ./runltp –f ipv6(测试IPv6环境下的基本网络功能)

27. ./runltp –f ipv6\_lib(IPv6环境网络开发共享库)

预制条件:内核支持IPv6

1. ./runltp –f multicast（ 测试多播的稳定性）

预制条件：

（1）设置环境变量export RHOST=<多播目标地址>

（2）/root/.rhosts,/home/user/.rhosts 添加内容：多播目标主机地址，多播目标主机用户，有多少多播目标主机就写多少条。

1. ./runltp –f network\_commands （ 测试ftp和ssh的稳定性）

预制条件：开启ftp和ssh

30. ./runltp –f network\_stress.whole（ 网络各个功能的压力性测试 ）

预制条件：

（1）部署一台服务器

（2）服务器上运行的服务： ssh DNS http ftp

31. ./runltp –f nptl（ 测试本地POSIX线程库的稳定性 ）

预制条件：内核支持POSIX本地线程库

1. ./runltp –f nw\_under\_ns（测试网络命名空间的稳定性）
2. ./runltp –f power\_management\_tests（电源管理模块的稳定性）

预制条件：内核版本2.6.31以上

1. ./runltp –f pty（测试虚拟终端稳定性）

预制条件：内核支持VT console

1. ./runltp –f quickhit（测试系统调用的稳定性）
2. ./runltp –f rpc 和 ./runltp –f rpc\_test（测试远程过程调用稳定性）

预制条件：内核支持远程过程调用

1. ./runltp –f scsi\_debug.part1（测试SCSI的稳定性）
2. ./runltp –f sctp（测试SCTP协议的稳定性）

预制条件：内核支持SCTP协议

1. ./runltp –f tcp\_cmds\_expect（TCP命令的可用性和稳定性）

预制条件：内核支持TCP/IP协议

1. ./runltp –f controllers（内核资源管理的稳定性测试）

预制条件：内核版本必须等于或者高于2.6.24

1. ./runltp –f cap\_bounds（POSIX功能绑定设置可用性)

预制条件：内核版本2.6.25以上

1. ./runltp –f containers（命名空间资源稳定性）
2. ./runltp –f cpuacct（测试不同cpu acctount控制器的特点）
3. ./runltp –f cpuhotplug（测试cpu热插拔功能的稳定性）
4. ./runltp –f crashme（测试crashme）

预制条件：做测试前，先备份系统

1. ./runltp –f hugetlb（测试 hugetlb）
2. ./runltp –f ima（测试ima）
3. ./runltp –f ipc（测试ipc）
4. ./runltp –f Kernel\_misc（测试 Kernel\_misc）
5. ./runltp –f ltp-aiodio.part1（测试 ltp-aiodio.part1）
6. ./runltp –f Ltp-aiodio.part2（测试 Ltp-aiodio.part2）
7. ./runltp –f ltp-aiodio.part3（测试 ltp-aiodio.part3）
8. ./runltp –f ltp-aiodio.part4（测试 ltp-aiodio.part4）
9. ./runltp –f ltp-aio-stress.part1（测试 io stress）
10. ./runltp –f ltp-aio-stress.part2（测试 io stress）
11. ./runltp –f mm（测试mm）
12. ./runltp –f modules（测试内核模块）
13. ./runltp –f numa（测试非统一内存访问）
14. ./runltp –f sched（测试调度压力）
15. ./runltp –f securebits（测试securebits）
16. ./runltp –f smack（smack安全模块测试）
17. ./runltp –f timers（测试posix计时器）
18. ./runltp –f tirpc\_tests（测试Tirpc\_tests）
19. ./runltp –f tpm\_tools（测试 tpm\_tools）

66../runltp –f tracing（跟踪测试）

# IOzone使用说明

## IOzone简介

### 1.1软件简介

iozone（www.iozone.org)是一个文件系统的benchmark（基准）工具，主要测试的是文件系统的各种读写性能，包含顺序读写以及随机读写，fread/fwrite读写，大跨度读写等等，在测试文件的读写性能方面，测试范围广，指标精确。

磁盘设备之上是文件系统，测试磁盘的工具往往就是调用驱动块设备驱动的接口进行读写测试。而文件系统的测试软件就是，针对文件系统层提供的功能进行测试，包括文件的打开关闭速度以及顺序读写，还有随机位置的读写以及进程并发数目等各个方面进行详细的测试。

### 1.2测试读写模式

|  |  |
| --- | --- |
| **测试项** | **说明** |
| Write | 测试向一个新文件写入的性能。当一个新文件被写入时，不仅仅是那些文件中的数据需要被存储，还包括那些用于定位数据存储在存储介质的具体位置的额外信息。这些额外信息被称作“元数据”。它包括目录信息，所分配的空间和一些与该文件有关但又并非该文件所含数据的其他数据。拜这些额外信息所赐，Write的性能通常会比Re-write的性能低。 |
| Re-write | 测试向一个已存在的文件写入的性能。当一个已存在的文件被写入时，所需工作量较少，因为此时元数据已经存在。Re-write的性能通常比Write的性能高。 |
| Read | 测试读一个已存在的文件的性能。 |
| Re-Read | 测试读一个最近读过的文件的性能。Re-Read性能会高些，因为操作系统通常会缓存最近读过的文件数据。因为系统缓存可以被用于读以提高性能。 |
| Random Read | 测试读一个文件中的随机偏移量的性能。许多因素都可能影响这种情况下的系统性能，例如：操作系统缓存的大小，磁盘数量，寻道延迟和其他。 |
| Random Write | 测试写一个文件中的随机偏移量的性能。同样，有许多因素可能影响这种情况下的系统性能，例如：操作系统缓存的大小，磁盘数量，寻道延迟和其他。 |
| Random Mix | 测试读写一个文件中的随机偏移量的性能。许多因素可能影响这种情况下的系统性能运作，例如：操作系统缓存的大小，磁盘数量，寻道延迟和其他。这个测试只有在吞吐量测试模式下才能进行。每个线程/进程运行读或写测试。这种分布式读/写测试是基于round robin 模式的。最好使用多于一个线程/进程执行此测试。 |
| Backwards Read | 测试使用倒序读一个文件的性能。这种读文件方法可能看起来很可笑，事实上有些应用确实这么干。MSC Nastran是一个使用倒序读文件的应用程序的一个例子。它所读的文件都十分大（大小从G级别到T级别）。尽管许多操作系统使用一些特殊实现来优化顺序读文件的速度，很少有操作系统注意到并增强倒序读文件的性能。 |
| Record Rewrite | 测试写与覆盖写一个文件中的特定块的性能。这个块可能会发生一些很有趣的事。如果这个块足够小（比CPU数据缓存小），测出来的性能将会非常高。如果比CPU数据缓存大而比TLB小，测出来的是另一个结果的性能。如果此二者都大，但比操作系统缓存小，得到的性能又是一个结果。若大到超过操作系统缓存，又是另一番结果。 |
| Strided Read | 测试跳跃读一个文件的性能。举例如下：在0偏移量处读4Kbytes，然后间隔200Kbytes,读4Kbytes，再间隔200Kbytes，如此反复。此时的模式是读4Kbytes，间隔200Kbytes并重复这个模式。这又是一个典型的应用行为，文件中使用了数据结构并且访问这个数据结构的特定区域的应用程序常常这样做。  许多操作系统并没注意到这种行为或者针对这种类型的访问做一些优化。同样这种访问行为也可能导致一些有趣的性能异常。一个例子是在一个数据片化的文件系统里，应用程序的跳跃导致某一个特定的磁盘成为性能瓶颈。 |
| Fwrite | 测试调用库函数fwrite()来写文件的性能。这是一个执行缓存与阻塞写操作的库例程。缓存在用户空间之内。如果一个应用程序想要写很小的传输块，fwrite()函数中的缓存与阻塞I/O功能能通过减少实际操作系统调用并在操作系统调用时增加传输块的大小来增强应用程序的性能。这个测试是写一个新文件，所以元数据的写入也是要的。 |
| Frewrite | 测试调用库函数fwrite()来写文件的性能。这也是一个执行缓存与阻塞写操作的库例程。是缓存在用户空间之内。如果一个应用程序想要写很小的传输块，fwrite()函数中的缓存与阻塞I/O功能可以通过减少实际操作系统调用并在操作系统调用时增加传输块的大小来增强应用程序的性能。这个测试是写入一个已存在的文件，由于无元数据操作，测试的性能会高些。 |
| Fread | 测试调用库函数fread()来读文件的性能。这是一个执行缓存与阻塞读操作的库例程。缓存在用户空间之内。如果一个应用程序想要读很小的传输块，fwrite()函数中的缓存与阻塞I/O功能能通过减少实际操作系统调用并在操作系统调用时增加传输块的大小从而增强应用程序的性能。 |

### 1.3主要参数说明

iozone常用的几个参数。

-a 全面测试，比如块大小它会自动加

-i N 用来选择测试项, 比如Read/Write/Random 比较常用的是0 1 2,可以指定成-i 0 -i 1 -i2.这些别的详细内容请查man

         N代表数字，不同的数字代表不同的测试模式，可以同时执行多个模式

|  |  |
| --- | --- |
| **0** | write/rewrite |
| **1** | read/re-read |
| 2 | random read/random write |
| 3 | backwards read |
| 4 | re-write-record |
| 5 | stride-read |
| 6 | fwrite/re-fwrite |
| 7 | fread/re-fread |
| 8 | random mix |
| 9 | pwrite/re-pwrite |
| **10** | pread/re-pread |
| **11** | pwritev/re-pwritev |
| **1**2 | preadv/re-preadv |

-r block size 指定一次写入/读出的块大小

-s file size 指定测试文件的大小

**测试的时候请注意，设置的测试文件大小一定要大过你的内存(最佳为内存的两倍大小)，不然linux会给你的读写的内容进行缓存。会使数值非常不真实。**

-f filename 指定测试文件的名字,完成后会自动删除(这个文件必须指定你要测试的那个硬盘中)

-F file1 file2... 指定多线程下测试的文件名

**批量测试项:**

-g -n 指定测试文件大小范围,最大测试文件为4G,可以这样写 -g 4G

-y -q 指定测试块的大小范围

**输出:**

下面是几个日志记录的参数.好象要输出成图象进行分析，需要指定-a的测试才能输出

-R 产生Excel到标准输出

-b 指定输出到指定文件上. 比如 -Rb ttt.xls

## IOzone安装

去官网<http://www.iozone.org/src/current/>下载iozone源码包，然后编译适合自己的target的执行文件

wget http://www.iozone.org/src/current/iozone3\_487.tar

这里我们以最新版本iozone3\_487.tar为例，可以根据自己的需求通过官网下载对应版本。

解压缩

tar -xvf iozone3\_487.tar

到安装目录下的src/current下执行make linux命令

cd iozone3\_487/src/current/

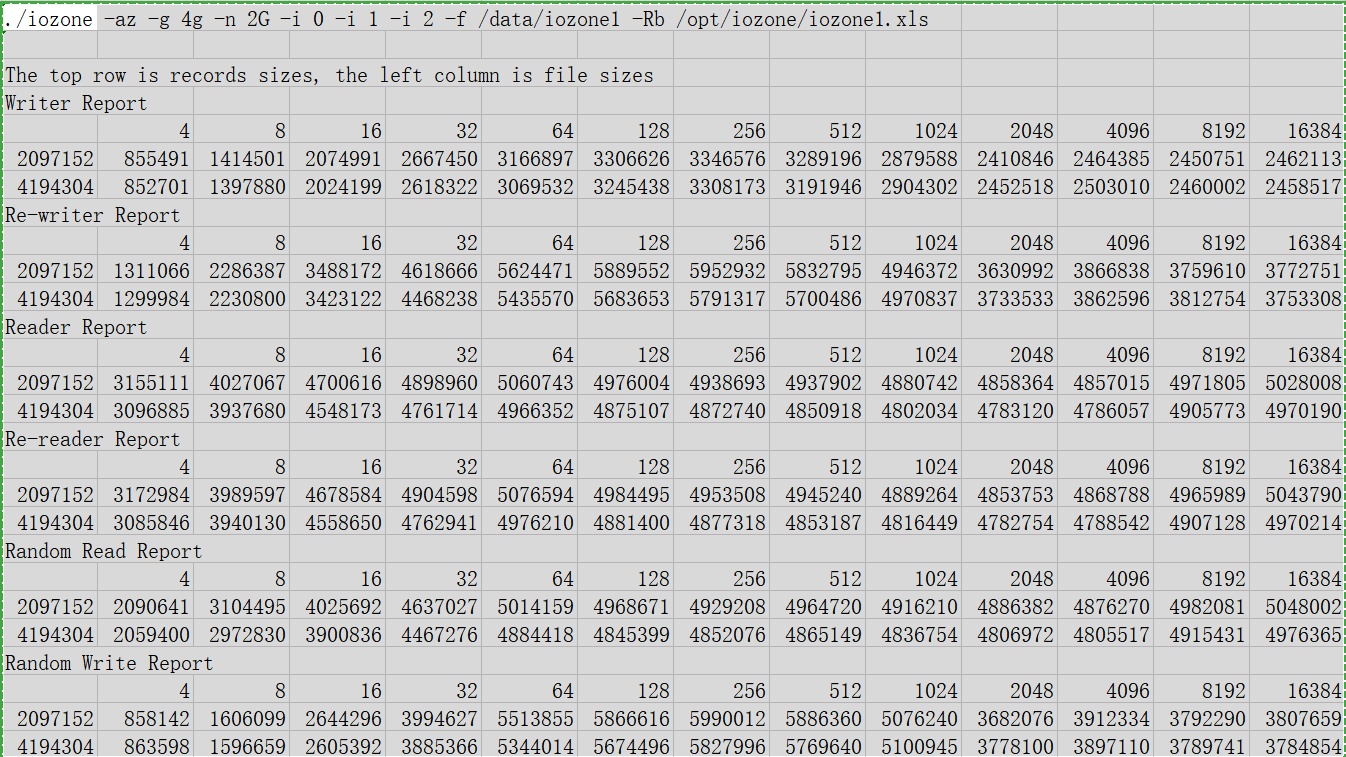
make linux

## IOzone测试

### 3.1 测试用例

#sudo ./iozone -az -g 4g -n 2G -i 0 -i 1 -i 2 -f /data/iozone1 -Rb /opt/iozone/iozone1.xls

### 3.2 结论与分析



**结论：**

1. 分别对Writer、Re-writer、Random Write、Random Read、Reader、Re-reader进行数据测试。
2. 每个Report有两行，分别是2097152以及4194304，代表测试文件大小分别为2G和4G。
3. 每个Report上面的4、8、16、......、16384，代表以4K、8K、......、16384K大小来进行传输。
4. 文件大小和传输单位大小的交叉部分为传输速度，单位为Kbytes/s。

**注意：**

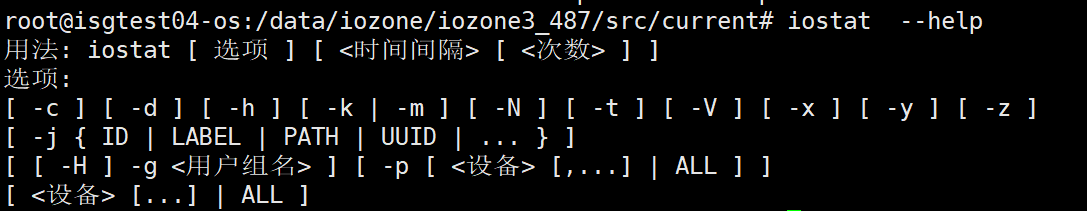
**测试的时候请注意，设置的测试文件的大小一定要大过你的内存(最佳为内存的两倍大小)，不然linux会给你的读写的内容进行缓存。会使数值非常不真实。**

## Iostat查询

iostat是I/O statistics（输入/输出统计）的缩写，iostat工具将对系统的磁盘操作活动进行监视。它的特点是汇报磁盘活动统计情况，同时也会汇报出CPU使用情况。iostat也有一个弱点，就是它不能对某个进程进行深入分析，仅对系统的整体情况进行分析。

### 4.1 iostat用法

用法：iostat [选项] [<时间间隔>] [<次数>]

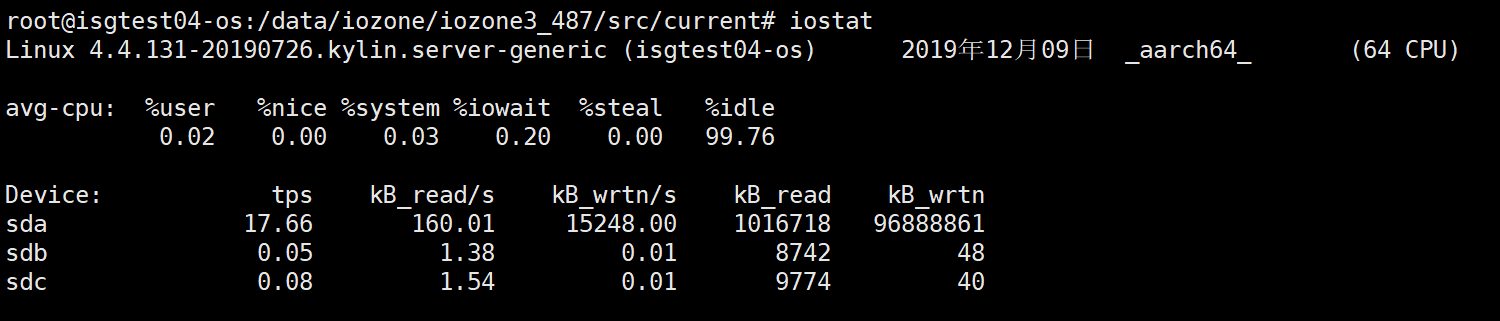


**命令参数：**

-c： 显示CPU使用情况  
 -d： 显示磁盘使用情况  
 -N： 显示磁盘阵列(LVM) 信息  
 -n： 显示NFS 使用情况  
 -k： 以 KB 为单位显示  
 -m： 以 M 为单位显示  
 -t： 报告每秒向终端读取和写入的字符数和CPU的信息  
 -V： 显示版本信息  
 -x： 显示详细信息  
 -p：[磁盘] 显示磁盘和分区的情况

### 4.2显示所有设备负载情况

#iostat



单独执行iostat，显示的结果为从系统开机到当前执行时刻的统计信息。

**以上输出中，包含三部分：**

| **选项** | **说明** |
| --- | --- |
| 第一行 | 最上面指示系统版本、主机名和当前日期 |
| avg-cpu | 总体cpu使用情况统计信息，对于多核cpu，这里为所有cpu的平均值 |
| Device | 各磁盘设备的IO统计信息 |

**Device中各列参数含义如下：**

| **选项** | **说明** |
| --- | --- |
| %user | CPU在用户态执行进程的时间百分比。 |
| %nice | CPU在用户态模式下，用于nice操作，所占用CPU总时间的百分比 |
| %system | CPU处在内核态执行进程的时间百分比 |
| %iowait | CPU用于等待I/O操作占用CPU总时间的百分比 |
| %steal | 管理程序(hypervisor)为另一个虚拟进程提供服务而等待虚拟CPU的百分比 |
| %idle | CPU空闲时间百分比 |

1. 若%iowait的值过高，表示硬盘存在I/O瓶颈   
 2. 若%idle的值高但系统响应慢时，有可能是CPU等待分配内存，此时应加大内存容量   
 3. 若%idle的值持续低于1，则系统的CPU处理能力相对较低，表明系统中最需要解决的资源是 CPU

**avg-cpu中各列参数含义如下：**

| **选项** | **说明** |
| --- | --- |
| Device | 设备名称 |
| tps | 每秒向磁盘设备请求数据的次数，包括读、写请求，为rtps与wtps的和。出于效率考虑，每一次IO下发后并不是立即处理请求，而是将请求合并(merge)，这里tps指请求合并后的请求计数。 |
| kB\_read/s | 取样时间间隔内每秒从设备（drive expressed）读取的数据量； |
| kB\_wrtn/s | 取样时间间隔内每秒向设备（drive expressed）写入的数据量 |
| kB\_read | 取样时间间隔内读扇区总数量 |
| kB\_wrtn | 取样时间间隔内写扇区总数量 |

### 4.3常用iostat命令

**定时显示所有信息**

#sudo iostat 2  3

以上命令输出Device的信息，采样时间为2秒，采样3次，若不指定采样次数，则iostat会一直输出采样信息，直到按”ctrl+c”退出命令。注意，第1次采样信息与单独执行iostat的效果一样，是从系统开机到当前执行时刻的统计信息。

**显示指定磁盘信息**

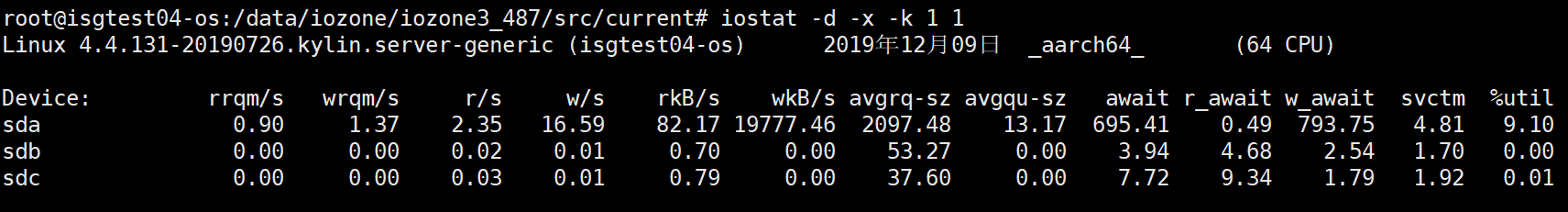
#sudo iostat -d /dev/sda

**以M为单位显示所有信息**

#sudo iostat -m

**查看设备使用率**

#sudo iostat -d -x -k 1 1



**以上各列的含义如下：**

| **选项** | **说明** |
| --- | --- |
| rrqm/s | 每秒对该设备的读请求被合并次数，文件系统会对读取同块(block)的请求进行合并 |
| wrqm/s | 每秒对该设备的写请求被合并次数 |
| r/s | 每秒完成的读次数 |
| w/s | 每秒完成的写次数 |
| rkB/s | 每秒读数据量(kB为单位) |
| wkB/s | 每秒写数据量(kB为单位) |
| avgrq-sz | 平均每次IO操作的数据量(扇区数为单位) |
| avgqu-sz | 平均等待处理的IO请求队列长度 |
| await | 平均每次IO请求等待时间(包括等待时间和处理时间，毫秒为单位) |
| svctm | 平均每次IO请求的处理时间(毫秒为单位) |
| %util | 采用周期内用于IO操作的时间比率，即IO队列非空的时间比率 |

### 4.4重点关注性能监控指标

上面说了这么多，也看了那么多的系统输出，那我们在日常运维中需要重点关注以下字段的输出内容来确定这台服务器是否存在IO性能瓶颈

**%iowait：**如果该值较高，表示磁盘存在I/O瓶颈。

**await：**一般情况下系统I/O响应时间应该低于5ms，如果大于10ms就比较大了。

**avgqu-sz：**如果I/O请求压力持续超出磁盘处理能力，该值将增加。如果单块磁盘的队列长度持续超过2，一般认为该磁盘存在I/O性能问题。需要注意的是，如果该磁盘为磁盘阵列虚拟的逻辑驱动器，需要再将该值除以组成这个逻辑驱动器的实际物理磁盘数目，以获得平均单块硬盘的I/O等待队列长度。

**%util：**一般情况下，如果该参数是100%表示设备已经接近满负荷运行了

最后，除了关注指标外，我们更需要结合部署的业务进行分析。对于磁盘随机读写频繁的业务，比如图片存取、数据库、邮件服务器等，此类业务tps才是关键点。对于顺序读写频繁的业务，需要传输大块数据的，如视频点播、文件同步，关注的是磁盘的吞吐量。

# SPECjvm2008使用说明

## SPECjvm2008简介

SPECjvm2008是一个观测JRE（java runtime enviroument）运行性能的基准测试套件。它的测试用例涵盖了大部分java基础应用场景，是架构选型和VM性能评测不可多得的利器。

JRE(java runtime environment)就是java程序的运行环境。你所写的java程序在经过javac编译后会形成字符行文件，这个java的中间文件，计算机是看不懂的，而这个文件是给jvm（java的虚拟机）用的，由jvm进行解释后形成机器语言给计算机。而这个jvm就在jre中，也就是说java程序的运行要有jre来担当.所以jdk和jre的区别就是一个用于开发，一个用于运行，他们都是java开发者所必备的（除非你就想运行java程序，那么用jre就足够了）。

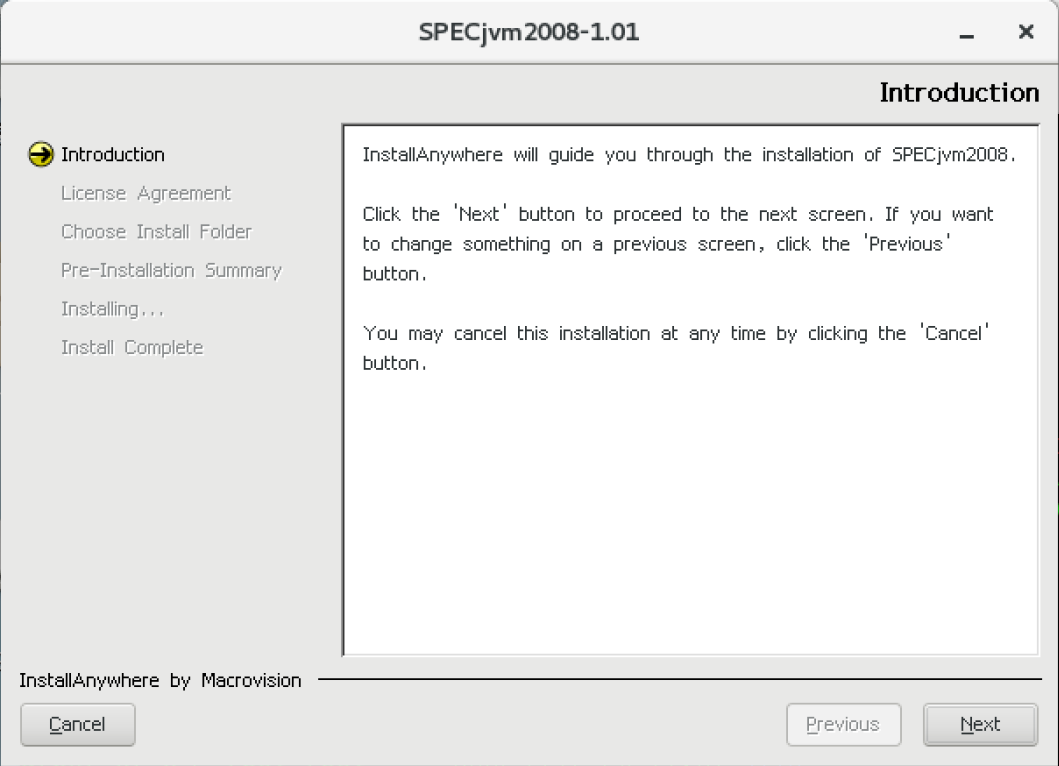
**SPECjvm2008测试用例列表：(单位 ops/m 每分钟操作数)**

|  |  |
| --- | --- |
| **测试项** | **功能说明** |
| startup.helloworld | 测试helloworld程序从运行开始到结束所需的时间 |
| startup.compiler.compiler | 普通java编译所需要的时间 |
| startup.compiler.sunflow | 编译sunflow图像渲染引擎所需要的时间 |
| startup.compress | 测试压缩程序，单次压缩所需的时间 |
| startup.crypto.aes | 测试AES/DES加密算法，单次加解密所需的时间 输入数据长度为 100 bytes , 713KB |
| startup.crypto.rsa | 测试RSA加密算法，单次加解密需要的时间 输入数据长度为 100 bytes, 16KB |
| startup.crypto.signverify | 测试单次使用MD5withRSA, SHA1withRSA, SHA1withDSA, SHA256withRSA来签名，识别所需要的时间。 输入数据长度为 1KB, 65KB, 1MB |
| startup.mpegaudio | 单次mpeg音频解码所需的时间 |
| startup.scimark.fft | 单次快速[傅立叶变换](http://zh.wikipedia.org/wiki/%E5%82%85%E9%87%8C%E5%8F%B6%E5%8F%98%E6%8D%A2)所需的时间 |
| startup.scimark.lu | 单次[LU分解](http://zh.wikipedia.org/zh-cn/LU%E5%88%86%E8%A7%A3)所需的时间 |
| startup.scimark.monte\_carlo | 单次运行[蒙特卡罗算法](http://zh.wikipedia.org/wiki/%E8%92%99%E7%89%B9%C2%B7%E5%8D%A1%E7%BD%97%E6%96%B9%E6%B3%95)所需的时间 |
| startup.scimark.sor | 单次运行jacobi[逐次超松弛迭代法](http://en.wikipedia.org/wiki/Successive_over-relaxation)所需的时间 |
| startup.scimark.sparse | 单次[稀疏矩阵](http://zh.wikipedia.org/wiki/%E7%A8%80%E7%96%8F%E7%9F%A9%E9%98%B5)乘积所需的时间 |
| startup.serial | 单次通过socket传输java序列化对象到对端反序列化完成所需的时间（基于jboss serialization benchmark） |
| startup.sunflow | 单次图片渲染处理所需的时间 |
| startup.xml.transform | 单次xml转换所需的时间，转换包括dom,sax,stream方式 |
| startup.xml.validation | 单次xml schema校验所需的时间 |
| compiler.compiler | 在规定时间内，多线程迭代测试普通java编译，得出 ops/m |
| compiler.sunflow | 在规定时间内，多线程迭代测试sunflow图像渲染，得出 ops/m |
| compress | 在规定时间内，多线程迭代测试压缩，得出 ops/m |
| crypto.aes | 在规定时间内，多线程迭代测试AES/DES加解密算法，得出 ops/m |
| crypto.rsa | 在规定时间内，多线程迭代测试RSA加解密算法，得出 ops/m |
| crypto.signverify | 在规定时间内，多线程迭代测试使用MD5withRSA, SHA1withRSA, SHA1withDSA, SHA256withRSA来签名，识别，得出 ops/m |
| derby | 在规定时间内，迭代测试数据库相关逻辑，包括数据库锁，BigDecimal计算等，最后得出 ops/m |
| mpegaudio | 在规定时间内，多线程迭代mpeg音频解码，得出 ops/m |
| scimark.fft.large | 在规定时间内，多线程迭代测试快速[傅立叶变换](http://zh.wikipedia.org/wiki/%E5%82%85%E9%87%8C%E5%8F%B6%E5%8F%98%E6%8D%A2)，使用32M大数据集，最后得出 ops/m |
| scimark.lu.large | 在规定时间内，多线程迭代测试[LU分解](http://zh.wikipedia.org/zh-cn/LU%E5%88%86%E8%A7%A3)，使用32M大数据集，最后得出 ops/m |
| scimark.sor.large | 在规定时间内，多线程迭代测试jacobi[逐次超松弛迭代法](http://en.wikipedia.org/wiki/Successive_over-relaxation)，使用32M大数据集，最后得出 ops/m |
| scimark.sparse.large | 在规定时间内，多线程迭代测试[稀疏矩阵](http://zh.wikipedia.org/wiki/%E7%A8%80%E7%96%8F%E7%9F%A9%E9%98%B5)乘积，使用32M大数据集，最后得出 ops/m |
| scimark.fft.small | 在规定时间内，多线程迭代测试快速[傅立叶变换](http://zh.wikipedia.org/wiki/%E5%82%85%E9%87%8C%E5%8F%B6%E5%8F%98%E6%8D%A2)，使用512K小数据集，最后得出 ops/m |
| scimark.lu.small | 在规定时间内，多线程迭代测试[LU分解](http://zh.wikipedia.org/zh-cn/LU%E5%88%86%E8%A7%A3)，使用512KB小数据集，最后得出 ops/m |
| scimark.sor.small | 在规定时间内，多线程迭代测试jacobi[逐次超松弛迭代法](http://en.wikipedia.org/wiki/Successive_over-relaxation)，使用512KB小数据集，最后得出 ops/m |
| scimark.sparse.small | 在规定时间内，多线程迭代测试[稀疏矩阵](http://zh.wikipedia.org/wiki/%E7%A8%80%E7%96%8F%E7%9F%A9%E9%98%B5)乘积，使用512KB小数据集，最后得出 ops/m |
| scimark.monte\_carlo | 在规定时间内，多线程迭代测试[蒙特卡罗算法](http://zh.wikipedia.org/wiki/%E8%92%99%E7%89%B9%C2%B7%E5%8D%A1%E7%BD%97%E6%96%B9%E6%B3%95)，得出 ops/m |
| serial | 在规定时间内，多线程迭代测试通过socket传输java序列化对象到对端反序列化（基于jboss serialization benchmark），得出 ops/m |
| sunflow | 在规定时间内，利用sunflow多线程迭代测试图片渲染，得出 ops/m |
| xml.transform | 在规定时间内，多线程迭代测试xml转换，得出 ops/m |
| xml.validation | 在规定时间内，多线程迭代测试xml schema验证，得出 ops/m |

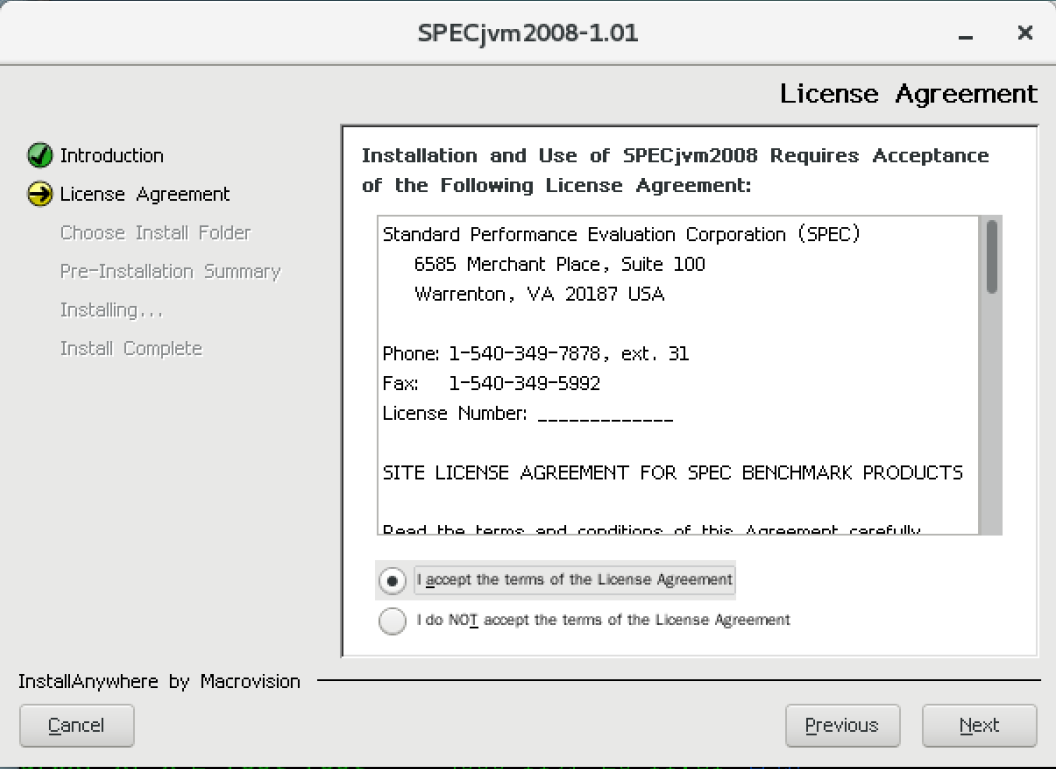
## SPECjvm2008安装

#java -jar SPECjvm2008\_1\_01\_setup.jar

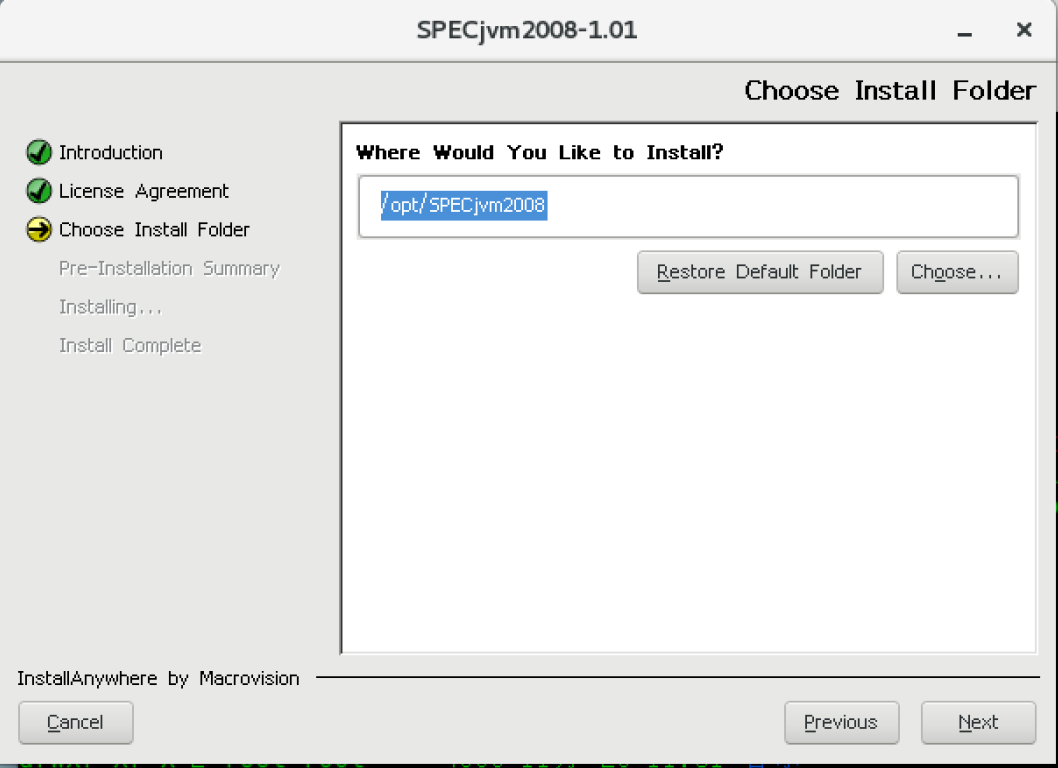
执行完以上命令后会弹出如下安装界面



点击“Next”执行下一步



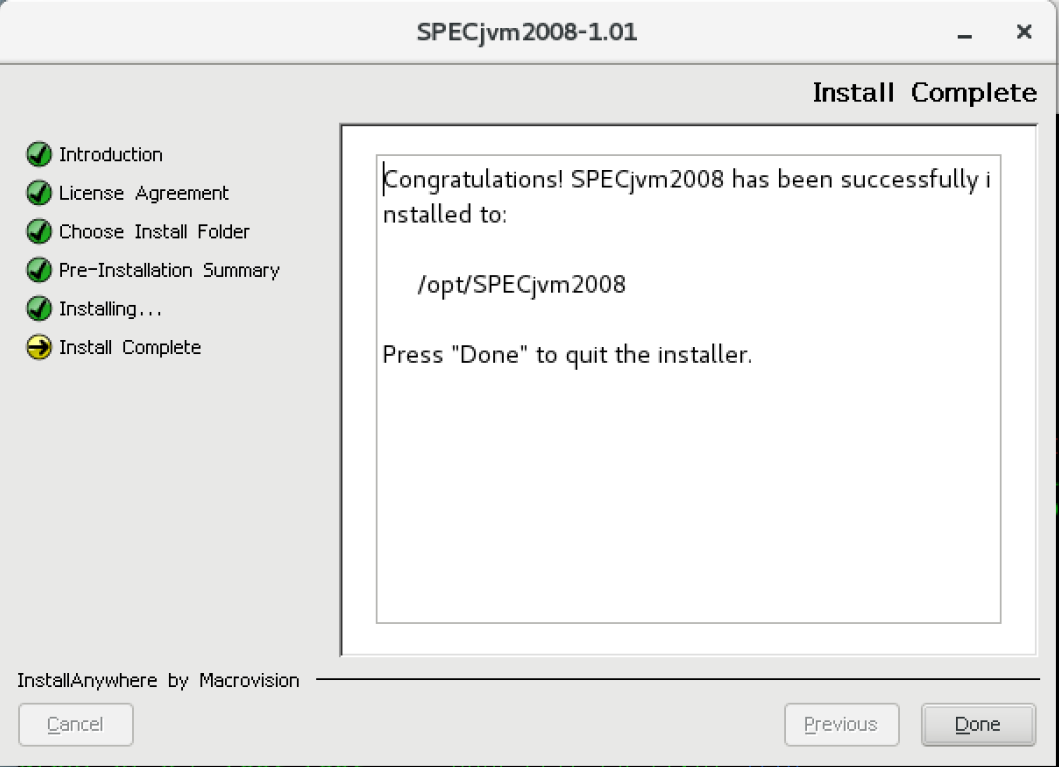
选择“I accept the terms of the License Agreement”点击“Next”执行下一步



默认路径为/SPECjvm2008，修改安装路径为/opt/SPECjvm2008，点击“”执行下一步



点击“Install”进行安装



安装完成

## 配置JAVA环境变量

#vim /etc/profile

在文件最后添加如下几行内容

|  |
| --- |
| JAVA\_HOME=/usr/lib/jvm/java-1.8.0-openjdk-arm64  CLASSPATH=.: $JAVA\_HOME/lib/tools.jar: /lib.dt.jar  PATH=$JAVA\_HOME/bin: $PATH  export JAVA\_HOME CLASSPATH PATH |

#source /etc/profile

执行source命令使/etc/profile文件中变量生效

## SPECjvm2008测试

#cd /opt/SPECjvm2008

#./run-specjvm.sh -ikv

运行以上脚本即可执行SPECjvm2008所有的测试项

如果只想单条测试，可以执行如下命令：

#./run-specjvm.sh  startup.helloworld -ikv

................

测试完成后相应的报表（jpg，txt等），报表默认的输出路径是安装目录的 results目录下。内部根据你执行测试套件的顺序，又分了不同的子文件夹。进入某个子文件夹，查看html格式的报表就行了。

# UnixBench使用说明

## UnixBench简介

UnixBench是一个类unix（Unix，BSD，Linux）系统下的性能测试工具，一个开源工具，被广泛用与测试linux系统主机的性能。Unixbench的主要测试项目有：系统调用、读写、进程、图形化测试、2D、3D、管道、运算、C库等系统基准性能提供测试数据。不单纯是CPU 内存 或者磁盘测试工具。测试结果不仅仅取决于硬件，也取决于系统、开发库、甚至是编译器。  
 下面使用了最新版UnixBench5.1.3来测试，注释了关于graphic的测试项（大多数服务器都是没有显卡或者是集显，所以图像性能无需测试），运行10-30分钟后（根据CPU内核数量，运算时间不等）得出分数，越高越好。

UnixBench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。

这样的设计是为了实现以下目标：  
 测试系统的单任务性能  
 测试系统的多任务性能  
 测试系统并行处理的能力

**测试项目说明：**

|  |  |
| --- | --- |
| **测试项** | **功能说明** |
| Dhrystone | 这个基准测试是用来计算和比较计算机性能的。由于没有浮点操作，这个测试主要集中在字符串处理上。这个测试深受以下因素的影响：硬件和软件设计、编译程序和链接程序选项、代码优化、高速缓冲存储器、等待状态和整数数据类型。 |
| Whetstone | 这个测试测量浮点运算速度和效率。这个测试包含几个模块，这些模块被认为是由在典型运用在科学应用中的操作组成的混合体。各种各样的C函数，包括sin、cos、sqrt、exp和log还有整形，浮点算术操作、数组访问、条件转移和过程调用，都使用了。整数和浮点算术，在这个测试中都会计算。 |
| Execl Throughput | 这个测试计算每秒钟出现的execl调用数。Execl是exec函数家族的一部分。它用一个新的进程映像更新当前的进程映像。它和许多其他相似的命令都是execve（）的前端程序。 |
| File Copy | 这个测试使用不同的缓冲区大小时，数据可以从一个文件被传输到另一个的速率。文件的读写复制测试可以获得在指定时间内（默认为10秒）能够读、写、复制的字符数目。 |
| Pipe Throughput | 管道是进程间通信最简单的方式之一。管道吞吐量指的是每秒中，一个进程能将512个字节数据写入管道和从管道中回读它们的次数。这个测试在真实的程序设计中并没有对应物。 |
| Pipe-based Context Switching | 这个测试是计算两个进程通过管道交换一个增长的整数的次数。基于管道的上下文切换测试更像一个真实的应用程序。这个测试产生一个可以用来继续双向管对话的子进程。 |
| Process Creation | 这个测试计算一个进程派生和收获一个马上退出的子进程的次数。由于进程创建涉及创建进程控制块和为新的进程分配内存，因此这项测试直接取决于内存带宽。这个测试一般地会用于比较各种各样的操作系统进程创建调用的执行。 |
| Shell Scripts | 这个shells scripts测试测量每秒进程能够启动和收获一组1，2，4，8个shell脚本程序的并行的拷贝的次数。 |
| System Call Overhead | 这个测试是估计进入和离开操作系统内核的开销。例如执行一个系统调用的开销。它由一个简单的系统组成，这个系统重复的调用getid系统调用，getid返回调用进程的进程IS号。这个测试是用执行这些调用的时间来估算进入和离开内核的开销。 |
| Graphical Tests | 这个测试提供了2D，3D的图形化测试。报告性能不仅依赖于硬件，还与你的系统是否对硬件有合适的驱动程序有关。 |

**注意：**

在测试之前首先确保被测试系统是干净的，没有大量后台占用资源的程序在运行。比较简单的方法是重新部署一个操作系统，做简单必要配置后，重启，就可以了。

如果是对比测试，把能控制的因素（测试环境）都控制成相同。比如测试Raspberry 2和3的时候，就可以使用同一张存储卡，系统版本也要保持一致。

## UnixBench安装

下载地址：[https://github.com/kdlucas/byte-unixbench](https://github.com/kdlucas/byte-unixbench/tree/master)

解压并进入软件目录

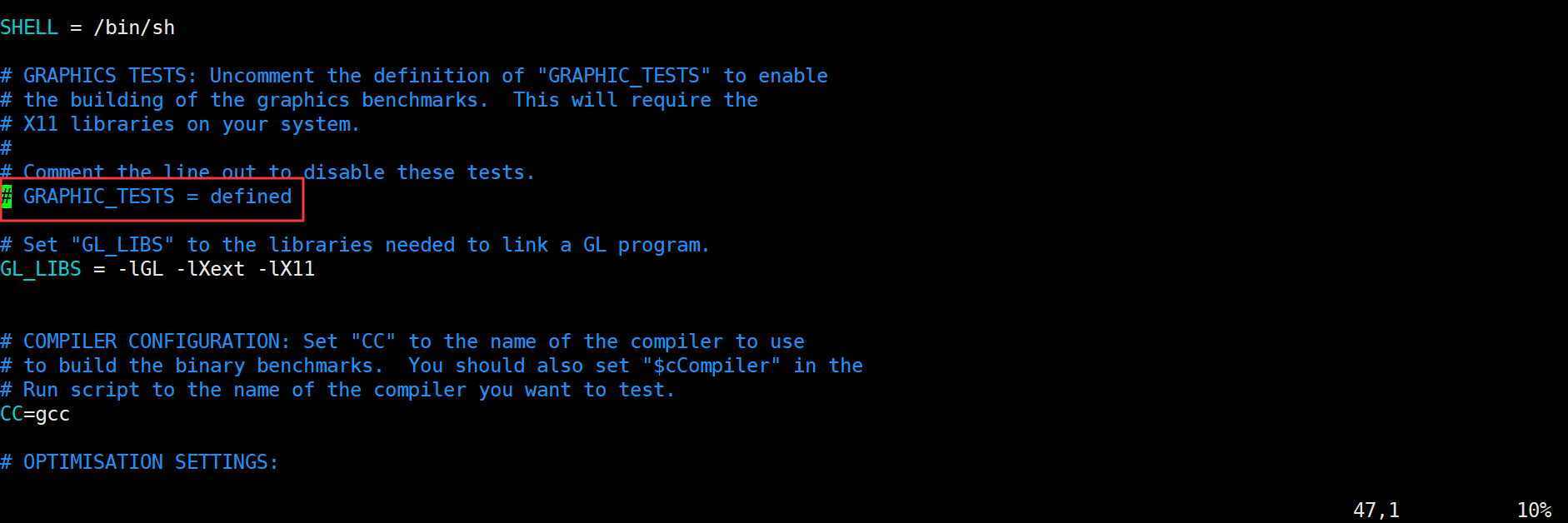
# tar -xvf UnixBench5.1.3.tgz

# cd UnixBench

最新版本UnixBench5.1.3，包含system和graphic测试，如果你需要测试graphic（图形化），则需要修改Makefile，不要注释掉”GRAPHIC\_TESTS = defined”，同时需要系统提供x11perf命令gl\_glibs库，如果是针对服务器等非图形环境的测试，需要关掉图形测试。

编辑Makefile文件

# vim Makefile



注释掉GRAPHIC\_TESTS = defined这一行，也就是在这一行开头加个“#”号。我这个版本是在第47行，另外我这个版本的这行默认是注释了的，所以不需要进行修改。

# make all

**注意：默认unixbench只支持到16cores，修改参数，取消此限制**

sed -i "s/\"System Benchmarks\", 'maxCopies' => 16/\"System Benchmarks\", 'maxCopies' => 0/" Run

sed -i 's/$copies > $maxCopies/$maxCopies > 0 \&\& $copies > $maxCopies/' Run

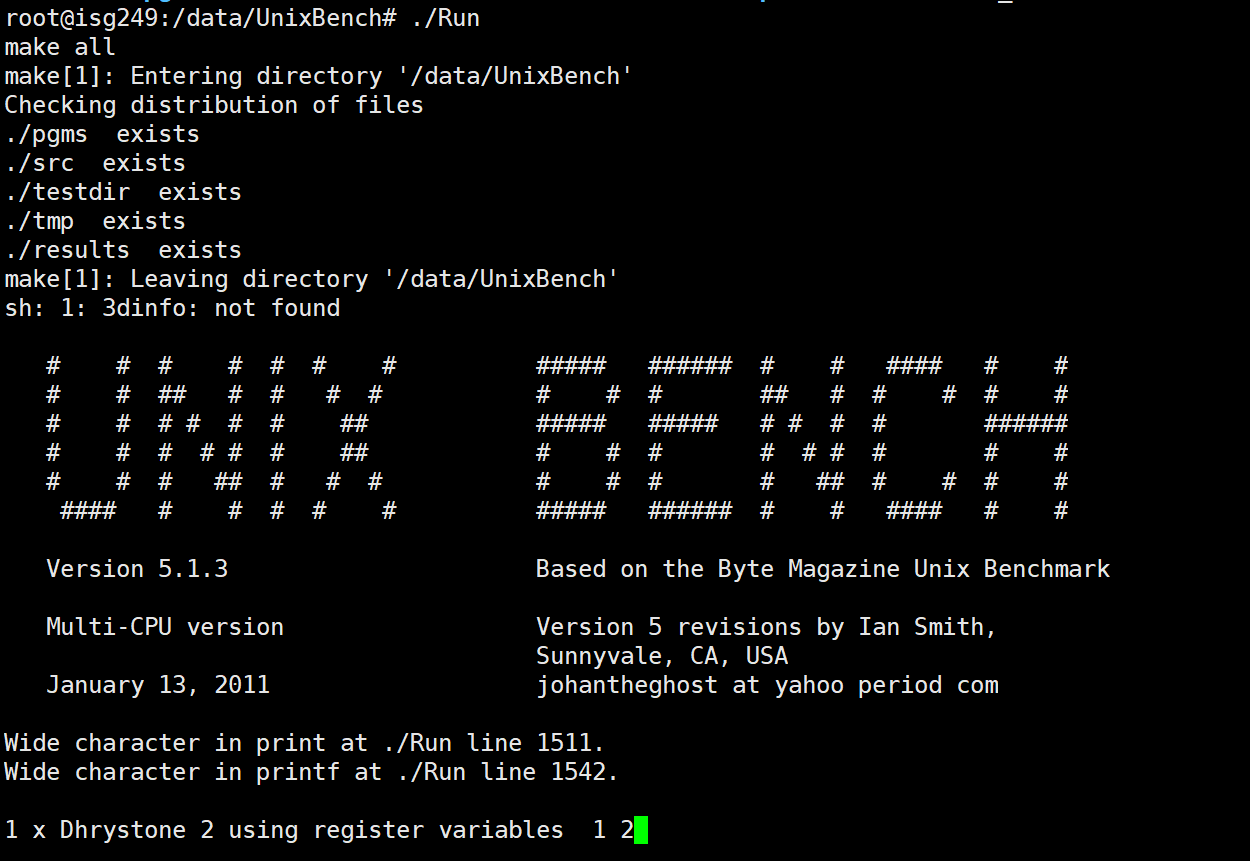
如果没有错误，就可以运行测试了

## UnixBench测试

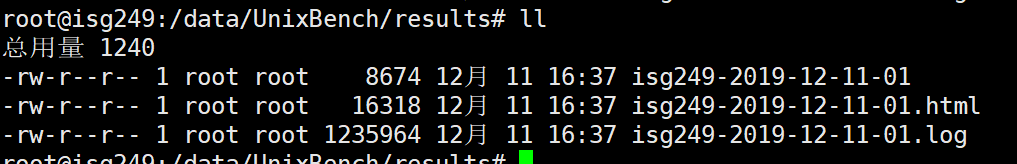
如果执行Run就是做一个系统测试，如果执行Run graphics就是执行图形测试，执行Run gindex就是两个测试都执行，测试时间是相当漫长的。

正常系统测试时间大概运行1个小时左右。

# ./Run

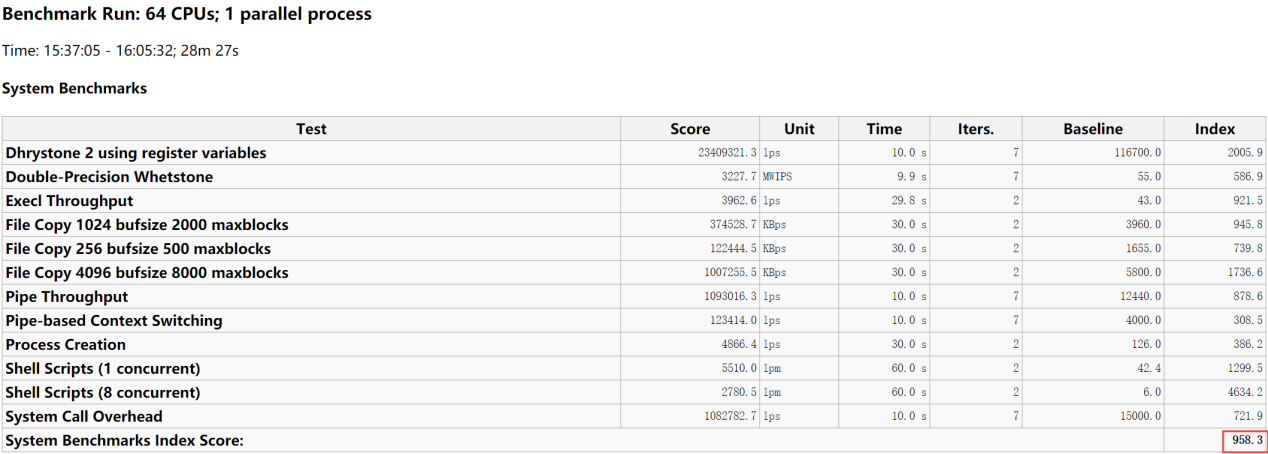


默认测试完成后测试结果会存放在results目录，如下：

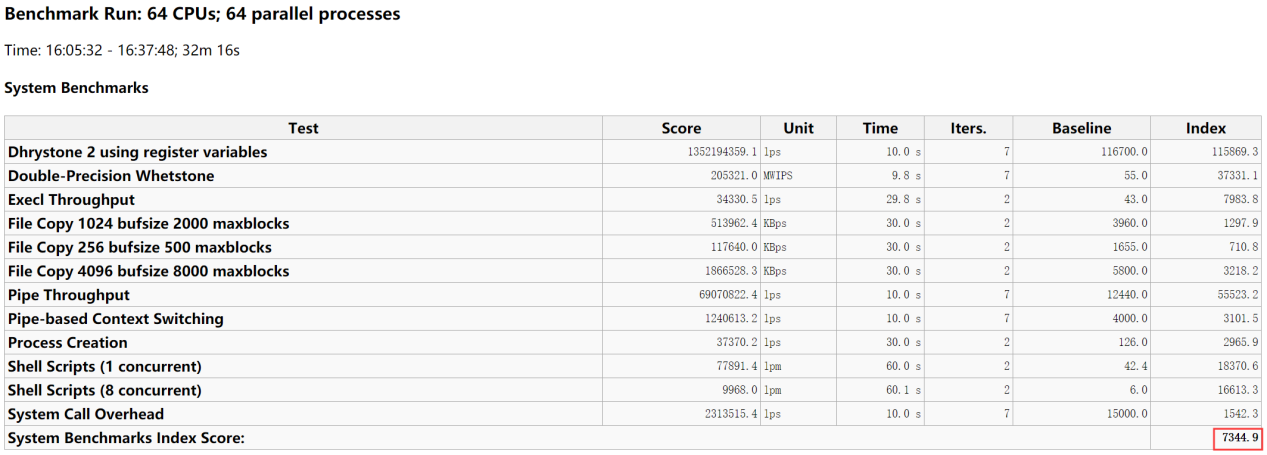


程序运行完会有两个跑分结果，一个是 1 parallel process 的结果，另一个是64 parallel process 的结果 （具体可以看html里的输出）。两者的区别即一个是单进程跑，一个是多进程跑 。

**单线程测试结果**



**64线程结果**



**测试项目分析**

测试过程中每个项目后面会有1 2 3 4 5 6 7 8 9 10 数字，意思是进行了10组测试，同理有的项目是1 2 3数字，意思是进行了3组测试，测试过程中部分内容及解释如下：

* **Dhrystone 2 using register variables**

此项用于测试 string handling，因为没有浮点操作，所以深受软件和硬件设计（hardware and software design）、编译和链接（compiler and linker options）、代码优化（code optimazaton）、对内存的cache（cache memory）、等待状态（？wait states）整数数据类型（integer data types）的影响

* **Double-Precision Whetstone**

这一项测试浮点数操作的速度和效率。这项测试包括几个模块，每个模块都包括一组用于科学计算的操作。覆盖面很广的一系列c函数：sin，cos，sqrt，exp，log 被用于整数和浮点数的数学运算、数组访问、条件分支（conditional branch）和程序调用。此测试同时测试了整数和浮点数算术运算。

* **Execl Throughput**

此测试考察每秒钟可以执行的 execl 系统调用的次数。 execl 系统调用是 exec 函数族的一员。它和其他一些与之相似的命令一样是 execve（） 函数的前端。

* **File copy**

测试从一个文件向另外一个文件传输数据的速率。每次测试使用不同大小的缓冲区。这一针对文件 read、write、copy 操作的测试统计规定时间（默认是 10s）内的文件 read、write、copy 操作次数。

|  |
| --- |
| Filesystem Throughput 1024 bufsize 2000 maxblocks  Filesystem Throughput 256 bufsize 500 maxblocks  Filesystem Throughput 4096 bufsize 8000 maxblocks |

* **Pipe Throughput**

管道（pipe）是进程间交流的最简单方式，这里的Pipe throughtput指的是一秒钟内一个进程可以向一个管道写512字节数据然后再读回的次数。需要注意的是，pipe throughtput在实际编程中没有对应的真实存在。

* **Pipe-based Context Switching**

这个测试两个进程（每秒钟）通过一个管道交换一个不断增长的整数的次数。这一点很像现实编程中的一些应用，这个测试程序首先创建一个子进程，再和这个子进程进行双向的管道传输。

* **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）。process creation的关注点是新进程进程控制块（process control block）的创建和内存分配，即一针见血地关注内存带宽。一般说来，这个测试被用于对操作系统进程创建这一系统调用的不同实现的比较。

* **Shell Scripts**

测试一秒钟内一个进程可以并发地开始一个shell脚本的n个拷贝的次数，n 一般取值 1，2，4，8.（我的系统上取1，8）。这个脚本对一个数据文件进行一系列的变形操作（transformation）。

|  |
| --- |
| Shell Scripts (1 concurrent) 1 2 3  Shell Scripts (8 concurrent) 1 2 3 |

* **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。它利用一个反复地调用 getpid 函数的小程序达到此目的。

**总结**

很多用户和工程师装好软件后，直接运行./Run就把结果跑出来了，然后还只取最后一个分值，比谁高谁低。

很多人就拿了7344.9那个分数去比了，但是上面那个单进程结果958.3也有意义，但凡接触了这么多案例没人关心过。多进程结果就是根据系统CPU核数，同时起多个进程的结果；单进程就是只起一个进程的结果，对于系统性能来说，两个结果都是有意义的。

如果大家对测试的分数算法感兴趣的话，可以参考以下网址：

<https://yq.aliyun.com/articles/674729>

# Netperf使用说明

## Netperf简介

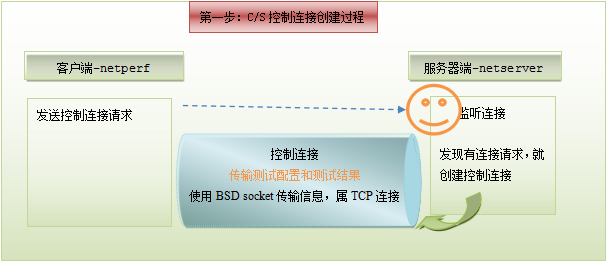
Netperf是由惠普公司开发的，测试网络栈。即测试不同类型的网络性能的benchmark工具，大多数网络类型TCP/UPD端对端的性能，得到网络上不同类型流量的性能参数。Netperf根据应用的不同可以进行不同模式的网络性能测试，即：批量数据传输模式和请求/应答模式。

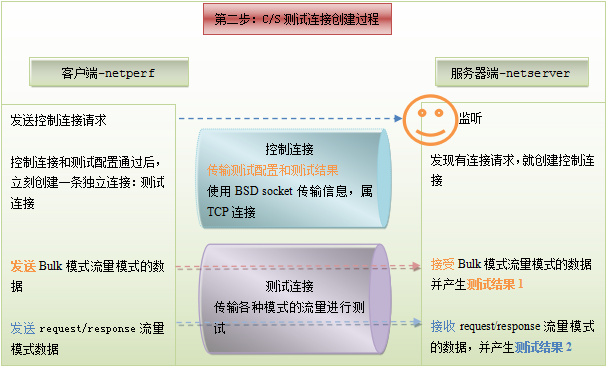
Netperf测试结果所反映的是一个系统能够以多快的速度向另外一个系统发送数据，以及另外一个系统能够以多快的速度接收数据。

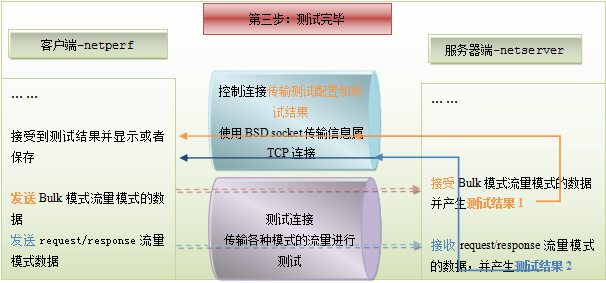
**工作原理：**

Netperf工具以client/server方式工作。server端是netserver，用来侦听来自client端的连接，client端是netperf，用来向server发起网络测试。在client与server之间，首先建立一个控制连接，传递有关测试配置的信息，以及测试的结果；在控制连接建立并传递了测试配置信息以后，client与server之间会再建立一个测试连接，进行来回传递特殊的流量模式，以测试网络的性能。

具体过程如下图所示：







**网络性能测量的五项指标**

* **可用性（availability):** 测试网络性能的第一步是确定网络是否正常工作，最简单的方法是使用ping命令。通过向远端的机器发送 icmp echo request，并等待接收 icmp echo reply 来判断远端的机器是否连通，网络是否正常工作。  
  Ping 命令有非常丰富的命令选项，比如 -c 可以指定发送 echo request 的个数，-s 可以指定每次发送的ping包大小 。
* **响应时间（response time)：p**ing 命令的 echo request/reply 一次往返所花费时间就是响应时间。有很多因素会影响到响应时间，如网段的负荷、网络主机的负荷、广播风暴、工作不正常的网络设备等等。
* **网络利用率（network utilization):** 是指网络被使用的时间占总时间（即被使用的时间+空闲的时间）的比例。
* **网络吞吐量（network throughput):** 是指在某个时刻，在网络中的两个节点之间，提供给网络应用的剩余带宽。网络吞吐量非常依赖于当前的网络负载情况。因此为了得到正确的网络吞吐量，最好在不同时间（一天中的不同时刻，或者一周中不同的天）分别进行测试，只有这样才能得到对网络吞吐量的全面认识。
* **网络带宽容量（network bandwidth capacity):** 与网络吞吐量不同，网络带宽容量指的是在网络的两个节点之间的最大可用带宽。这是由组成网络的设备的能力所决定的。

## Netperf安装

下载地址：<https://github.com/HewlettPackard/netperf>

解压并进入软件目录安装软件

# tar -xvf netperf-2.7.0.tar.gz

# cd netperf-2.7.0/

# ./configure --build=arm #这里需要指定ARM架构

# make && make install

默认安装路径是/usr/local下的各目录。

安装完以后，会生成两个工具：netserver和netperf

**注意：测试过程中服务端和客户端都需要安装netperf，服务端使用netserver启动后，客户端就可以使用netperf来测试网络的性能。**

**服务器端：**

服务器端启动服务，执行netserver命令

# netserver

|  |
| --- |
| Starting netserver with host 'IN(6)ADDR\_ANY' port '12865' and family AF\_UNSPEC |

默认情况下netserver开启端口号为12865，可以通过以下命令指定服务端口（如果服务端指定端口，则客户端也需要指定服务器端口）：

netserver -D -p 8888

**客户端：**

客户端执行netperf相关命令即可

**Netperf使用**

netperf语法格式为:

Usage: netperf [global options] -- [test specific options]

**全局命令行参数包括如下选项：**

|  |  |
| --- | --- |
| **参数** | **说明** |
| -H host | 指定远端运行netserver的server IP地址 |
| -l testlen | 指定测试的时间长度(秒) |
| -t testname | 指定进行的测试类型(TCP\_STREAM，UDP\_STREAM，TCP\_RR，TCP\_CRR，UDP\_RR) |

**测试相关的局部参数包括如下选项：**

|  |  |
| --- | --- |
| **参数** | **说明** |
| -s size | 设置本地系统的socket发送与接收缓冲大小 |
| -S size | 设置远端系统的socket发送与接收缓冲大小 |
| -m size | 设置本地系统发送测试分组的大小 |
| -M size | 设置远端系统接收测试分组的大小 |
| -D | 对本地与远端系统的socket设置TCP\_NODELAY选项 |
| -r req,resp | 设置request和reponse分组的大小 |

## Netperf测试

### 3.1批量网络流量的性能测试

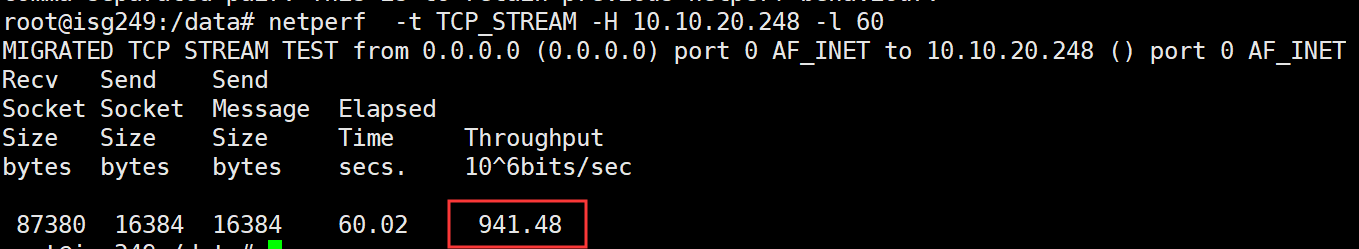
批量数据传输典型的例子有ftp和其它类似的网络应用（即一次传输整个文件）。根据使用传输协议的不同，批量数据传输又分为TCP批量传输和UDP批量传输。

#### 3.1.1 TCP\_STREAM测试

Netperf缺省情况下进行TCP批量传输，即-t TCP\_STREAM。测试过程中netperf向netserver发送批量的TCP数据分组，以确定数据传输过程中的吞吐量：

**用例1 测试60秒内TCP批量数据传输表现**

# netperf -t TCP\_STREAM -H 10.10.20.248 -l 60



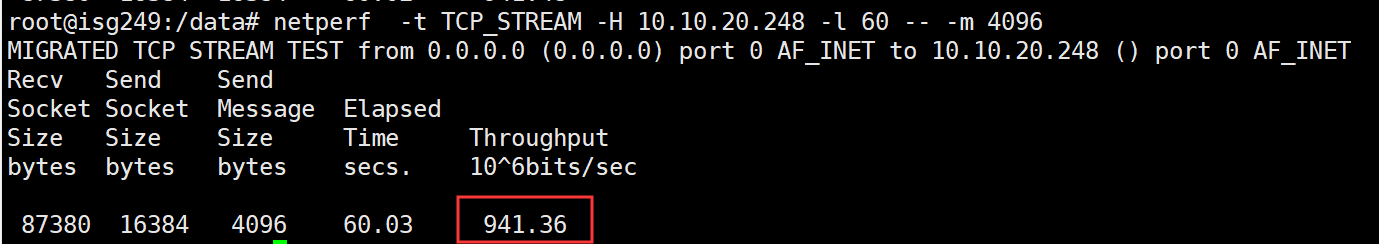
从netperf的结果输出中，我们可以知道以下的一些信息：

|  |
| --- |
| 远端系统（即server）使用大小为87380字节的socket接收缓冲；  本地系统（即client）使用大小为16384字节的socket发送缓冲；  向远端系统发送的测试分组大小为16384字节；  测试的时间为60秒。  吞吐量的测试结果为941.48 Mbits/秒。 |

在默认情况下，netperf向发送的测试分组大小设置为本地系统所使用的socket发送缓冲大小，测试时间为10秒。

**用例2 测试分组为4096，观察吞吐量的变化**

# netperf -t TCP\_STREAM -H 10.10.20.248 -l 60 -- -m 4096



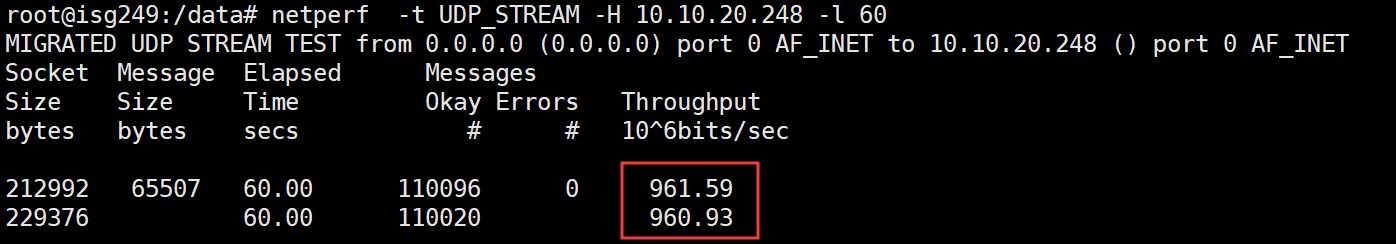
|  |
| --- |
| 从上面两个示例中可以看出，当减小测试分组的大小为4096后，吞吐量几乎没有变化。 由此可以说明网络中间的路由器不存在缓冲区的问题。 |

#### 3.1.2 UDP\_STREAM测试

用来测试进行UDP批量传输时的网络性能。需要特别注意的是，此时测试分组的大小不得大于socket的发送与接收缓冲大小，否则netperf会报错：

**用例1 测试60秒内UDP批量数据传输表现**

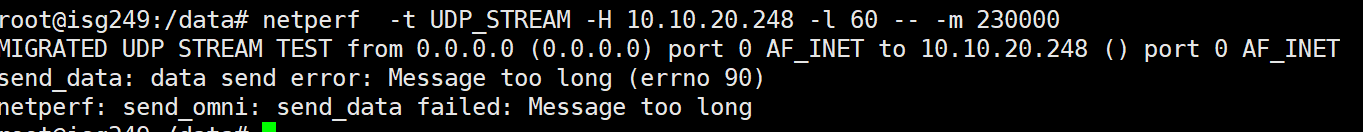
# netperf -t UDP\_STREAM -H 10.10.20.248 -l 60



|  |
| --- |
| 测试结果说明：接收的110020少于发送的110096，吞吐量也有差别  UDP\_STREAM方式的结果中有两行测试数据：  第一行显示的是本地系统的发送统计，这里的吞吐量表示netperf向本地socket发送分组的能力，但是我们知道UDP是不可靠的传输协议，发送出去的分组数量不一定等于接收到的分组数量。  第二行显示的就是远端系统接收的情况，由于client与server直接连接在一起，而且网络中没有其它的流量，所以本地系统发送过去的分组几乎都被远端系统正确的接收了，远端系统的吞吐量也几乎等于本地系统的发送吞吐量。但是在实际环境中，一般远端系统的socket缓冲大小不同于本地系统的socket缓冲区大小，而且由于UDP协议的不可靠性，远端系统的接收吞吐量要远远小于发送出去的吞吐量。 |

**用例2 缓存大于接收**

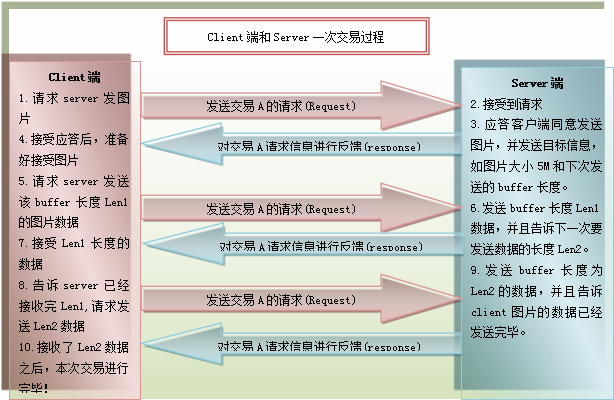
# netperf -t UDP\_STREAM -H 10.10.20.248 -l 60 -- -m 230000



|  |
| --- |
| 从例1中可以看出socket的发送与接收缓冲大小分别为212992和229376，而分组的大小230000大于212992，因此报错。  为了避免这样的情况，可以通过命令行参数限定测试分组的大小，或者增加socket的发送/接收缓冲大小。UDP\_STREAM方式使用与TCP\_STREAM方式相同的局部命令行参数，因此可以使用-m来修改测试中使用分组的大小： |

### 3.2测试请求/应答网络流量的性能

另一类常见的网络流量类型是应用在client/server结构中的request/response模式。在每次交易（transaction）中，client向server发出小的查询分组，server接收到请求，经处理后返回大的结果数据。如下图所示：

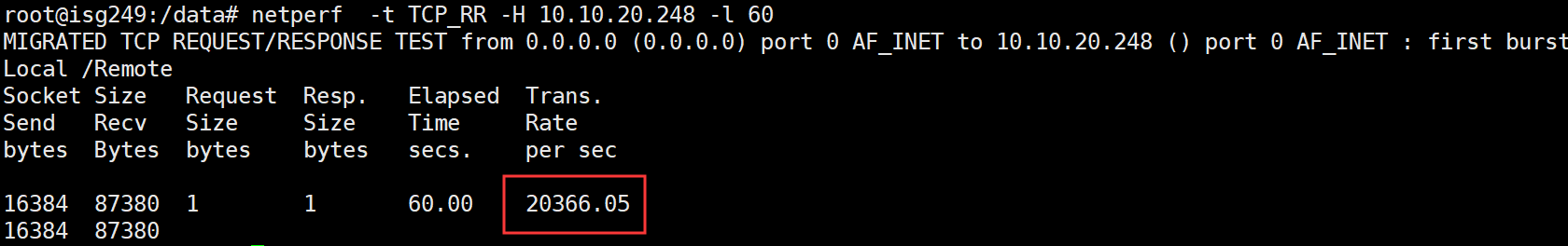


#### 3.2.1 TCP\_RR 测试

TCP\_RR方式的测试对象是多次TCP request和response的交易过程，但是它们发生在同一个TCP连接中，这种模式常常出现在数据库应用中。数据库的client程序与server程序建立一个TCP连接以后，就在这个连接中传送数据库的多次交易过程。

**用例1 测试60秒内TCP\_RR数据传输表现**

# netperf -t TCP\_RR -H 10.10.20.248 -l 60



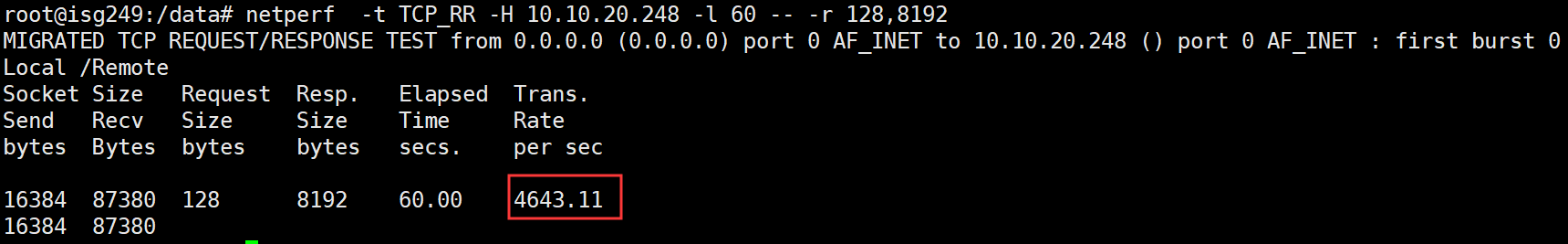
|  |
| --- |
| 从netperf的结果输出中可以看出：第一行显示本地系统的情况，第二行显示的是远端系统的信息，平均的交易率（transaction rate）为20366.05 次/秒。注意默认情况下每次交易中的request和response分组的大小都为1个字节，不具有实际意义。 |

用户可以通过测试相关的参数来改变request和response分组的大小，TCP\_RR方式下的参数如下表所示：

|  |  |
| --- | --- |
| **参数** | **说明** |
| -s size | 设置本地系统的socket发送与接收缓冲大小 |
| -S size | 设置远端系统的socket发送与接收缓冲大小 |
| -r req,resp | 设置request和reponse分组的大小 |
| -D | 对本地与远端系统的socket设置TCP\_NODELAY选项 |

**用例2 通过使用-r参数测试**

# netperf -t TCP\_RR -H 10.10.20.248 -l 60 -- -r 128,8192



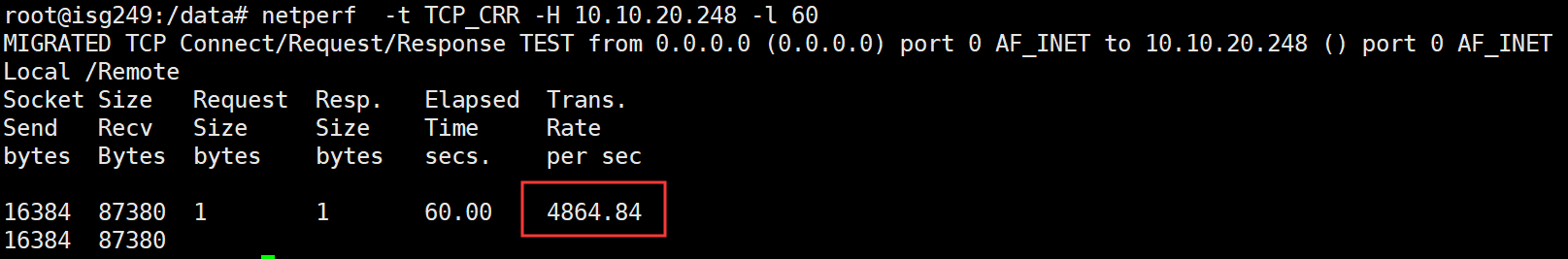
|  |
| --- |
| 从结果中可以看出，由于request/reponse分组的大小增加了，导致了交易率明显的下降。注：相对于实际的系统，这里交易率的计算没有充分考虑到交易过程中的应用程序处理时延，因此结果往往会高于实际情况。 |

#### 3.2.2 TCP\_CRR 测试

与TCP\_RR不同，TCP\_CRR为每次交易建立一个新的TCP连接。最典型的应用就是HTTP，每次HTTP交易是在一条单独的TCP连接中进行的。因此需要不停地建立新的TCP连接，并且在交易结束后断开TCP连接，交易率一定会受到很大的影响。

**用例1 测试60秒内TCP\_CRR数据传输表现**

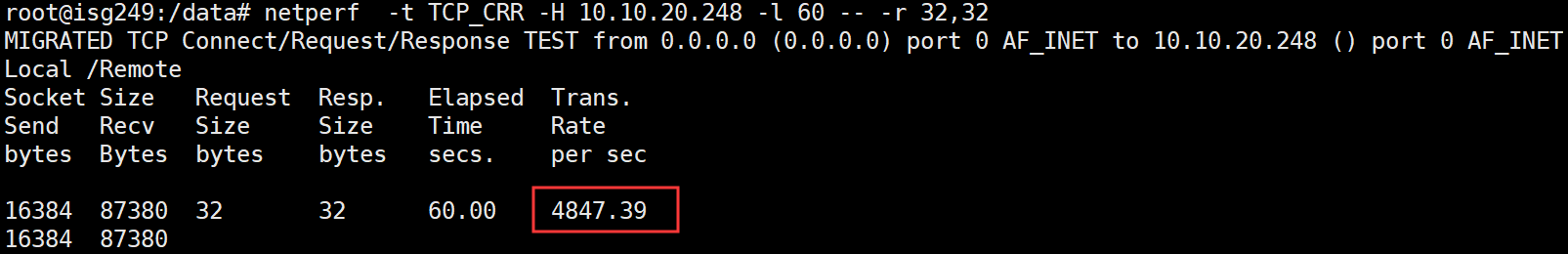
# netperf -t TCP\_CRR -H 10.10.20.248 -l 60



|  |
| --- |
| 从结果中可以看出，从每次重新建立TCP连接，效率明显下降。  即使是使用1个字节的request/response分组，交易率也明显的降低了，只有4864.84次/秒。 |

**用例2 通过使用-r参数测试**

# netperf -t TCP\_CRR -H 10.10.20.248 -l 60 -- -r 32,32

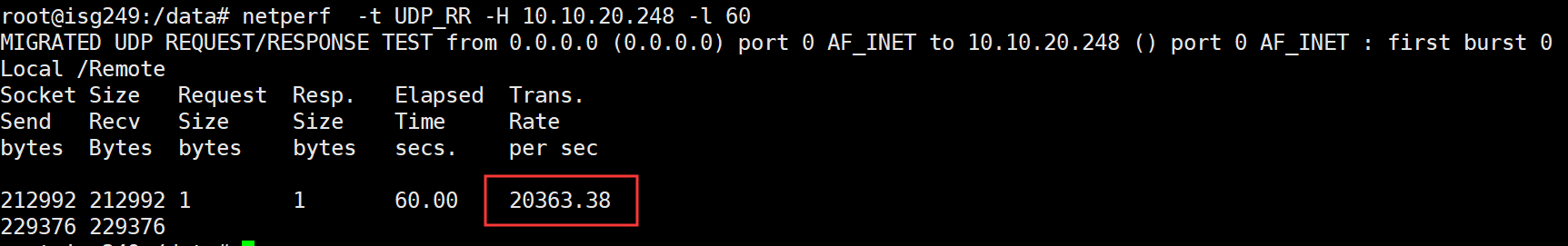


#### 3.2.3 UDP\_RR 测试

UDP\_RR方式使用UDP分组进行request/response的交易过程。由于没有TCP连接所带来的负担，所以我们推测交易率会有一定的提升。

**用例1 测试60秒内UDP\_RR数据传输表现**

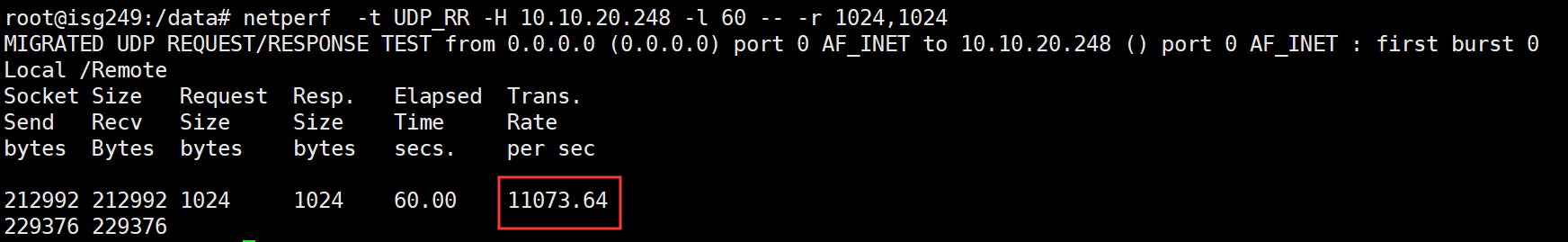
# netperf -t UDP\_RR -H 10.10.20.248 -l 60



|  |
| --- |
| 结果证实了我们的推测，交易率为20363.38次/秒，高过TCP\_RR的数值。如果出现了相反的结果，即交易率反而降低了，也不需要担心，因为这说明了在网络中，路由器或其它的网络设备对UDP采用了与TCP不同的缓冲区空间和处理技术。 |

**用例2 通过使用-r参数测试**

# netperf -t UDP\_RR -H 10.10.20.248 -l 60 -- -r 1024,1024



|  |
| --- |
| 从结果中可以看出，如果发送和接收使用1024 bytes（大包）的话会对性能产生一定影响。 |

# iPerf使用说明

## iPerf简介

网络带宽是衡量两个网络节点之间通信性能的重要的指标之一。在评估云服务的时候，要充分考虑网络带宽对所承载的网路服务的影响。若已经部署的网络服务出现了故障，有时候可以考虑从网络实时带宽的角度来进行故障排查。

网络带宽测试的工具有很多，测试方法也多种多样。针对虚拟机和云服务，我们推荐使用 iPerf 来进行网络带宽测试。iPerf 是专业的网络测试工具，它基于 TCP/IP 和 UDP/IP 协议，用以测量两个网络节点之间 TCP 和 UDP 端口的网络带宽，还能提供网络延迟、丢包率等统计信息。

iPerf常用的版本有iPerf2和 iPerf3。iPerf3在iPerf2的基础上新增了一些功能，例如发送方/接收方角色互换，以 JSON 格式输出结果，零拷贝方式传输数据等等。也去掉了iPerf2中所支持少许功能，例如双向测试，以逗号为分隔符输出结果等。iPerf3和iPerf2所执行的命令名字也不一样，iPerf3为iperf3，iPerf2 为 iperf。大家可以根据实际需要来选择安装的版本。下文中的测试都以iPerf3为例。

iPerf在工作时，测试的两端一方作为Server，另一方为Client。程序启动的命令相同，通过不同的参数来区别以哪种工作方式运行。通常情况下先启动 Server端，使iPerf监听在某个固定端口。然后在Client端执行相应的命令开始测试。

iperf 是一个[TCP/IP](http://www.ha97.com/tag/tcpip)和UDP/IP网络性能测试工具，能够提供网络吞吐率信息，以及震动、丢包率、最大段和最大传输单元大小等统计信息；从而能够帮助我们测试网络性能，定位网络瓶颈。因为最近有用到这个工具，并且这个工具做的非常不错，这里记录一下工具的使用方法。

iperf 是个开源并且跨平台的软件，代码托管在 [GitHub](https://github.com/esnet/iperf) 上，可以从 [Releases](https://github.com/esnet/iperf/releases) 找到各个发行版本，也可以去[官网](https://iperf.fr/iperf-download.php)下载各个平台的版本。使用 iperf 时，需要分别运行服务端和客户端，在测试是最好保证两个端的软件版本一致，这样会免去一些没必要的麻烦。

想了解更多信息请看：<https://software.es.net/iperf>

源代码下载和问题跟踪：<https://github.com/esnet/iperf>

## iPerf安装

iPerf支持大多数主流的操作系统，例如Windows、Ubuntu、Fedora、openSUSE 和Mac OS等等。甚至在Android和iPhone/iPad的应用商店中，也有一款名为 HE.NET Network Tools的App集成了iPerf工具。官网上列出了所有支持的操作系统的iPerf下载链接。最新的版本为 iPerf 3.1.7（不同操作系统所支持的最高版本可能不同）。如果要使用 iPerf2，可以下载2.0.X的版本。

Windows版的iPerf下载、解压后可以直接在命令提示符下运行。Windows 版的iPerf还有UWP版本，可以从Microsoft Store的Windows App中下载。针对不同发行版本的Linux，iPerf 提供了RPM 、DEB等格式的安装包，直接用rpm、dpkg等命令安装即可。

当然在没有找到合适的安装包时，大家也可以选择源码包编译安装。以下是在Kylin中使用源码包安装iPerf3的安装步骤，仅供参考。

下载iPerf3的网址是:

<https://downloads.es.net/pub/iperf/>

要查看最新的代码，请在以下位置克隆git库:

<https://github.com/esnet/iperf.git>

# wget https://downloads.es.net/pub/iperf/iperf-3.7.tar.gz

解压并进入软件目录安装软件

# tar -xvf iperf-3.7.tar.gz

# cd iperf-3.7

# ./configure

# make

# make install

安装完成后，iperf3被安装至/usr/local/bin/下，在系统的任意路径都可以执行。

若iperf3运行报以下错误：

|  |
| --- |
| iperf3: error while loading shared libraries: libiperf.so.0: cannot open shared object file: No such file or directory |

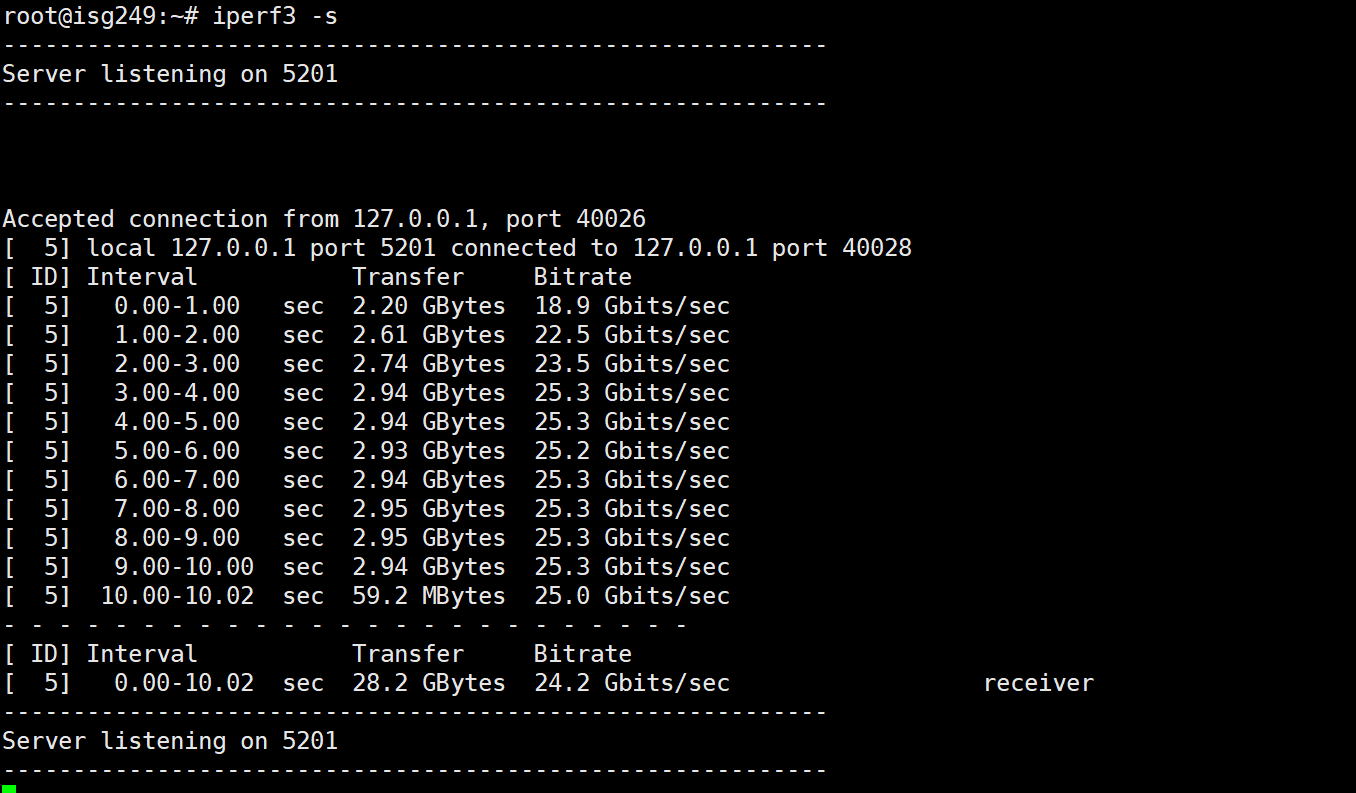
可以通过运行ldconfig命令来解决此问题。

## iPerf测试

安装好后，可以先在本机做一个简单的回环测试，结果如下：

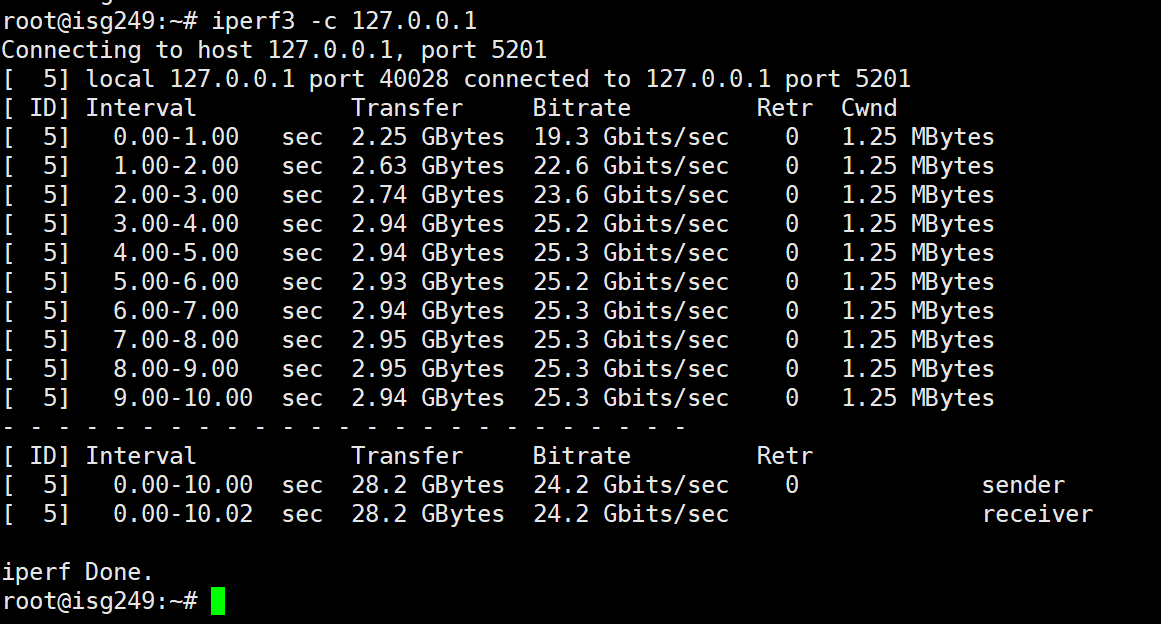
**服务端运行：**

# iperf3 -s



**客户端运行：**

# iperf3 -c 127.0.0.1



默认情况下会使用TCP连接，绑定在 5201 端口上，可以从上述结果看到，当前本机的带宽为 24.2 Gbits/sec 。

iPerf中的可选参数比较多，具体可以参见其用户手册。

<https://iperf.fr/iperf-doc.php>

### 3.1 iPerf工作模式和参数

iPerf 的工作时需要指定Server模式或Client模式，通过不同的参数来区别（-s 和-c）。iPerf3 所支持的功能也都通过设置不同的参数来实现。iPerf 的参数分为 3 类：通用参数、Server端参数、Client端参数。通用参数为Server端和Client端都可以使用的参数。以下是 iPerf 3.7 支持的所有参数：

**通用参数：**

|  |  |  |
| --- | --- | --- |
| |  |  | | --- | --- | |  | -p, --port #，Server 端监听、Client 端连接的端口号；  -f, --format [kmgKMG]，报告中所用的数据单位，Kbits, Mbits, KBytes, Mbytes；  -i, --interval #，每次报告的间隔，单位为秒；  -F, --file name，测试所用文件的文件名。如果使用在 Client 端，发送该文件用作测试；如果使用在 Server 端，则是将数据写入该文件，而不是丢弃；  -A, --affinity n/n,m，设置 CPU 亲和力；  -B, --bind ，绑定指定的网卡接口；  -V, --verbose，运行时输出更多细节；  -J, --json，运行时以 JSON 格式输出结果；  --logfile f，输出到文件；  -d, --debug，以 debug 模式输出结果；  -v, --version，显示版本信息并退出；  -h, --help，显示帮助信息并退出。 | |

**Server端参数：**

|  |
| --- |
| -s, --server，以 Server 模式运行；  -D, --daemon，在后台以守护进程运行；  -I, --pidfile file，指定 pid 文件；  -1, --one-off，只接受 1 次来自 Client 端的测试，然后退出。 |

**Client端参数：**

|  |
| --- |
| -c, --client ，以 Client 模式运行，并指定 Server 端的地址；  -u, --udp，以 UDP 协议进行测试；  -b, --bandwidth #[KMG][/#]，限制测试带宽。UDP 默认为 1Mbit/秒，TCP 默认无限制；  -t, --time #，以时间为测试结束条件进行测试，默认为 10 秒；  -n, --bytes #[KMG]，以数据传输大小为测试结束条件进行测试；  -k, --blockcount #[KMG]，以传输数据包数量为测试结束条件进行测试；  -l, --len #[KMG]，读写缓冲区的长度，TCP 默认为 128K，UDP 默认为 8K；  --cport ，指定 Client 端运行所使用的 TCP 或 UDP 端口，默认为临时端口；  -P, --parallel #，测试数据流并发数量；  -R, --reverse，反向模式运行（Server 端发送，Client 端接收）；  -w, --window #[KMG]，设置套接字缓冲区大小，TCP 模式下为窗口大小；  -C, --congestion ，设置 TCP 拥塞控制算法（仅支持 Linux 和 FreeBSD ）；  -M, --set-mss #，设置 TCP/SCTP 最大分段长度（MSS，MTU 减 40 字节）；  -N, --no-delay，设置 TCP/SCTP no delay，屏蔽 Nagle 算法；  -4, --version4，仅使用 IPv4；  -6, --version6，仅使用 IPv6；  -S, --tos N，设置 IP 服务类型（TOS，Type Of Service）；  -L, --flowlabel N，设置 IPv6 流标签（仅支持 Linux）；  -Z, --zerocopy，使用 “zero copy”（零拷贝）方法发送数据；  -O, --omit N，忽略前 n 秒的测试；  -T, --title str，设置每行测试结果的前缀；  --get-server-output，从 Server 端获取测试结果；  --udp-counters-64bit，在 UDP 测试包中使用 64 位计数器（防止计数器溢出）。 |

iPerf功能十分强大，支持的参数特别多。但是在实际使用中，并不需要同时使用这么多参数。使用时，根据实际需求来设置关键参数就可以了。

### 3.2 iPerf案例介绍

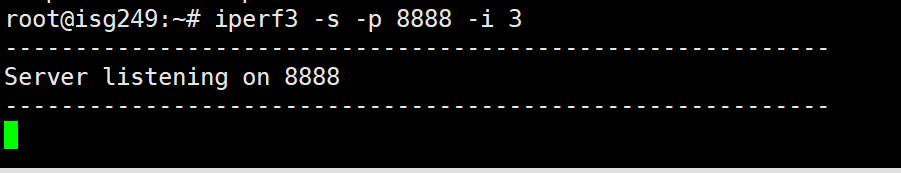
开始测试之前，需要先弄清测试对象所处网络位置。如果测试对象之间有防火墙阻隔，则需要先设置防火墙规则，使其允许测试两端的iPerf进行通信。通常情况下，测试时我们会指定iPerf Server端监听的端口，比如设为 TCP-5201，那么在防火墙上就需要开启对应的TCP-5201端口的访问规则。

需要说明的是，iPerf 测试不可避免产生数据流量。如果测试时有进出业务的数据流量，很可能会产生相对应的影响。所以测试前务必做好测试计划，以免不必要的数据流量造成计划外的支出。

**测试带宽**

首先在 Server 端，我们运行以下命令使iPerf监听 8888 端口，每3秒输出一次结果。

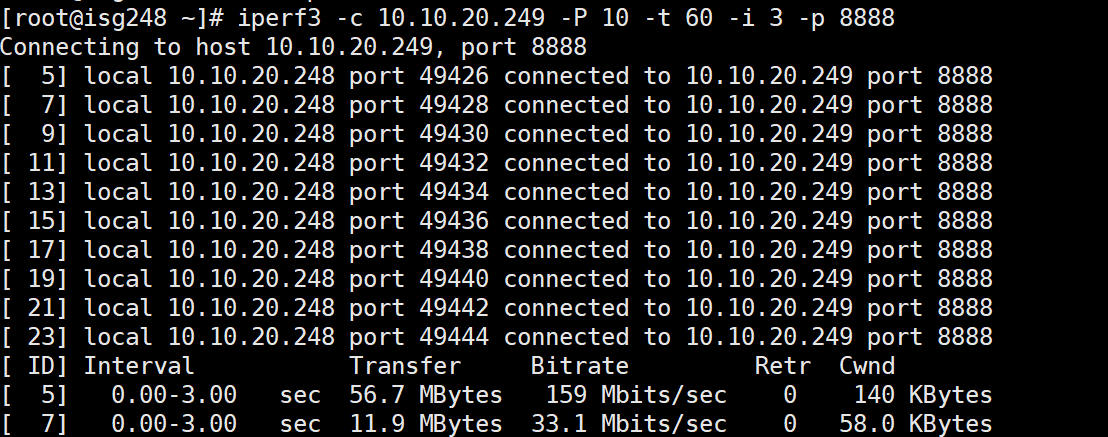
# iperf3 -s -p 8888 -i 3



当终端显示Server listening on 8888时，就表示Server已经正常运行，等待测试了。然后在Client端，我们并发10个数据流，测试总时长为60 秒，每3秒输出一次结果。以下为测试所使用的命令。

# iperf3 -c 10.10.20.249 -P 10 -t 60 -i 3 -p 8888

当Client端的命令执行后，测试就开始了。



终端会每隔3秒滚动显示测试的结果，最后还有整个测试的总结。

测试时，Server端和Client端都会输出测试结果。测试过程中，根据报告间隔时间的参数，终端会不断地显示当前时间间隔内测试结果。当测试结束后，iPerf 将测试结果汇总，输出最终测试结果。

下面是10个数据流测试过程中Server端的输出结果。

# iperf3 -s -p 8888 -i 3

|  |
| --- |
| root@isg249:~# iperf3 -s -p 8888 -i 3  -----------------------------------------------------------  Server listening on 8888  -----------------------------------------------------------  Accepted connection from 10.10.20.248, port 49424  [ 5] local 10.10.20.249 port 8888 connected to 10.10.20.248 port 49426  [ 8] local 10.10.20.249 port 8888 connected to 10.10.20.248 port 49428  [ 10] local 10.10.20.249 port 8888 connected to 10.10.20.248 port 49430  [ 12] local 10.10.20.249 port 8888 connected to 10.10.20.248 port 49432  [ 14] local 10.10.20.249 port 8888 connected to 10.10.20.248 port 49434  [ 16] local 10.10.20.249 port 8888 connected to 10.10.20.248 port 49436  [ 18] local 10.10.20.249 port 8888 connected to 10.10.20.248 port 49438  [ 20] local 10.10.20.249 port 8888 connected to 10.10.20.248 port 49440  [ 22] local 10.10.20.249 port 8888 connected to 10.10.20.248 port 49442  [ 24] local 10.10.20.249 port 8888 connected to 10.10.20.248 port 49444  [ ID] Interval Transfer Bitrate  [ 5] 0.00-3.00 sec 56.1 MBytes 157 Mbits/sec  [ 8] 0.00-3.00 sec 11.6 MBytes 32.5 Mbits/sec  [ 10] 0.00-3.00 sec 7.39 MBytes 20.7 Mbits/sec  [ 12] 0.00-3.00 sec 27.6 MBytes 77.1 Mbits/sec  [ 14] 0.00-3.00 sec 21.9 MBytes 61.3 Mbits/sec  [ 16] 0.00-3.00 sec 15.2 MBytes 42.5 Mbits/sec  [ 18] 0.00-3.00 sec 56.1 MBytes 157 Mbits/sec  [ 20] 0.00-3.00 sec 56.1 MBytes 157 Mbits/sec  [ 22] 0.00-3.00 sec 56.1 MBytes 157 Mbits/sec  [ 24] 0.00-3.00 sec 28.5 MBytes 79.8 Mbits/sec  [SUM] 0.00-3.00 sec 337 MBytes 941 Mbits/sec  - - - - - - - - - - - - - - - - - - - - - - - - -  ……  **#这里省略中间部分**  [ 5] 60.00-60.00 sec 91.9 KBytes 153 Mbits/sec  [ 8] 60.00-60.00 sec 59.4 KBytes 98.8 Mbits/sec  [ 10] 60.00-60.00 sec 5.66 KBytes 9.41 Mbits/sec  [ 12] 60.00-60.00 sec 42.4 KBytes 70.6 Mbits/sec  [ 14] 60.00-60.00 sec 0.00 Bytes 0.00 bits/sec  [ 16] 60.00-60.00 sec 26.9 KBytes 44.7 Mbits/sec  [ 18] 60.00-60.00 sec 96.2 KBytes 160 Mbits/sec  [ 20] 60.00-60.00 sec 96.2 KBytes 160 Mbits/sec  [ 22] 60.00-60.00 sec 91.9 KBytes 153 Mbits/sec  [ 24] 60.00-60.00 sec 50.9 KBytes 84.7 Mbits/sec  [SUM] 60.00-60.00 sec 561 KBytes 934 Mbits/sec  - - - - - - - - - - - - - - - - - - - - - - - - -  [ ID] Interval Transfer Bitrate  [ 5] 0.00-60.00 sec 1.10 GBytes 157 Mbits/sec receiver  [ 8] 0.00-60.00 sec 424 MBytes 59.3 Mbits/sec receiver  [ 10] 0.00-60.00 sec 90.8 MBytes 12.7 Mbits/sec receiver  [ 12] 0.00-60.00 sec 573 MBytes 80.1 Mbits/sec receiver  [ 14] 0.00-60.00 sec 347 MBytes 48.5 Mbits/sec receiver  [ 16] 0.00-60.00 sec 261 MBytes 36.4 Mbits/sec receiver  [ 18] 0.00-60.00 sec 1.10 GBytes 157 Mbits/sec receiver  [ 20] 0.00-60.00 sec 1.10 GBytes 157 Mbits/sec receiver  [ 22] 0.00-60.00 sec 1.10 GBytes 157 Mbits/sec receiver  [ 24] 0.00-60.00 sec 550 MBytes 76.8 Mbits/sec receiver  **[SUM] 0.00-60.00 sec 6.58 GBytes 941 Mbits/sec receiver** |

可以看到 iPerf 在不同的时间间隔内，都完整的显示了每个数据流传送的数据大小以及带宽。然后在第 11行显示当前时间间隔内的传送的总数据大小和总带宽。

下面是10个数据流测试过程中Client端的输出结果。

iperf3 -c 10.10.20.249 -P 10 -t 60 -i 3 -p 8888

|  |
| --- |
| [root@isg248 ~]# iperf3 -c 10.10.20.249 -P 10 -t 60 -i 3 -p 8888  Connecting to host 10.10.20.249, port 8888  [ 5] local 10.10.20.248 port 49426 connected to 10.10.20.249 port 8888  [ 7] local 10.10.20.248 port 49428 connected to 10.10.20.249 port 8888  [ 9] local 10.10.20.248 port 49430 connected to 10.10.20.249 port 8888  [ 11] local 10.10.20.248 port 49432 connected to 10.10.20.249 port 8888  [ 13] local 10.10.20.248 port 49434 connected to 10.10.20.249 port 8888  [ 15] local 10.10.20.248 port 49436 connected to 10.10.20.249 port 8888  [ 17] local 10.10.20.248 port 49438 connected to 10.10.20.249 port 8888  [ 19] local 10.10.20.248 port 49440 connected to 10.10.20.249 port 8888  [ 21] local 10.10.20.248 port 49442 connected to 10.10.20.249 port 8888  [ 23] local 10.10.20.248 port 49444 connected to 10.10.20.249 port 8888  [ ID] Interval Transfer Bitrate Retr Cwnd  [ 5] 0.00-3.00 sec 56.7 MBytes 159 Mbits/sec 0 140 KBytes  [ 7] 0.00-3.00 sec 11.9 MBytes 33.1 Mbits/sec 0 58.0 KBytes  [ 9] 0.00-3.00 sec 7.57 MBytes 21.2 Mbits/sec 0 33.9 KBytes  [ 11] 0.00-3.00 sec 28.3 MBytes 79.1 Mbits/sec 0 132 KBytes  [ 13] 0.00-3.00 sec 22.4 MBytes 62.6 Mbits/sec 0 90.5 KBytes  [ 15] 0.00-3.00 sec 15.5 MBytes 43.3 Mbits/sec 0 63.6 KBytes  [ 17] 0.00-3.00 sec 57.0 MBytes 159 Mbits/sec 0 139 KBytes  [ 19] 0.00-3.00 sec 56.8 MBytes 159 Mbits/sec 0 140 KBytes  [ 21] 0.00-3.00 sec 56.8 MBytes 159 Mbits/sec 0 141 KBytes  [ 23] 0.00-3.00 sec 29.3 MBytes 81.9 Mbits/sec 0 140 KBytes  [SUM] 0.00-3.00 sec 342 MBytes 956 Mbits/sec 0  - - - - - - - - - - - - - - - - - - - - - - - - -  ……  **# 这里省略中间部分**  [ 5] 57.00-60.00 sec 55.9 MBytes 156 Mbits/sec 0 177 KBytes  [ 7] 57.00-60.00 sec 26.1 MBytes 73.0 Mbits/sec 0 141 KBytes  [ 9] 57.00-60.00 sec 3.60 MBytes 10.1 Mbits/sec 0 33.9 KBytes  [ 11] 57.00-60.00 sec 28.7 MBytes 80.3 Mbits/sec 0 216 KBytes  [ 13] 57.00-60.00 sec 15.7 MBytes 43.8 Mbits/sec 0 105 KBytes  [ 15] 57.00-60.00 sec 10.8 MBytes 30.3 Mbits/sec 0 79.2 KBytes  [ 17] 57.00-60.00 sec 55.9 MBytes 156 Mbits/sec 0 139 KBytes  [ 19] 57.00-60.00 sec 56.3 MBytes 158 Mbits/sec 0 178 KBytes  [ 21] 57.00-60.00 sec 55.9 MBytes 156 Mbits/sec 0 141 KBytes  [ 23] 57.00-60.00 sec 27.3 MBytes 76.5 Mbits/sec 0 140 KBytes  [SUM] 57.00-60.00 sec 336 MBytes 941 Mbits/sec 0  - - - - - - - - - - - - - - - - - - - - - - - - -  [ ID] Interval Transfer Bitrate Retr  [ 5] 0.00-60.00 sec 1.10 GBytes 157 Mbits/sec 0 sender  [ 5] 0.00-60.00 sec 1.10 GBytes 157 Mbits/sec receiver  [ 7] 0.00-60.00 sec 425 MBytes 59.4 Mbits/sec 0 sender  [ 7] 0.00-60.00 sec 424 MBytes 59.3 Mbits/sec receiver  [ 9] 0.00-60.00 sec 91.0 MBytes 12.7 Mbits/sec 0 sender  [ 9] 0.00-60.00 sec 90.8 MBytes 12.7 Mbits/sec receiver  [ 11] 0.00-60.00 sec 574 MBytes 80.2 Mbits/sec 0 sender  [ 11] 0.00-60.00 sec 573 MBytes 80.1 Mbits/sec receiver  [ 13] 0.00-60.00 sec 348 MBytes 48.6 Mbits/sec 0 sender  [ 13] 0.00-60.00 sec 347 MBytes 48.5 Mbits/sec receiver  [ 15] 0.00-60.00 sec 261 MBytes 36.5 Mbits/sec 0 sender  [ 15] 0.00-60.00 sec 261 MBytes 36.4 Mbits/sec receiver  [ 17] 0.00-60.00 sec 1.10 GBytes 157 Mbits/sec 0 sender  [ 17] 0.00-60.00 sec 1.10 GBytes 157 Mbits/sec receiver  [ 19] 0.00-60.00 sec 1.10 GBytes 157 Mbits/sec 0 sender  [ 19] 0.00-60.00 sec 1.10 GBytes 157 Mbits/sec receiver  [ 21] 0.00-60.00 sec 1.10 GBytes 157 Mbits/sec 0 sender  [ 21] 0.00-60.00 sec 1.10 GBytes 157 Mbits/sec receiver  [ 23] 0.00-60.00 sec 550 MBytes 76.9 Mbits/sec 0 sender  [ 23] 0.00-60.00 sec 550 MBytes 76.8 Mbits/sec receiver  **[SUM] 0.00-60.00 sec 6.58 GBytes 942 Mbits/sec 0 sender**  **[SUM] 0.00-60.00 sec 6.58 GBytes 941 Mbits/sec receiver**  iperf Done. |

Client端的数据除了传输的数据大小和实时带宽，还有TCP重传次数（Retr）和窗口大小（Cwnd）。

这里有一个参数需要特别注意，那就是TCP窗口大小，可以使用-w参数指定。

网络通道的容量capacity = bandwidth \* round-trip time

而理论TCP窗口的大小就是网络通道的容量。

比如，网络带宽为40Mbit/s，回环路径消耗时间是2ms，那么TCP的窗口大小不小于40Mbit/s×2ms = 80kbit = 10Kbytes

此时我们可以查询iperf默认的TCP窗口大小来决定是否需要设置此参数，在此例中，窗口大小应设计大于10Kbytes，当然，这仅仅是理论值，在实际测试中可能需要作出调整。

**测试吞吐量，抖动和丢包率**

在实际测试时，可以通过多次调整并发数据流数量（经测试 FT2000+最大支持128个并发数 -P xxx），来获取真实的最大带宽值。除了并发数据流数量，TCP 窗口大小、最大分段长度、拥塞算法等等都会影响到应用程序实际能获取到的网络带宽。有经验的网络工程师也会通过数据计算，然后设置这些相关的参数进行测试，来获取到最大网络带宽。

iPerf同样也可以用于测量UDP数据包吞吐量、丢包和延迟指标。与TCP测试不同的是，UDP测试不采取尽可能快地发送流量的方式。与之相对的是，iPerf尝试发送1 Mbps的流量，这个流量是打包在1470字节的UDP数据包中（成为以太网的一帧）。我们可以通过指定一个目标带宽参数来增加数据量，单位可以是Kbps 或Mbps（-b #K 或 --b #M）。

# iperf3 -c 10.10.20.248 -P 10 -t 30 -i 3 -p 8888 -u

|  |
| --- |
| root@isg249:~# iperf3 -c 10.10.20.248 -P 10 -t 30 -i 3 -p 8888 -u  Connecting to host 10.10.20.248, port 8888  [ 5] local 10.10.20.249 port 59986 connected to 10.10.20.248 port 8888  [ 7] local 10.10.20.249 port 38428 connected to 10.10.20.248 port 8888  [ 9] local 10.10.20.249 port 54437 connected to 10.10.20.248 port 8888  [ 11] local 10.10.20.249 port 37744 connected to 10.10.20.248 port 8888  [ 13] local 10.10.20.249 port 47144 connected to 10.10.20.248 port 8888  [ 15] local 10.10.20.249 port 47419 connected to 10.10.20.248 port 8888  [ 17] local 10.10.20.249 port 55156 connected to 10.10.20.248 port 8888  [ 19] local 10.10.20.249 port 59870 connected to 10.10.20.248 port 8888  [ 21] local 10.10.20.249 port 58049 connected to 10.10.20.248 port 8888  [ 23] local 10.10.20.249 port 54965 connected to 10.10.20.248 port 8888  [ ID] Interval Transfer Bitrate Total Datagrams  [ 5] 0.00-3.00 sec 385 KBytes 1.05 Mbits/sec 272  [ 7] 0.00-3.00 sec 385 KBytes 1.05 Mbits/sec 272  [ 9] 0.00-3.00 sec 385 KBytes 1.05 Mbits/sec 272  [ 11] 0.00-3.00 sec 385 KBytes 1.05 Mbits/sec 272  [ 13] 0.00-3.00 sec 385 KBytes 1.05 Mbits/sec 272  [ 15] 0.00-3.00 sec 385 KBytes 1.05 Mbits/sec 272  [ 17] 0.00-3.00 sec 385 KBytes 1.05 Mbits/sec 272  [ 19] 0.00-3.00 sec 385 KBytes 1.05 Mbits/sec 272  [ 21] 0.00-3.00 sec 385 KBytes 1.05 Mbits/sec 272  [ 23] 0.00-3.00 sec 385 KBytes 1.05 Mbits/sec 272  [SUM] 0.00-3.00 sec 3.76 MBytes 10.5 Mbits/sec 2720  ……  **# 这里省略中间部分**  [ ID] Interval Transfer Bitrate Jitter Lost/Total Datagrams  [ 5] 0.00-30.00 sec 3.75 MBytes 1.05 Mbits/sec 0.000 ms 0/2716 (0%) sender  [ 5] 0.00-30.03 sec 3.75 MBytes 1.05 Mbits/sec 0.006 ms 0/2716 (0%) receiver  [ 7] 0.00-30.00 sec 3.75 MBytes 1.05 Mbits/sec 0.000 ms 0/2716 (0%) sender  [ 7] 0.00-30.03 sec 3.75 MBytes 1.05 Mbits/sec 0.028 ms 0/2716 (0%) receiver  [ 9] 0.00-30.00 sec 3.75 MBytes 1.05 Mbits/sec 0.000 ms 0/2716 (0%) sender  [ 9] 0.00-30.03 sec 3.75 MBytes 1.05 Mbits/sec 0.012 ms 0/2716 (0%) receiver  [ 11] 0.00-30.00 sec 3.75 MBytes 1.05 Mbits/sec 0.000 ms 0/2716 (0%) sender  [ 11] 0.00-30.03 sec 3.75 MBytes 1.05 Mbits/sec 0.002 ms 0/2716 (0%) receiver  [ 13] 0.00-30.00 sec 3.75 MBytes 1.05 Mbits/sec 0.000 ms 0/2716 (0%) sender  [ 13] 0.00-30.03 sec 3.75 MBytes 1.05 Mbits/sec 0.002 ms 0/2716 (0%) receiver  [ 15] 0.00-30.00 sec 3.75 MBytes 1.05 Mbits/sec 0.000 ms 0/2716 (0%) sender  [ 15] 0.00-30.03 sec 3.75 MBytes 1.05 Mbits/sec 0.003 ms 0/2716 (0%) receiver  [ 17] 0.00-30.00 sec 3.75 MBytes 1.05 Mbits/sec 0.000 ms 0/2716 (0%) sender  [ 17] 0.00-30.03 sec 3.75 MBytes 1.05 Mbits/sec 0.003 ms 0/2716 (0%) receiver  [ 19] 0.00-30.00 sec 3.75 MBytes 1.05 Mbits/sec 0.000 ms 0/2716 (0%) sender  [ 19] 0.00-30.03 sec 3.75 MBytes 1.05 Mbits/sec 0.003 ms 0/2716 (0%) receiver  [ 21] 0.00-30.00 sec 3.75 MBytes 1.05 Mbits/sec 0.000 ms 0/2716 (0%) sender  [ 21] 0.00-30.03 sec 3.75 MBytes 1.05 Mbits/sec 0.004 ms 0/2716 (0%) receiver  [ 23] 0.00-30.00 sec 3.75 MBytes 1.05 Mbits/sec 0.000 ms 0/2716 (0%) sender  [ 23] 0.00-30.03 sec 3.75 MBytes 1.05 Mbits/sec 0.003 ms 0/2716 (0%) receiver  [SUM] 0.00-30.00 sec 37.5 MBytes 10.5 Mbits/sec 0.000 ms 0/27160 (0%) sender  [SUM] 0.00-30.03 sec 37.5 MBytes 10.5 Mbits/sec 0.007 ms 0/27160 (0%) receiver  iperf Done. |

这样我们就可以看到吞吐量（间隔3秒测量的），以及丢包数（丢失的数据屯接收到的数据对比）和延迟（如jitter——在连续传输中的平滑平均值差）。延迟和丢失可以通过应用的改变而被兼容。比如，视频流媒体通过缓冲输入而能够容忍更多的延迟，而语音通讯则随着延迟增长性能下降明显。

UDP测试可以通过改变报文缓冲长度进行优化，长度单位为Kbytes 或 Mbytes（-l #K or #M）。与以太网帧的1500比特的MTU（最大转换单位）不同的是，802.11数据帧可以达到2304比特（在加密之前）。

但是，如果你正在测试的路径中包括Ethernet和802.11，那么要控制你的测试数据包长度，使它在一个Ethernet帧以内，以避免分片。

另一个有趣的iPerf UDP测试选项是服务类型（Type of Service，ToS），它的大小范围从0x10 (最小延迟) 到0x2 (最少费用)。在使用802.11e来控制服务质量的WLAN中，ToS是映射在Wi-Fi多媒体(WMM)存取范畴的。

正如我们所看到的，iPerf简化了对基于TCP数据流应用和UDP数据包应用的端对端性能测定。然而，iPerf仍然无法模拟所有类型的应用——比如，对于交互式上网的模拟就不是很好。同时，用于iPerf Wi-Fi测试的WLAN适配器也会影响你的测试——为了获得更好的测试结果，我们可以配置一个类似于“实际”用户的有代表性的适配器。

# FIO使用说明

## FIO简介

FIO是测试IOPS的非常好的工具，用来对硬件进行压力测试和验证，最新版本支持18种不同的I/O引擎，包括:sync,mmap, libaio, posixaio, SG v3, splice, null, network, syslet, guasi, solarisaio等等。

FIO是一个会产生一系列线程或进程来处理用户指定I/O动作类型的工具，最典型的应用是写一个工作文件来模拟匹配I/O负载。注意：不要对有数据的磁盘或者分区做测试，会破坏已存在的数据。

FIO是一个非常简单好用的性能测试工具，它能对裸设备、块设备、和带文件系统的设备进行测试(当然裸设备需要使用RAW提前挂载。)。

1）支持十几种存储引擎，可以自定义  
 2）自带做图工具，调用gnuplot做图  
 3）支持几乎所有的存储描述参数  
 4）大量对CPU，内存，进程/线程，文件，IO特性的配置  
 5）压缩，trace回放等这些都包含，灵活的配置

**硬盘性能衡量指标**

* IOPS：每秒的读写次数，单位为次（计数）。存储设备的底层驱动类型决定了不同的IOPS
  + 总IOPS：每秒执行的I/O操作总次数
  + 随机读IOPS：每秒指定的随机读I/O操作的平均次数
  + 随机写IOPS 每秒指定的随机写I/O操作的平均次数
  + 顺序读IOPS 每秒指定的顺序读I/O操作的平均次数
  + 顺序写IOPS 每秒指定的顺序写I/O操作的平均次数
* 吞吐量：每秒的读写数据量，单位为MB/S  
  吞吐量市值单位时间内可以成功传输的数据数量。  
  如果需要部署大量顺序读写的应用，典型场景比如hadoop离线计算型业务，需要关注吞吐量
* 时延：IO操作的发送时间到接收确认所经过的时间，单位为秒  
  如果应用对时延比较敏感，比如数据库（过高时延会导致应用性能下降或报错），建议使用SSD存储

## FIO安装

下载FIO的网址是：

<https://brick.kernel.dk/snaps/>

**注意: 性能测试建议直接通过写裸盘的方式进行测试，会得到较为真实的数据。但直接测试裸盘会破坏文件系统结构，导致数据丢失，请在测试前确认磁盘中数据已备份。**

CentOS可以直接使用yum安装，麒麟系统可以使用apt-get安装，我们这里以源码方式进行安装。

# wget https://brick.kernel.dk/snaps/fio-3.16.tar.gz

# tar -xvf fio-3.16.tar.gz

# cd fio-3.16

# ./configure

# make && make install

## FIO测试

### FIO参数说明

可以使用fio -help查看每个参数详细说明，具体的参数可以在官网查看how to文档，如下为几个常见的参数描述

|  |
| --- |
| filename=/dev/sda　 支持文件系统或者裸设备，-filename=/dev/sda2或-filename=/dev/sdb  direct=1 测试过程绕过机器自带的buffer，使测试结果更真实  rw=randwread 测试随机读的I/O  rw=randwrite 测试随机写的I/O  rw=randrw 测试随机混合写和读的I/O  rw=read 测试顺序读的I/O  rw=write 测试顺序写的I/O  rw=rw 测试顺序混合写和读的I/O  bs=4k 单次io的块文件大小为4k  bsrange=512-2048 同上，提定数据块的大小范围  size=5g 本次的测试文件大小为5g，以每次4k的io进行测试  numjobs=30 本次的测试线程为30  runtime=1000 测试时间为1000秒，如果不写则一直将5g文件分4k每次写完为止  ioengine=psync io引擎使用pync方式，如果要使用libaio引擎，需要yum install libaio-devel包  rwmixwrite=30 在混合读写的模式下，写占30%  group\_reporting 关于显示结果的，汇总每个进程的信息  此外  lockmem=1g 只使用1g内存进行测试  zero\_buffers 用0初始化系统buffer  nrfiles=8 每个进程生成文件的数量 |

### FIO测试

**100%随机，100%读， 4K**

fio -filename=/dev/sdb -direct=1 -iodepth 1 -thread -rw=randread -ioengine=psync -bs=4k -size=100G -numjobs=50 -runtime=180 -group\_reporting -name=rand\_100read\_4k

**100%随机，100%写，4K**

fio -filename=/dev/sdb -direct=1 -iodepth 1 -thread -rw=randwrite -ioengine=psync -bs=4k -size=100G -numjobs=50 -runtime=180 -group\_reporting -name=rand\_100write\_4k

**100%顺序，100%读 ，4K**

fio -filename=/dev/sdb -direct=1 -iodepth 1 -thread -rw=read -ioengine=psync -bs=4k -size=100G -numjobs=50 -runtime=180 -group\_reporting -name=sqe\_100read\_4k

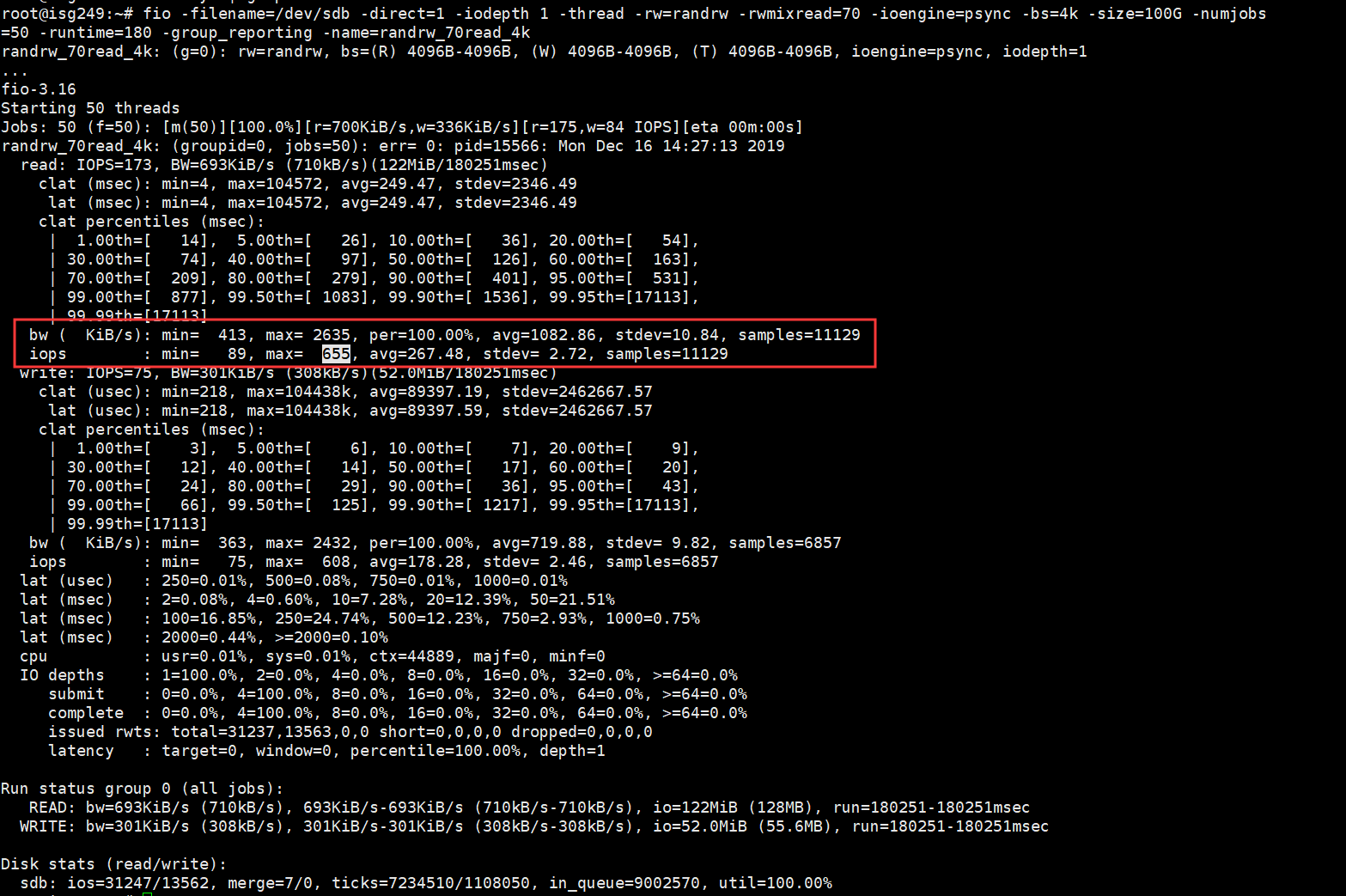
**100%顺序，100%写 ，4K**

fio -filename=/dev/sdb -direct=1 -iodepth 1 -thread -rw=write -ioengine=psync -bs=4k -size=100G -numjobs=50 -runtime=180 -group\_reporting -name=sqe\_100write\_4k

**混合随机读写，100%随机，70%读，30%写，4K**

fio -filename=/dev/sdb -direct=1 -iodepth 1 -thread -rw=randrw -rwmixread=70 -ioengine=psync -bs=4k -size=100G -numjobs=50 -runtime=180 -group\_reporting -name=randrw\_70read\_4k

以下是混合读写的测试结果



测试结果如上，主要关注 bw 和 iops 结果

* bw：磁盘的吞吐量，这个是顺序读写考察的重点
* iops：磁盘的每秒读写次数，这个是随机读写考察的重点

### 测试场景

**顺序读写** （吞吐量，常用单位为 MB/s）：文件在硬盘上存储位置是连续的。

适用场景：大文件拷贝（比如视频音乐）。速度即使很高，对数据库性能也没有参考价值。

**4K 随机读写** （IOPS，常用单位为次）：在硬盘上随机位置读写数据，每次 4KB。

适用场景：操作系统运行、软件运行、数据库。