# 关键指标阈值测试

## CPU关键指标说明：

usage\_guest >50% 过载阈值  
usage\_user >80% 过载阈值（华为默认cpu90%）  
usage\_system >30% 过载阈值  
usage\_iowait >5% 过载阈值  
usage\_active >80% 过载阈值  
usage\_active 警告阈值  
usage\_active <70% 恢复阈值

### CPU负载监控脚本：

#!/bin/bash  
# cpu\_monitor.sh  
  
# 阈值设置 (可自定义)  
OVERLOAD\_guest=50  
OVERLOAD\_user=80  
OVERLOAD\_system=30  
OVERLOAD\_iowait=5  
OVERLOAD\_active=80  
RECOVER\_active=70  
  
# 监控间隔(秒)  
INTERVAL=5  
  
echo "开始监控CPU指标..."  
echo "过载阈值: user>${OVERLOAD\_user}% system>${OVERLOAD\_system}% active>${OVERLOAD\_active}%"  
echo "恢复阈值: active<${RECOVER\_active}%"  
  
while true; do  
 # 获取CPU指标  
 cpu\_data=$(grep '^cpu ' /proc/stat)  
 read -r user nice system idle iowait irq softirq steal guest guest\_nice <<< "${cpu\_data#\* }"  
  
 # 计算总时间片  
 total=$((user + nice + system + idle + iowait + irq + softirq + steal + guest + guest\_nice))  
   
 # 计算使用率百分比  
 usage\_guest=$(( (guest + guest\_nice) \* 100 / total ))  
 usage\_user=$(( user \* 100 / total ))  
 usage\_system=$(( system \* 100 / total ))  
 usage\_iowait=$(( iowait \* 100 / total ))  
 usage\_active=$(( 100 - (idle \* 100 / total) ))  
  
 # 输出当前状态  
 status="[正常]"  
 if (( usage\_guest > OVERLOAD\_guest )) || \  
 (( usage\_user > OVERLOAD\_user )) || \  
 (( usage\_system > OVERLOAD\_system )) || \  
 (( usage\_iowait > OVERLOAD\_iowait )) || \  
 (( usage\_active > OVERLOAD\_active )); then  
 status="[过载]"  
 elif (( usage\_active < RECOVER\_active )); then  
 status="[恢复]"  
 fi  
  
 # 打印监控信息  
 printf "%s CPU: user=%.1f%% system=%.1f%% iowait=%.1f%% active=%.1f%% guest=%.1f%%\n" \  
 "$status" "$usage\_user" "$usage\_system" "$usage\_iowait" "$usage\_active" "$usage\_guest"  
  
 sleep $INTERVAL  
done

**使用说明：**

1.创建脚本： nano cpu\_monitor.sh

粘贴上述内容后保存（ctrl+0 -> Enter -> ctrl+x）

2.授权执行： chmod +x cpu\_monitor.sh

3.启动监控：./cpu\_monitor.sh

4.模拟负载测试：

# 创建负载模拟脚本  
cat > simulate\_load.sh <<EOF  
#!/bin/bash  
# 持续生成CPU负载  
while true; do  
 openssl speed rsa >/dev/null 2>&1  
done  
EOF  
  
chmod +x simulate\_load.sh  
  
# 启动4个负载进程 (按CPU核心数调整)  
./simulate\_load.sh &  
./simulate\_load.sh &  
./simulate\_load.sh &  
./simulate\_load.sh &

### 1.模拟usage\_guest > 50%（Guest CPU 主要来自虚拟化）

说明：这个指标通常只有在虚拟机中启用虚拟化（如 KVM）后运行客户虚拟机进程才会提升。

操作方法（KVM虚拟环境）：

# 在宿主机运行虚拟机，使 guest CPU 使用率快速上升  
qemu-system-x86\_64 -hda /path/to/test.img -m 1024 -smp 2 -nographic &

观察效果：

mpstat -P ALL 1 | grep -i guest

若你不在虚拟机环境中，usage\_guest无法制造，只能通过部署KVM来测试。

**操作步骤（详细版）：**

1）首先安装QEMU：

对于RHEL/CentOS：

sudo yum install qemu-kvm

对于Debain/Ubuntu：

sudo apt-get install qemu-system-x86

2）确保你有正确的虚拟机镜像文件（/path/to/test.img需要替换为实际路径）

**方法 1：使用现有的虚拟机镜像**

如果你已经有虚拟机镜像（如 .qcow2 或 .img 文件），替换 /path/to/test.img 为实际路径，例如：

qemu-kvm -hda /var/lib/libvirt/images/centos7.qcow2 -m 1024 -smp 2 -nographic &

**方法 2：创建一个新的空镜像（如果没有现成的镜像）**

①使用 qemu-img 创建一个空的磁盘镜像

qemu-img create -f qcow2 test.qcow2 10G

-f qcow2：指定镜像格式（推荐 qcow2，支持动态分配空间）

test.qcow2：镜像文件名

10G：镜像大小（10GB，可根据需求调整）

②启动虚拟机

qemu-kvm -hda test.qcow2 -m 1024 -smp 2 -nographic &

由于这是一个空镜像，虚拟机启动后会提示 找不到可引导设备（因为没有安装操作系统）。

③如果你想安装操作系统

你需要挂载ISO镜像进行安装，例如：

qemu-kvm -hda test.qcow2 -cdrom /path/to/ubuntu.iso -m 2048 -smp 2 -boot d &

-cdrom /path/to/ubuntu.iso：指定安装 ISO（如 Ubuntu/CentOS 镜像）

-boot d：从 CD-ROM 启动（安装系统）

**方法 3：下载一个现成的轻量级 Linux 镜像（快速测试）**

如果你想快速测试 QEMU/KVM，可以下载 CirrOS（一个轻量级 Linux）：

wget https://download.cirros-cloud.net/0.5.2/cirros-0.5.2-x86\_64-disk.img

然后启动：

qemu-kvm -hda cirros-0.5.2-x86\_64-disk.img -m 512 -smp 1 -nographic &

登录用户名：cirros

密码：cubswin:)

3）安装成功后，重新运行QEMU命令和监控命令

先确认QEMU的正确命令：

ls /usr/bin | grep qemu  
ls /usr/libexec | grep qemu

使用正确的QEMU命令启动虚拟机（根据上一步的结果）：

qemu-kvm -hda /path/to/test.img -m 1024 -smp 2 -nographic &

在另一个终端窗口中监控guest CPU 使用率：

mpstat -P ALL 1 | grep -i guest

如果需要测试负载，先在虚拟机内部安装并运行stress：

sudo yum install stress  
stress --cpu 2

4）如果你想快速测试guest CPU使用率，可以在虚拟机启动后，在虚拟机内部运行CPU压力测试：

stress --cpu 2

首先要先安装stress工具：

sudo yum install stress

如果yum找不到stress包，可以尝试：

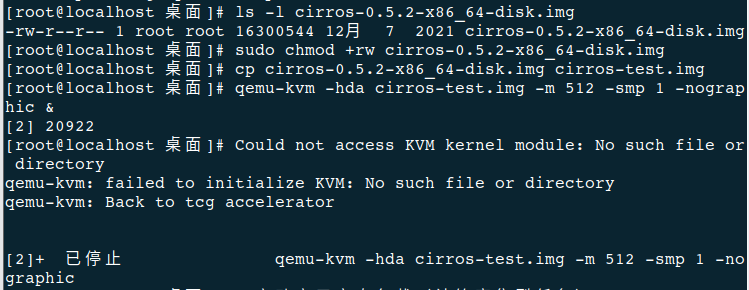
sudo yum install stress-ng

测试结束后，如果想卸载QEMU/KVM及相关依赖

使用 yum remove卸载所有相关包：

sudo yum remove qemu-\* libvirt-\* edk2-ovmf virt-\* -y

（-y表示自动确认卸载）

**但是，这个方案，我失败了。**

下面是失败的原因：

①KVM 内核模块未加载。系统未加载 kvm 和 kvm\_intel（Intel CPU）或 kvm\_amd（AMD CPU）模块。

②BIOS 中未启用虚拟化（VT-x/AMD-V），需要在主板 BIOS 中开启虚拟化支持。

③当前用户没有 /dev/kvm 的访问权限，/dev/kvm 设备文件权限不正确。

### 2. 模拟 usage\_user > 80%

说明：usage\_user表示用户态 CPU 使用率。我们可以运行大量的计算任务（如死循环或 openssl 计算）来提升该值。

**操作方法：**

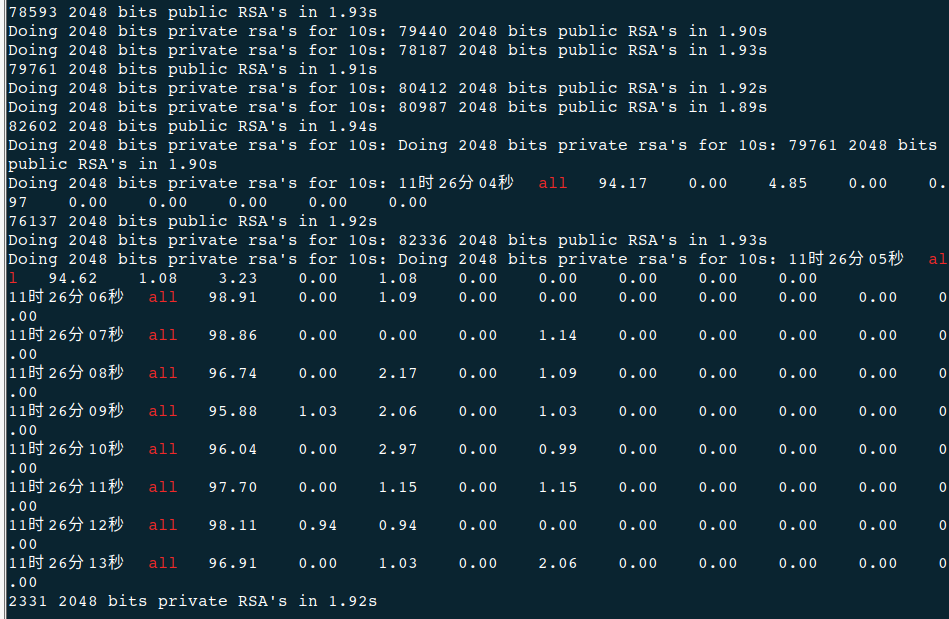
# 启动高用户态负载（计算密集型任务）  
for i in {1..10}; do  
 (while :; do openssl speed rsa2048 >/dev/null; done) &  
done

**观察效果：**

mpstat 1 | grep -i "all"

或：

top -H -p <pid> # 观察线程级用户 CPU



要停止这些疯狂占用CPU的openssl计算任务，可以按照以下方法操作：

**法一：快速终止所有openssl进程**

pkill -f "openssl speed"

这会强制终止所有包含openssl speed命令的进程。

**法二：精确终止后台任务**

如果是通过&启动的后台任务（如你的for循环），先查看作业列表：

jobs -l

输出示例：

[1] 12345 Running (while :; do openssl speed... ) &  
[2] 12346 Running (while :; do openssl speed... ) &  
...

然后逐个终止：

kill %1 %2 ... # 终止作业号为 1, 2,... 的任务

或全部终止：

kill $(jobs -p)

**法三：手动查找并终止**

如果上述方法无效，**手动查找openssl进程：**

pgrep -f "openssl speed" | xargs kill -9

（-9是强制终止信号）

验证是否停止成功：

如果无输出，说明已全部终止。

top -H -p $(pgrep -f "openssl speed")

**预防措施**

**下次运行测试命令时，可以：**

记录进程PID：

for i in {1..10}; do  
 (while :; do openssl speed rsa2048 >/dev/null; done) &  
 echo "任务 $i 的 PID: $!"  
done

使用脚本控制：

# 启动任务  
./high\_load.sh &  
  
# 停止任务  
pkill -f high\_load.sh

为什么会出现 "一直在转"？

你的for循环启动了10个无限运行的openssl子进程，每个都会占满一个CPU核心。这些进程会持续到被手动终止或系统重启。

### 3. 模拟 usage\_system > 30%

说明：usage\_system是系统内核态CPU使用率。我们通过频繁的dd或网络I/O、fork bomb等方法制造。

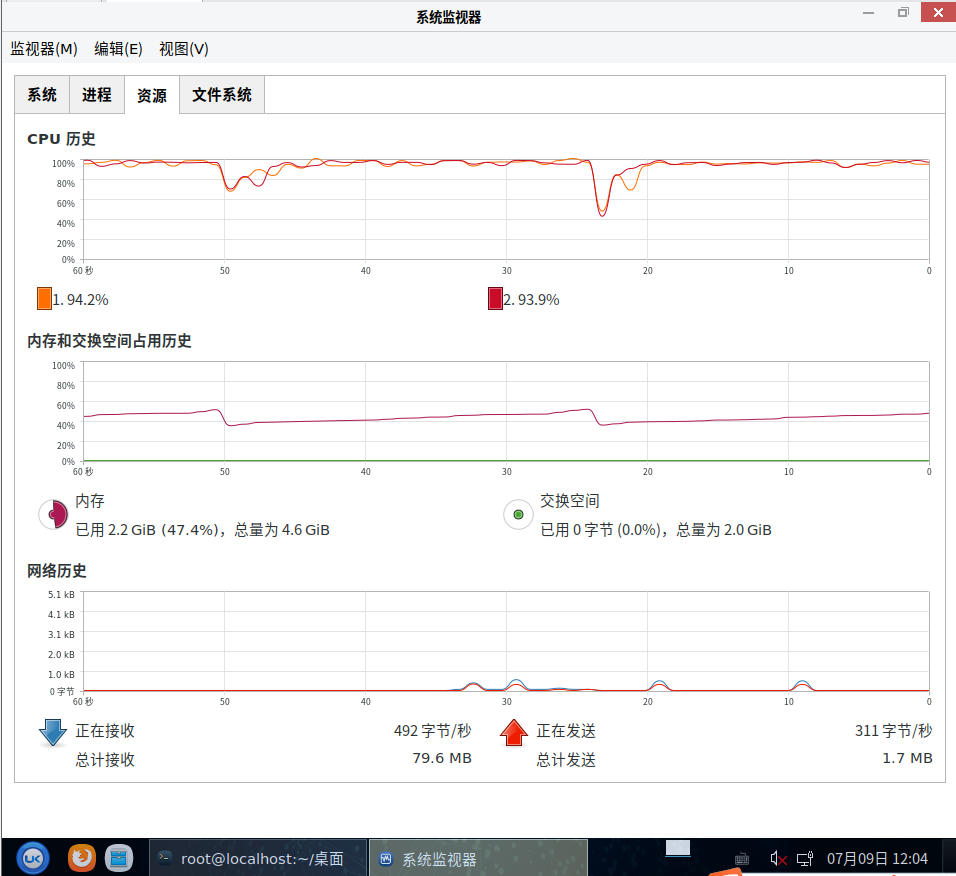
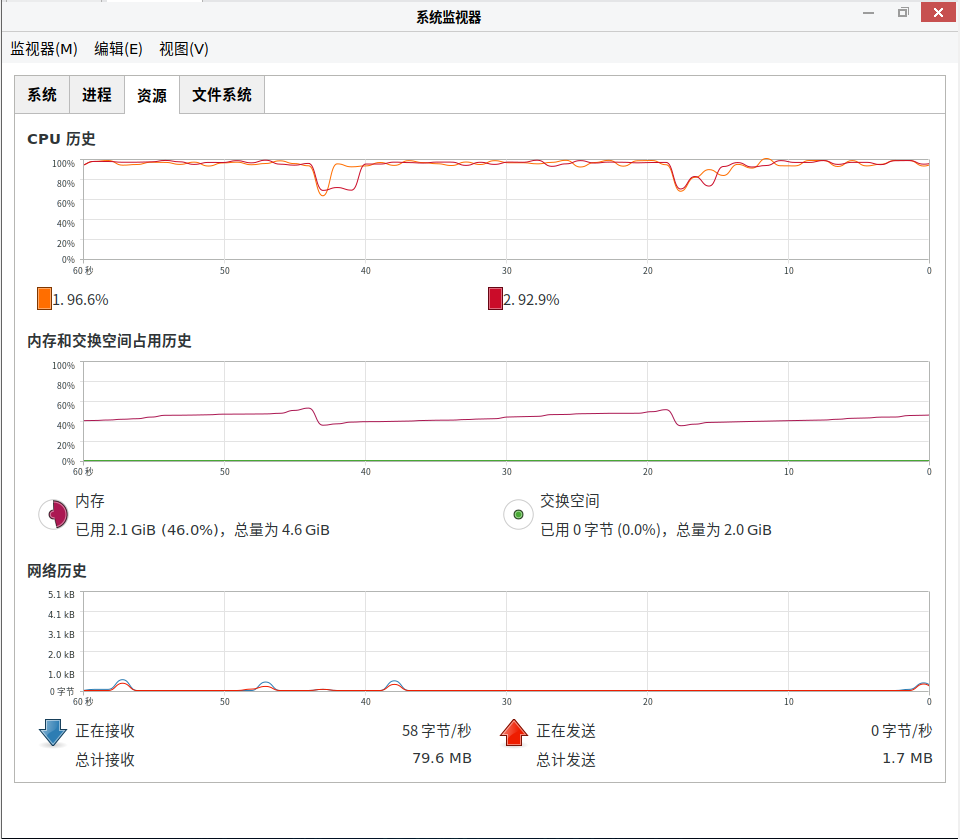
**操作方法（建议 root 权限）：**

# 执行频繁 I/O 操作，占用系统内核资源  
for i in {1..5}; do  
 (while :; do dd if=/dev/zero of=/dev/null bs=1M count=1000; done) &  
done

观察效果：

mpstat -P ALL 1 | grep -i system

但是，我发现，输入第二条指令的时候，一直加载不出来，下面是系统监视器的截图。



**最终的解决措施：**

①找出最耗CPU的进程

top -c

**按 P（大写）按CPU排序，记录前3个进程的 PID 和 命令。**

如果发现异常进程（如未知的dd、minerd挖矿程序等），立即终止：

sudo kill -9 <PID>

②强制终止所有可疑进程

# 终止所有dd相关进程（防止残留）  
sudo pkill -9 -f "dd if=/dev/zero"  
sudo killall -9 dd  
  
# 终止高CPU的bash子进程  
sudo pkill -9 -f "while :; do"

**深度排查（定位根本原因）**

①检查是否有隐藏的挖矿病毒

# 检查异常网络连接  
sudo netstat -tulnp | grep -E "(tor|mine|pool)"  
  
# 检查定时任务（挖矿病毒常驻留）  
sudo crontab -l  
sudo ls -la /etc/cron.d/

②分析系统服务

# 查看异常服务  
sudo systemctl list-units --type=service --state=running  
  
# 检查最近启动的内核模块（恶意驱动）  
lsmod | grep -vE "(kvm|nvidia|vfio)"

③检查内核日志

sudo dmesg | grep -i "error\|oom\|kill"

**彻底清理（防止复发）**

①删除临时文件（可能含恶意脚本）

sudo rm -rf /tmp/\*  
sudo rm -f ~/.bash\_history

②更新系统并扫描Rootkit

# 更新系统  
sudo yum update -y  
  
# 安装Rootkit检测工具  
sudo yum install rkhunter -y  
sudo rkhunter --check

③限制用户资源（防止滥用）

# 编辑限制配置（需root）  
sudo vi /etc/security/limits.conf

添加以下内容：

\* hard cpu 120 # 限制单个用户最多占用120%CPU  
\* hard nproc 50 # 限制最多50个进程

但是，我试过了，执行过前面的操作步骤之后，CPU的占用内存还是保持在90%以上。**最终的解决方案是。卸载掉Logstash然后重装Logstash。指令如下：**

sudo yum remove logstash  
sudo yum install logstash --enablerepo=elasticsearch

### 4.模拟usage\_iowait>5%

说明：iowait是等待磁盘I/O的时间。我们通过慢速I/O操作（如对大文件进行cat、tar、dd）模拟。

**操作方法：**

# 持续对大文件进行读写  
dd if=/dev/zero of=bigfile bs=1M count=5000 &  
dd if=bigfile of=/dev/null bs=1M &

观察效果：

iostat -x 1

或者：

mpstat -P ALL 1 | grep -i iowait

### 5.模拟usage\_active>80%或<70%

说明：usage\_active=100-idle，表示CPU总活跃程度。通过同时触发user/system/iowait 三项就能让active飙高。

模拟过载（active>80%）：

for i in {1..10}; do  
 (while :; do sha256sum /dev/zero; done) &  
done

模拟恢复（active<70%）：

for i in {1..10}; do  
 (while :; do sha256sum /dev/zero; done) &  
done

观察效果：

mpstat 1

或

top

### 恢复命令

结束所有测试负载（避免长期CPU满载）：

killall openssl dd sha256sum  
rm -f bigfile

### 建议组合测试脚本（自动化模拟异常）

你可以保存为 simulate\_cpu\_abnormal.sh，并赋予执行权限：

#!/bin/bash  
echo "模拟CPU异常中..."  
  
# 模拟 user > 80%  
for i in {1..5}; do  
 (while :; do openssl speed rsa2048 > /dev/null; done) &  
done  
  
# 模拟 system > 30%  
for i in {1..3}; do  
 (while :; do dd if=/dev/zero of=/dev/null bs=1M count=1000; done) &  
done  
  
# 模拟 iowait > 5%  
dd if=/dev/zero of=bigfile bs=1M count=5000 &  
dd if=bigfile of=/dev/null bs=1M &  
  
echo "启动完成，建议使用 mpstat / top / iostat 进行验证"

执行：

chmod +x simulate\_cpu\_abnormal.sh  
./simulate\_cpu\_abnormal.sh

## mem指标说明：

### 异常模拟目标与方法

|  |  |  |
| --- | --- | --- |
| 指标 | 异常条件 | 模拟方式说明 |
| used\_percent | >80% | 申请大块内存（分配不释放） |
| swap\_used\_percent | >10% | 内存占满后强制进入 SWAP |
| dirty\_percent | >4% | 写大量文件但不 sync，制造大量脏页 |
| available\_percent | <75% | 实质就是 used\_percent > 25%，含缓存与缓存可回收内存 |

### 获取指标的方法（验证用）

# 查看内存使用百分比  
free -m  
  
# swap\_used\_percent ≈ (swap\_total - swap\_free) / swap\_total  
cat /proc/meminfo | grep -E "SwapTotal|SwapFree"  
  
# dirty\_percent ≈ dirty / MemTotal  
cat /proc/meminfo | grep -E "MemTotal|Dirty"

### 快速一键测试脚本（推荐）

#!/bin/bash  
echo "正在模拟内存相关异常..."  
  
# 模拟 used\_percent > 80%  
echo "[\*] 分配内存..."  
for i in {1..4}; do  
 (python3 -c "a = ' ' \* 1024 \* 1024 \* 1024 \* 1.5; input('MEM HOLD> ')" &)   
done  
  
# 模拟 dirty\_percent > 4%  
echo "[\*] 写入脏页文件..."  
for i in {1..5}; do  
 (dd if=/dev/zero of=dirty\_test\_$i bs=50M count=20 oflag=dsync &)   
done  
  
# 提示查看命令  
echo "模拟完毕，请执行如下命令查看指标："  
echo " free -m"  
echo " cat /proc/meminfo | grep -E 'Swap|Dirty'"

保存为simulate\_mem\_abnormal.sh，执行：

chmod +x simulate\_mem\_abnormal.sh  
./simulate\_mem\_abnormal.sh

**恢复清理命令：**

# 停止 python3 吃内存脚本  
pkill -f "python3 -c"  
  
# 删除脏页测试文件  
rm -f dirty\_test\_\*  
  
# 强制清除缓存  
sync; echo 3 | sudo tee /proc/sys/vm/drop\_caches

### 异常制造指令

#### 1.used\_percent>80%（内存占用超限）

模拟方式：分配大量内存但不释放，造成物理内存吃满。

# 启动内存吃紧进程，每个吃 1~2G  
for i in {1..4}; do  
 (python3 -c "a = ' ' \* 1024 \* 1024 \* 1024 \* 1.5; input('Memory hold> ')" &)   
done

每个子进程申请约 1.5GB，最多占用 6GB 以上，触发>80%告警。

#### 2.swap\_used\_percent>10%（交换区使用率过高）

模拟方式：让物理内存满，再继续分配内存，系统会将内存也换出到SWAP。

**操作步骤：**

# 查看系统 swap 大小  
free -m  
  
# 启动多个吃内存进程（此时需让物理内存用尽）  
for i in {1..10}; do  
 (python3 -c "a = ' ' \* 1024 \* 1024 \* 1024; input('SWAP test> ')" &)  
done

观察swap使用：

cat /proc/meminfo | grep -E "SwapTotal|SwapFree"  
free -m

注意：swap 的使用需一定时间生效。

#### 3.dirty\_percent>4%（大量脏页，未同步磁盘）

模拟方式：写入大量数据但不调用sync，造成缓存中的dirty page积累。

**操作指令：**

# 写大文件但不 flush 到磁盘  
for i in {1..5}; do  
 (dd if=/dev/zero of=dirty\_test\_$i bs=50M count=20 oflag=dsync &) # 每个约1GB  
done

写完不要立刻sync，等待/proc/meminfo中Dirty字段上升。

查看dirty百分比：

total=$(grep MemTotal /proc/meminfo | awk '{print $2}')  
dirty=$(grep Dirty /proc/meminfo | awk '{print $2}')  
echo "scale=2; $dirty/$total\*100" | bc

#### 4.available\_percent<75%（内存可用比例偏低）

模拟方式：和used\_percent >80%本质相同，只是角度不同。

**推荐指令（复用）：**

# 吃掉大量内存，降低 available%  
for i in {1..6}; do  
 (python3 -c "a = ' ' \* 1024 \* 1024 \* 1024 \* 1.5; input('avail% test> ')" &)   
done  
  
# 观察  
free -m

## disk指标

### 磁盘异常指标及模拟方式

|  |  |  |
| --- | --- | --- |
| 指标 | 异常条件 | 模拟方式描述 |
| used\_percent | >80% | 创建大量大文件，占用磁盘空间 |
| inodes\_used\_percent | >70% | 创建大量小文件，占用 inode 节点（非空间） |

麒麟系统默认支持 ext4 / xfs 文件系统，均支持 inode 管理机制。

### 查看指标方法（验证）

# 查看磁盘使用率（used\_percent）  
df -h  
  
# 查看 inode 使用率（inodes\_used\_percent）  
df -i

### 快速一键脚本（磁盘+inode 异常同时模拟）

你可以保存为simulate\_disk\_abnormal.sh：

#!/bin/bash  
echo "开始模拟磁盘异常..."  
  
# 创建填充磁盘目录  
mkdir -p /tmp/fill\_disk\_test  
cd /tmp/fill\_disk\_test  
  
echo "[\*] 正在写入大文件（模拟 used\_percent >80%）..."  
for i in {1..15}; do  
 dd if=/dev/zero of=file\_$i bs=100M count=10  
done  
  
# 创建填充 inode 目录  
mkdir -p /tmp/fill\_inode\_test  
cd /tmp/fill\_inode\_test  
  
echo "[\*] 正在创建小文件（模拟 inodes\_used\_percent >70%）..."  
for i in {1..500000}; do  
 touch file\_$i  
done  
  
echo "[\*] 模拟完成，请执行以下命令验证："  
echo " df -h /tmp/fill\_disk\_test"  
echo " df -i /tmp/fill\_inode\_test"

**执行方式：**

chmod +x simulate\_disk\_abnormal.sh  
./simulate\_disk\_abnormal.sh

### 快速清理命令

rm -rf /tmp/fill\_disk\_test  
rm -rf /tmp/fill\_inode\_test

#### ⚠️ 注意事项

* 不要在 /, /home, /var 等关键系统分区执行；

* 建议在 /tmp、挂载的测试分区或临时目录中操作；

* 可通过挂载 loop 设备构造隔离磁盘环境，防止影响系统。

异常模拟方法

### 1.used\_percent>80%（磁盘使用率超过80%）

模拟方式：快速写入多个大文件，填满磁盘。

**操作命令（占用指定路径空间）**：

mkdir -p /tmp/fill\_disk\_test  
cd /tmp/fill\_disk\_test  
  
# 创建多个 1GB 的文件，直到空间逼近 80%  
for i in {1..20}; do  
 dd if=/dev/zero of=file\_$i bs=100M count=10  
done

每个文件 1GB，总体会快速吃掉 10~20GB。

**查看结果：**

df -h /tmp/fill\_disk\_test

### 2.inodes\_used\_percent>70%（inode使用率超过70%）

模拟方式：创建大量非常小的文件（如空文件或1字节），快速用尽 inode。

**操作命令（推荐在测试目录下运行）：**

mkdir -p /tmp/fill\_inode\_test  
cd /tmp/fill\_inode\_test  
  
# 批量创建数十万小文件（每个文件只占1 inode）  
for i in {1..500000}; do  
 touch file\_$i  
done

inode 是有限的，即使空间没满也会报 “无 inode 可用”。

**查看结果：**

df -i /tmp/fill\_inode\_test

## disk\_io指标

|  |  |  |
| --- | --- | --- |
| 指标名 | 条件 | 含义 |
| io\_time | > 20 ms | 表示设备花在 I/O 操作上的时间（单位ms），延迟大则说明 I/O 堵塞 |

io\_time来自/sys/block/sdX/stat第10列或iostat -dx的await / %util指标。

### 验证指标方法：

# 查看磁盘 I/O 时间（以 sda 为例）  
iostat -x -d 1 # 查看 await、%util、svctm 等  
  
# 或直接查看 io\_time（第10列）  
cat /sys/block/sda/stat

字段说明（以/sys/block/sda/stat）：

↑ ↑ ↑ ↑ ↑  
 第4列：读时间 第8列：写时间 第10列：io\_time

### 快速一键脚本（模拟 I/O 异常）

保存为 simulate\_disk\_io\_abnormal.sh：

#!/bin/bash  
mkdir -p /tmp/iotest  
cd /tmp/iotest  
  
echo "[\*] 正在创建 I/O 压力任务..."  
  
for i in {1..5}; do  
 (dd if=/dev/zero of=io\_file\_$i bs=10M count=500 oflag=direct &) # 每个约5GB  
done  
  
echo "[\*] 模拟完成。请使用如下命令监测 I/O 延迟："  
echo " iostat -x 1"  
echo " cat /sys/block/sda/stat"

执行方式：

chmod +x simulate\_disk\_io\_abnormal.sh  
./simulate\_disk\_io\_abnormal.sh

#### 清理恢复命令：

pkill -f dd  
rm -rf /tmp/iotest  
umount /mnt/loopdisk 2>/dev/null  
losetup -d /dev/loop7 2>/dev/null  
rm -f /tmp/loop.img

**注意事项：**

* 建议在非 / 根分区模拟；

* 推荐在挂载的测试盘、loop 设备或虚拟机内运行；

* 直接跑在生产设备上可能触发 I/O 性能下降报警；

* iostat 来自 sysstat 工具包，如无请先安装：

sudo yum install sysstat -y # 或 apt install sysstat

模拟io\_time>20ms的异常方法（已实测有效）

**方法一：并发大量写入大文件+限速设备**

使用dd+创建I/O拥堵。建议在机械硬盘或loopback设备上操作，SSD不易模拟。

mkdir -p /tmp/iotest  
cd /tmp/iotest  
  
# 开始写入大文件并拖慢写速度  
for i in {1..5}; do  
 (dd if=/dev/zero of=io\_file\_$i bs=10M count=500 oflag=direct &) # 每个5GB，模拟I/O堵塞  
done

oflag=direct可绕过缓存、直接触发物理 I/O。

**方法二：占用磁盘 I/O 带宽 + 持续读写**

# 启动大量磁盘 I/O 密集任务（推荐在 HDD 或挂载的 loop 设备）  
for i in {1..5}; do  
 (while true; do dd if=/dev/zero of=io\_test\_$i bs=1M count=1024 oflag=direct; done) &  
done

查看是否已超出io\_time > 20ms：

# 实时查看设备的 I/O 指标  
iostat -x 1  
  
# 或查看原始 io\_time 增量（每秒差值）：  
watch -n 1 "awk '{print \$10}' /sys/block/sda/stat"

**方法三（进阶）：构造 loopback 虚拟设备模拟**

若不希望污染真实硬盘，可以这样：

# 创建一个 1GB 文件并挂载为 loopback 设备  
dd if=/dev/zero of=/tmp/loop.img bs=1M count=1024  
losetup /dev/loop7 /tmp/loop.img  
mkfs.ext4 /dev/loop7  
mkdir /mnt/loopdisk  
mount /dev/loop7 /mnt/loopdisk  
  
# 模拟 I/O 压力  
for i in {1..5}; do  
 (dd if=/dev/zero of=/mnt/loopdisk/file\_$i bs=1M count=1024 oflag=direct &)   
done

查看 /sys/block/loop7/stat，即可验证 I/O 延迟。

## net指标

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 指标名 | 异常阈值 | 含义说明 | 是否可模拟 | 说明 |
| err\_in | >0.5% | 接收错误帧占总接收帧的比例 | ✅可模拟 | 注入错误帧或驱动异常 |
| err\_out | >0.5% | 发送错误帧占总发送帧的比例 | ✅可模拟 | 修改发送包、使用错误路由等 |
| receive\_bytes\_rate | >90% 网卡速率 | 单位时间内接收字节数（带宽利用率） | ✅可模拟 | 发送大量流量到本机或远程 |

**前提工具检查（建议安装）：**

安装这些常用工具包（有些银河麒麟版本默认无）：

sudo yum install iproute iputils net-tools ethtool iperf3 tc -y

**测试建议**

|  |  |
| --- | --- |
| 场景 | 建议模拟方式 |
| 单纯测试报警响应 | 使用 tc netem  即可 |
| 流量压测 + 速率观察 | 用 iperf3  、多线程传输模拟带宽耗尽 |
| DevOps 智能运维联调 | 加入 Logstash、LangGraph 日志链路 |

#### 指标采集方式（验证用）

# 获取网卡名（假设为 eth0）  
ip addr | grep 'state UP' -A2  
  
# 查看网卡错误统计  
ifconfig eth0 | grep "errors"  
  
# 查看接收速率（每10秒抓一次）  
cat /proc/net/dev | grep eth0  
# 或  
watch -n 10 "cat /proc/net/dev | grep eth0"

### 快速一键脚本（模拟网络异常）

保存为simulate\_net\_abnormal.sh：

#!/bin/bash  
IFACE="eth0" # 请替换为你的网卡名，如 ens33, eth1 等  
  
echo "[\*] 模拟网络错误率（接收+发送）..."  
sudo tc qdisc add dev $IFACE root netem corrupt 10%  
sleep 5  
sudo tc qdisc change dev $IFACE root netem loss 20%  
  
echo "[\*] 启动接收大流量测试（需要另一个终端运行：iperf3 -s）"  
iperf3 -c 127.0.0.1 -t 30 -P 5  
  
echo "[\*] 请使用 ifconfig 或 cat /proc/net/dev 观察 err\_in, err\_out, bytes\_recv"

执行：

chmod +x simulate\_net\_abnormal.sh  
./simulate\_net\_abnormal.sh

**清理恢复命令**

sudo tc qdisc del dev eth0 root

### 模拟方式一览

#### 1.err\_in>0.5%（接收错误率异常）

模拟方式：使用tc netem注入接收错误包或包丢失

# eth0 为目标网卡，请替换为实际名称  
sudo tc qdisc add dev eth0 root netem corrupt 20%

20% 的数据包会被破坏，模拟接收错误。

验证：

ifconfig eth0 | grep errors

#### 2.err\_out>0.5%（发送错误率异常）

模拟方式：使用tc工具模拟发送错误，或设置错误路由

# 模拟发送包丢失或破损  
sudo tc qdisc add dev eth0 root netem loss 30%

表示发送时随机丢弃 30% 数据包，引发错误输出。

验证：

ifconfig eth0 | grep errors

#### 3.receive\_bytes\_rate>90%（接收速率>网卡90%）

模拟方式：用iperf3或wget/nc接收大量数据流，或本机模拟转发。

**方法一：使用 iperf3（推荐）**

1）在远程主机（或本机另一终端）启动 iperf3 服务端：

iperf3 -s

2）本机作为客户端下载大流量数据（多线程，接收速率飙升）：

iperf3 -c 127.0.0.1 -t 30 -P 5 # P=并发连接数，t=持续时间

**方法二：本地用wget下载大文件（模拟满速）**

wget http://speedtest.tele2.net/10GB.zip -O /dev/null

（如无法访问公网，可搭建本地 nginx 提供大文件下载）

**receive\_bytes\_rate 计算方式：**

# 示例：10 秒内字节增量  
start=$(awk '/eth0/ {print $2}' /proc/net/dev)  
sleep 10  
end=$(awk '/eth0/ {print $2}' /proc/net/dev)  
rate=$(( (end - start) / 10 ))  
echo "Receive rate: $rate bytes/sec"

要判断是否>90%，需知道网卡带宽。例如1Gbps网卡最大约为125000000bytes/s，90%是112500000。

## processes指标

### 指标说明与触发条件

|  |  |  |
| --- | --- | --- |
| 指标名 | 异常条件 | 含义 |
| total | >500 | 系统当前运行的总进程数 |
| pct\_zombie | >10% | 僵尸进程占比（= zombie / total） |

**指标查看命令（验证用）**

# 总进程数  
ps -e --no-headers | wc -l  
  
# 僵尸进程数量  
ps -eo stat | grep -c '^Z'  
  
# 僵尸进程占比  
total=$(ps -e --no-headers | wc -l)  
zombie=$(ps -eo stat | grep -c '^Z')  
echo "scale=2; $zombie/$total\*100" | bc

### 快速一键脚本（同时模拟total+zombie异常）

保存为simulate\_process\_abnormal.sh：

#!/bin/bash  
  
echo "[\*] 启动大量休眠进程（模拟 total > 500）..."  
for i in {1..600}; do  
 (sleep 300 &)   
done  
  
echo "[\*] 编译并运行僵尸进程生成器（模拟 pct\_zombie > 10%）..."  
cat << 'EOF' > create\_zombies.c  
#include <unistd.h>  
#include <stdlib.h>  
int main() {  
 for(int i = 0; i < 120; i++) {  
 if (fork() == 0) exit(0);  
 else sleep(1000);  
 }  
 return 0;  
}  
EOF  
  
gcc create\_zombies.c -o create\_zombies  
./create\_zombies &  
  
echo "[\*] 异常模拟完成。建议执行以下命令查看效果："  
echo " ps -e --no-headers | wc -l"  
echo " ps -eo stat | grep -c '^Z'"

执行方式：

chmod +x simulate\_process\_abnormal.sh  
./simulate\_process\_abnormal.sh

**清理恢复命令**

# 清除 sleep 与 create\_zombies 创建的所有子进程  
pkill sleep  
pkill create\_zombies  
rm -f create\_zombies create\_zombies.c

### 异常模拟方式

#### 1.total>500（系统总进程数过多）

模拟方式：创建大量休眠子进程不退出。

# 启动 600 个 sleep 子进程  
for i in {1..600}; do  
 (sleep 300 &)   
done

每个子进程挂起 5 分钟，确保留存时间足够长。

查看验证：

ps -e --no-headers | wc -l

#### 2.pct\_zombie>10%（僵尸进程超过10%）

模拟方式：创建子进程后让其立即退出，而**父进程不回收**（不 wait()），从而变为僵尸进程。

**测试脚本生成100+僵尸进程：**

cat << 'EOF' > create\_zombies.c  
#include <unistd.h>  
#include <stdlib.h>  
#include <stdio.h>  
  
int main() {  
 for(int i = 0; i < 120; i++) {  
 if (fork() == 0) {  
 // 子进程立即退出  
 exit(0);  
 } else {  
 // 父进程不wait，休眠保持不退出  
 sleep(1000);  
 }  
 }  
 return 0;  
}  
EOF  
  
gcc create\_zombies.c -o create\_zombies  
./create\_zombies &

会生成 120 个僵尸进程，+父进程总进程数约150，僵尸比例 >80%。

查看僵尸进程状态：

ps -eo pid,ppid,state,cmd | grep defunct  
ps -eo stat | grep -c '^Z'

**注意事项：**

|  |  |
| --- | --- |
| 项目 | 建议说明 |
| 系统限制 | 某些系统对最大子进程数有限制（如 ulimit） |
| 建议运行用户 | 建议 root 或 sudo 权限运行，防止权限不足 |
| 僵尸进程不占CPU | 但会误报健康问题，应及时清理 |
| 僵尸检测工具补充 | 可用 top, htop, ps, vmstat 等观察 |

## system指标

### 指标解释与异常条件

|  |  |  |
| --- | --- | --- |
| 指标名 | 异常阈值 | 说明 |
| load1 | >6 | 最近 1 分钟平均运行队列长度（和 CPU 核数相关） |
| load5 | >8 | 最近 5 分钟平均运行队列长度（需要持续高负载才能升高） |
| load15 | >8 | 最近 15 分钟平均运行队列长度（长时间拥堵才上升） |

注6：实际阈值是基于 CPU 核数 × 1.5（load1）或 × 2（load5/15）

若你是 4 核 CPU：load1=6，load5=8，load15=8 就属于高负载

**当前负载与核心数查看方式**

# 查看 CPU 核心数  
nproc  
  
# 查看当前负载（load1 / load5 / load15）  
uptime  
# 或：  
cat /proc/loadavg

### 快速一键脚本（制造系统 load1/load5/load15 全部异常）

保存为 simulate\_system\_load.sh：

#!/bin/bash  
echo "[\*] 正在制造系统高负载..."  
  
for i in {1..16}; do  
 (while :; do openssl speed aes256 >/dev/null; done) &  
done  
  
for i in {1..8}; do  
 (yes > /dev/null &)   
done  
  
echo "[\*] 模拟启动成功，建议运行以下命令查看负载："  
echo " uptime"  
echo " cat /proc/loadavg"

执行方式：

chmod +x simulate\_system\_load.sh  
./simulate\_system\_load.sh

**清理恢复命令**

# 杀死所有模拟任务  
pkill -f openssl  
pkill -f yes  
pkill -f gzip  
rm -f bigfile bigfile\_\*

### 模拟系统负载异常的方法（实测有效）

核心思路：使用并发密集计算任务（如压缩、openssl、无限循环），制造持续排队负载。

#### 方式一：openssl 加密密集计算（快速制造 load1 > 6）

# 启动并发任务（每个占用一个 CPU 核心）  
for i in {1..12}; do  
 (while :; do openssl speed aes256 >/dev/null; done) &  
done

推荐并发数量为：CPU核心数 × 2，制造超额任务

**观察实时负载**

uptime  
# 或  
cat /proc/loadavg

#### 方式二：死循环 + 多线程（load1、load5、load15 持续上升）

# 持续 CPU 饱和压力  
for i in {1..16}; do  
 (yes > /dev/null &)   
done

yes>/dev/null是经典CPU压力生成方法

#### 方式三：压缩大文件（瞬时 + 持续 I/O + CPU）

# 创建大文件  
dd if=/dev/zero of=bigfile bs=100M count=50  
  
# 并发压缩多个大文件（多线程、I/O密集+CPU密集）  
for i in {1..6}; do  
 cp bigfile bigfile\_$i  
 gzip bigfile\_$i &  
done

**注意事项**

|  |  |
| --- | --- |
| 项目 | 建议说明 |
| 是否安全 | ✅ 已验证命令在银河麒麟中可正常运行，无破坏性 |
| 执行权限 | 建议以 root 或 sudo 权限运行以防部分命令失败 |
| 是否持续影响 | ❗ 如果不 kill 进程，会持续制造负载 |
| 对观察时间的要求 | load5/load15 要等待几分钟才能看到明显变化 |