# Cgroup在Ceph中的应用

## 一、Cgroup简介

Cgroups是control groups的缩写，是Linux内核提供的一种可以限制、记录、隔离进程/进程组所使用的物理资源（如：CPU, Mem- ory, IO等）的机制。最初由Google的工程师提出，后来被整合进Linux内核。

Cgroups最初的目标是为资源管理提供的一个统一的框架，既整合现有的Cpuset等子系统，也为未来开发新的子系统提供接口。现在的 Cgroups适用于多种应用场景，从单个进程的资源控制，到实现操作系统层次的虚拟化（OS Level Virtualization）。Cgroups提 供以下功能：

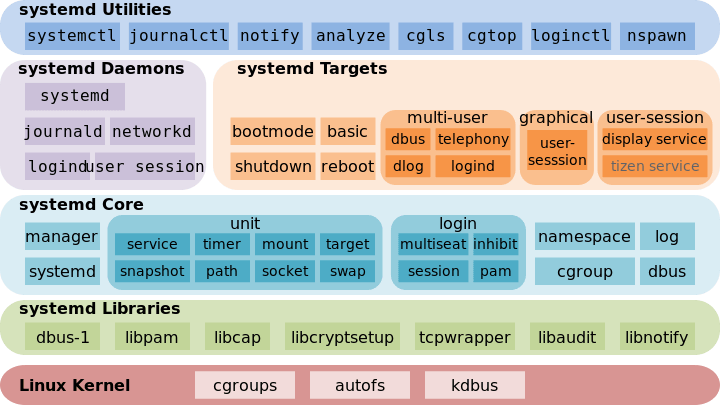
1. 限制进程组可以使用的资源数量（Resource limiting ）。比如：Memory子系统可以为进程组设定一个Memory使用上限，一旦 进程组使用的内存达到限额再申请内存，就会触发OOM Killer。
2. 进程组的优先级控制（Prioritization）。比如：可以使用CPU子系统为某个进程组分配特定CPUshare
3. 进程组隔离（Isolation）。比如：使用ns子系统可以使不同的进程组使用不同的namespace，以达到隔离的目的，不同的进程组 有各自的进程、网络、文件系统挂载空间
4. 记录进程组使用的资源数量（Accounting）。比如：可以使用Cpuacct子系统记录某个进程组使用的CPU时间
5. 进程组控制（Control）。比如：使用freezer子系统可以将进程组挂起和恢复

### cgroup子系统介绍

1. blkio: 这个子系统为块设备设定输入/输出限制，比如物理设备（磁盘，固态硬盘，USB等等）
2. cpu: 这个子系统使用调度程序提供对CPU的Cgroup任务访问
3. cpuacct: 这个子系统自动生成Cgroup中任务所使用的 CPU 报告
4. cpuset: 这个子系统为 Cgroup中的任务分配独立CPU（在多核系统）和内存节点
5. devices: 这个子系统可允许或者拒绝Cgroup中的任务访问设备
6. freezer: 这个子系统挂起或者恢复Cgroup中的任务
7. memory: 这个子系统设定Cgroup中任务使用的内存限制，并自动生成由那些任务使用的内存资源报告
8. net\_cls: 这个子系统使用等级识别符（classid）标记网络数据包，可允许Linux 流量控制程序（tc）识别从具体cgroup中生 成的数据包
9. ns: 名称空间子系统

### cgroup使用示例

在CentOS系统中，安装了libcgroup-tools后，就能使用其提供的工具来进行资源限制，另外，CentOS7中使用了systemd来替代之 前版本的sysvinit启动方式，并在system启动脚本中加入了对cgroup的支持，如下图所示：



(systemd架构图)

由于systemd对cgroup的支持并不完善，所以一般还是推荐安装libcgroup-tools来作为临时解决方案，这里我将在systemd中使用 libcgroup-tools来进行测试，实际效果跟使用cgexec的效果是一样的，测试代码如下:

测试内存限制：

test.cc  
  
#include <stdio.h>  
#include <string>  
#include <unistd.h>  
  
using std::string;  
  
int main()  
{  
 string \*str;  
  
 while (true)  
 {  
 str = new string;  
 str->assign(string(BUFSIZ, 'a'));  
 ::usleep(10);  
 }  
  
 return 0;  
}

测试CPU占用限制:

#include <unistd.h>  
  
int main()  
{  
 while (true)  
 {  
 ::usleep(1);  
 }  
  
 return 0;  
}

执行make test生成可执行文件

#### 1. 使用systemd

systemd启动脚本中加入cgroup资源限制比较简单，目前测试比较好用的是内存限制和CPU使用百分比限制，systemd脚本如下：

[Unit]  
Description=Test Service  
  
[Service]  
ExecStart=/root/test  
MemoryAccounting=true  
MemoryLimit=500M  
CPUAccounting=true  
CPUQuota=1%

[root@vmware ~]# systemctl status test  
● test.service - Test Service  
 Loaded: loaded (/usr/lib/systemd/system/test.service; enabled; vendor preset: disabled)  
 Active: active (running) since Thu 2016-09-22 16:15:13 CST; 2s ago  
 Main PID: 6306 (test)  
 Memory: 326.9M (limit: 500.0M)  
 CGroup: /system.slice/test.service  
 └─6306 /root/test  
  
Sep 22 16:15:13 vmware systemd[1]: Started Test Service.  
Sep 22 16:15:13 vmware systemd[1]: Starting Test Service...

[root@vmware ~]# systemctl status test  
● test.service - Test Service  
 Loaded: loaded (/usr/lib/systemd/system/test.service; enabled; vendor preset: disabled)  
 Active: failed (Result: signal) since Thu 2016-09-22 16:15:36 CST; 954ms ago  
 Process: 6306 ExecStart=/root/test (code=killed, signal=KILL)  
 Main PID: 6306 (code=killed, signal=KILL)  
  
Sep 22 16:15:13 vmware systemd[1]: Started Test Service.  
Sep 22 16:15:13 vmware systemd[1]: Starting Test Service...  
Sep 22 16:15:36 vmware systemd[1]: test.service: main process exited, code=killed, status=9/KILL  
Sep 22 16:15:36 vmware systemd[1]: Unit test.service entered failed state.  
Sep 22 16:15:36 vmware systemd[1]: test.service failed.

另外从dmesg中也能看到该进程被内核kill掉

9963.177067] CPU: 0 PID: 6306 Comm: test Tainted: G OE ------------   
3.10.0-327.el7.x86\_64 #1  
[ 9963.177068] Hardware name: VMware, Inc. VMware Virtual Platform/440BX Desktop Reference Platform,  
 BIOS 6.00 07/02/2015  
[ 9963.177069] ffff8800358d5c00 000000009061b3ff ffff880037f73cd0 ffffffff816351f1  
[ 9963.177071] ffff880037f73d60 ffffffff81630191 0000000000000001 ffff880009dc4000  
[ 9963.177073] ffffffff37f73d30 fffeefff00000000 0000000000000001 ffff8800358d6103  
[ 9963.177074] Call Trace:  
[ 9963.177116] [<ffffffff816351f1>] dump\_stack+0x19/0x1b  
[ 9963.177138] [<ffffffff81630191>] dump\_header+0x8e/0x214  
[ 9963.177160] [<ffffffff8116cdee>] oom\_kill\_process+0x24e/0x3b0  
[ 9963.177165] [<ffffffff81088dae>] ? has\_capability\_noaudit+0x1e/0x30  
[ 9963.177184] [<ffffffff811d3b05>] mem\_cgroup\_oom\_synchronize+0x555/0x580  
[ 9963.177186] [<ffffffff811d2ef0>] ? mem\_cgroup\_charge\_common+0xc0/0xc0  
[ 9963.177188] [<ffffffff8116d664>] pagefault\_out\_of\_memory+0x14/0x90  
[ 9963.177190] [<ffffffff8162e592>] mm\_fault\_error+0x8e/0x180  
[ 9963.177192] [<ffffffff816410b1>] \_\_do\_page\_fault+0x3e1/0x420  
[ 9963.177193] [<ffffffff81641113>] do\_page\_fault+0x23/0x80  
[ 9963.177195] [<ffffffff8163d408>] page\_fault+0x28/0x30  
[ 9963.177197] Task in /system.slice/test.service killed as a result of limit of /system.slice/test.service  
[ 9963.177199] memory: usage 512000kB, limit 512000kB, failcnt 492267  
[ 9963.177200] memory+swap: usage 2526644kB, limit 9007199254740991kB, failcnt 0  
[ 9963.177200] kmem: usage 0kB, limit 9007199254740991kB, failcnt 0  
[ 9963.177201] Memory cgroup stats for /system.slice/test.service: cache:0KB rss:512000KB   
 rss\_huge:0KB mapped\_file:0KB swap:2014644KB inactive\_anon:256056KB a  
 ctive\_anon:255920KB inactive\_file:0KB active\_file:0KB unevictable:0KB  
[ 9963.177208] [ pid ] uid tgid total\_vm rss nr\_ptes swapents oom\_score\_adj name  
[ 9963.177295] [ 6306] 0 6306 634771 127859 1246 503995 0 test  
[ 9963.177295] Memory cgroup out of memory: Kill process 6306 (test) score 941 or sacrifice child  
[ 9963.177324] Killed process 6306 (test) total-vm:2539084kB, anon-rss:510612kB, file-rss:824kB

加入CPU使用率限制之后，CPU的使用率始终维持在给定的值

6731 root 20 0 12488 828 692 R 1.3 0.1 0:00.19 /root/test

#### 2. 使用libcgroup-tools

使用libcgroup-tools的方式主要是使用了cgconfigparser来解析/etc/cgconfig.d目录下的文件并创建配置文件中指定的子系统，

## 二、在ceph中使用cgroup来限制所用资源

为什么要用cgroup限制ceph的使用资源呢，其中一方面是为了防止在融合架构中ceph占用过多的内存和cpu，另一方面也是为了防止其 它进程占用ceph的资源，导致存储出现故障。

比如monitor和osd都是典型的多线程程序，如果有多个线程都需要相同的数据，那么将这些线程绑定到一个特定的CPU上是非常有意义的， 这样就能确保它们可以访问相同的缓存数据，从而至少能提高缓存命中率，否则的话，这些线程可能会在不同的CPU上执行，这样会频繁地 使其他缓存项失效。但是像ceph这样动不动一个进程就有上百个线程的情况，绑定到一个CPU上或许有些不可取，一般的做法是一个线程分 配一个核，所以在这种情况下，可以为所有的ceph进程分配几个核。

还有就是关于ceph的内存占用，对于IO密集型的应用，cache能加速应用性能，经过测试发现ceph在缓存占用这块需要限制下，目前给 的限制是mon内存1G，缓存4G，osd的内存2G，缓存8G。虽然这样可能会导致ceph的性能降低，但是能保证与其它组件一起稳定运行，

另外典型的故障就是由于磁盘IO过高导致ceph monitor挂掉，一般情况下非IO密集型的应用占用磁盘IO并不是很高，但ceph的mon是 使用的系统盘，在与openstack的其它组件部署在一起的时候，磁盘IO过高就会导致monitor出现故障。

当然，一般只需要限制cpu，cpuset和memory这些子系统就足以了。我这里所用的方式是对于ceph的每个osd进程或者mon进程都创建有 对应的一份cgroup配置文件，这个配置文件是在部署ceph的时候。在这个配置文件中，我们就可以通过一些参数配置来限制进程的资源利 用情况。