Managing CPU Resources with RHEL6 cgroups (A Demo)

Version 31

created by Bob Kozdemba on Oct 18, 2010 4:46 PM, last modified by Bob Kozdemba on Jan 18, 2013 2:52 PM

Visibility: Open to anyone

I've put together a quick demonstration to show how cgroups can be used to manage resources on an cpu intensive application (included). To show the demo, you will need a machine running RHEL6 with at least 2 cores (disable hyper-threading in the BIOS) and the rpms which are located in the attachments section of this document.

Configuration

Choose either the persistent or temporary configuration

Persistent Configuration

1a) Add 2 new control groups to `/etc/cgconfig.conf and replace` **`bkozdemb`** `with your own uid and gid.`

```
group group1 {
    perm {
        task {
            uid = bkozdemb;
            gid = bkozdemb;
        }
        admin {
            uid = bkozdemb;
            gid = bkozdemb;
              }
           }
cpuset {
              cpuset.cpus = "0,1";
              cpuset.mems = 0;
       }
       cpu   {
              cpu.shares = 2048;
       }
       freezer {
              freezer.state=THAWED;
       }
}
```

```
group group2 {
   perm {
      task {
         uid = bkozdemb;
         gid = bkozdemb;
      }
      admin {
         uid = bkozdemb;
         gid = bkozdemb;
          }
        }
cpuset {
           cpuset.cpus = "0,1";
           cpuset.mems = 0;
          }
     cpu  {
           cpu.shares = 2048;
          }
      freezer {
           freezer.state=THAWED;
        }
}
```

1b) Restart and enable the cgconfig service

```
# service cgconfig restart; chkconfig cgconfig on
```

Temporary Configuration

2a) Restart and enable the cgconfig service
2b) Create the control groups using the libcgroup command line tools.

```
01. #!/bin/bash
02. #
03. # Shell script to create cgroups for demos.
04. #
05. # Exec as root
06. #
07. # Substitute your uid/gid
08. #
09. uid=bkozdemb
10. gid=bkozdemb
11.
12. service cgconfig restart
13.
14. for group in `seq 1 2`
15.  do
```

```
16.
17.    cgcreate -t $uid:$gid -a $uid:$gid \
18.    -g cpu:group$group -g cpuset:group$group -g freezer:group$group
19.    cgset -r cpu.shares=2048 group$group
20.    cgset -r cpuset.cpus="0-1" group$group
21.    cgset -r cpuset.mems=0 group$group
22.
23.  done
```

3) Verify your cgroup was created

```
# lscgroup
```

4) Import my gpg public key.

rpm --import RPM-GPG-KEY-koz

5) Install the cgdemos rpm. To satisfy yum dependencies, make sure your system is subscribed to the RHEL Server Optional channel on RHN. (These programs should get installed into /usr/local/bin and /usr/local/share)

```
# yum localinstall cgdemos-1.0-1.x86-64.rpm
```

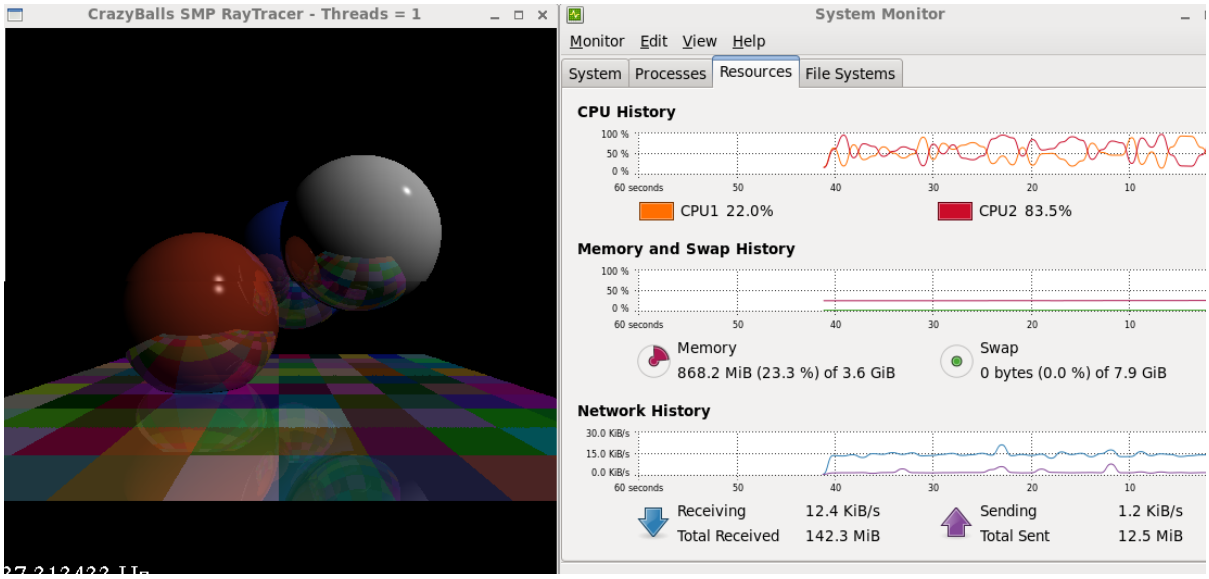
Overview

The tarball includes a resource hungry, multi-threaded, real-time ray tracing demo application called **`smpray`**. I wrote it a few years ago when I worked at SGI and decided to resurrect it when I heard about cgroups. To show how cpu resources can be managed, I like to show the behavior of **smpray** before and after resource control. Although very few of our customers run real-time ray tracers, this program consumes cpu resources in the same way that a heavily loaded virtual machine or database engine would do so. The cgbuddy is a simple program that makes libcgroup API calls in order to modify the cgroup name/value pairs in cgroup virtual file system. Instead of using cgbuddy to change the cgroup values, have a look at the **`cgset(1)`** command.

Running the Demo

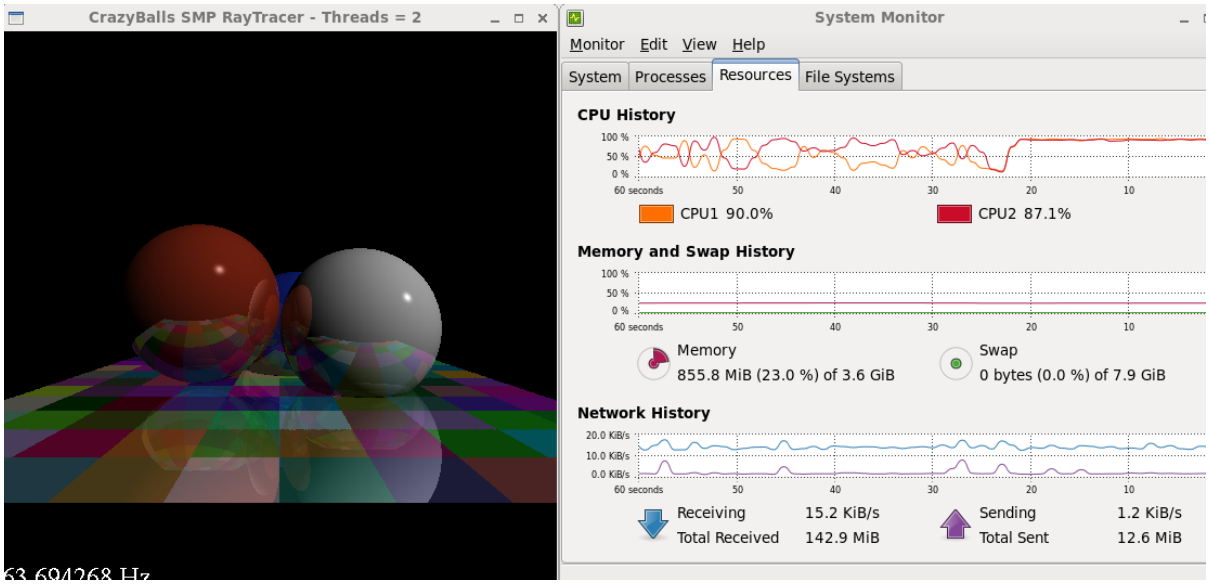
1) First, run the gnome-system-monitor in the background. Now run smpray with 1 thread and you should see 3 shiny reflective spheres dancing about. Take note of the reported frame rate and observe the CPU history graph. Notice that the application is migrating back and forth between CPUs #1 and #2. This is the default behavior of the RHEL6 Complete Fair Scheduler (CFS).

```
$ gnome-system-monitor &
$ smpray -t1
```



2) Kill the single threaded version and run smpray with 2 threads. The frame rate should almost double and both CPUs #1 and #2 should be quite busy. This simple program, run by a regular user, has just consumed nearly all of the available CPU resources as evident by the CPU history graph.

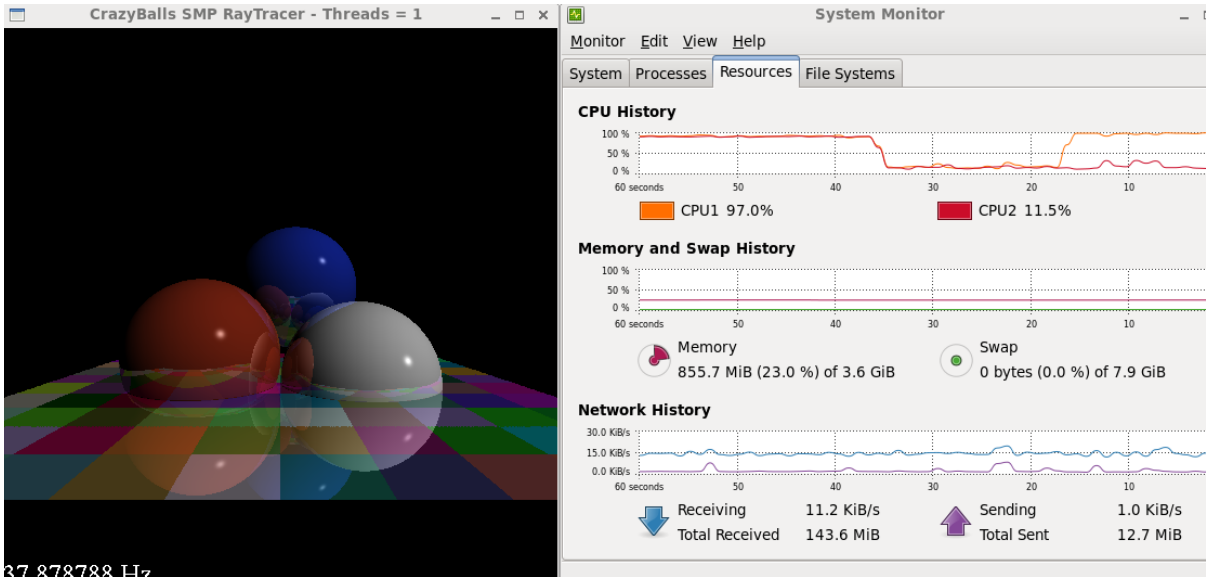
```
$ smpray -t2
```



3) Kill the 2 threaded version and run cgbuddy in the background. Now run smpray with 1 thread under resource control and use cgbuddy to change **cpuset.cpus=0** (remember to press 'Enter'). The frame rate should be similar to step 1, however, the CPU history graph shows that this application is pinned to CPU #1 because of resource control. If you don't want to use cgbuddy, run **'cgset -r**

`cpuset.cpus=0 /group1`' to change the parameter.

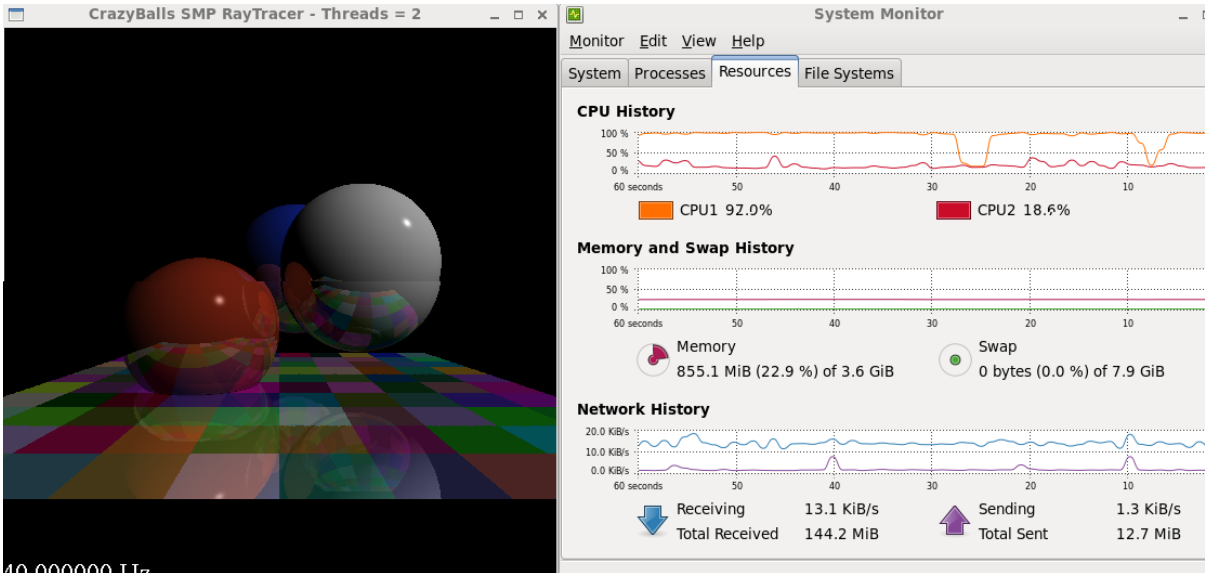
```
$ cgbuddy &
$ cgexec -g cpuset,cpu,freezer:/group1 smpray -t1
```



4) After killing the single threaded version, run smpray with 2 threads under resource control. Notice that because both threads are pinned to CPU #1, the frame rate does not speed up as it did in step 2.

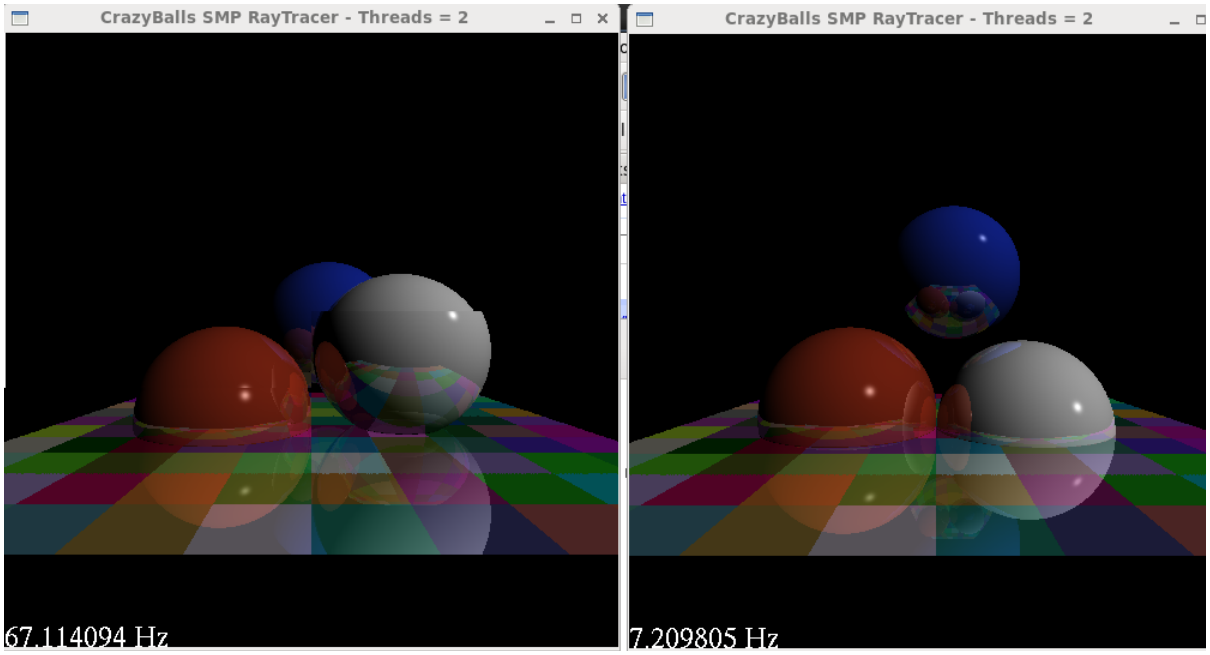
$ cgexec -g cpuset, cpu, freezer:/group1 smpray -t2

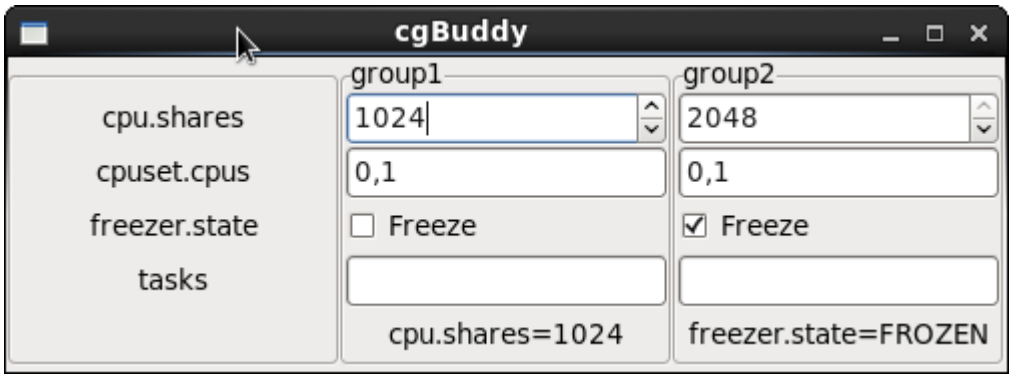
For the final demo, use `cgbuddy` or `cgset` to return **cpuset.cpus=0,1 for /group1**



5) To demonstrate how cgroups can limit an application to a fraction of a CPU when resources become scarce, run a 2 threaded version in each control group. Now use the cgbuddy program to change **cpu.shares=8** in one of the control groups. Notice how the program with 8 shares runs substantially slower. If you experiment with the number of shares in each group, you will notice that cpu resources are based on the ratios of the shares. Also, observe what happens when the freezer control group is checked.

```
$ cgexec -g cpu, cpusets, freezer:/group1 smpray -t2 &
$ cgexec -g cpu, cpusets, freezer:/group2 smpray -t2 &
```





*Remember to press 'Enter' after changing any text entry fields.

Final Comments

To facilate a smoother flowing demonstration, the cgdemos package creates a directory (/usr/local /share/cgdemos) with gnome a launcher for each step above so you can leave your hands off the keyboard. The **`smpray`** application is written in 'C' and uses the pthreads library for multiprocessing and OpenGL/freeglut to support 2D graphics rendering. All of the 3D lighting and rendering calculations are performed by the ray tracer algorithms in software. The cgbuddy program makes use of the libcgroup API and the gtk toolkit.

Please send questions and feedback to bkozdemb@redhat.com

References

http://docs.redhat.com/docs/en-US/Red_Hat_Enterprise_Linux/6/html

/Resource_Management_Guide/index.html
http://fedoraproject.org/wiki/Features/ControlGroups
http://www.redhat.com/summit/2011/presentations/summit/in_the_weeds/friday
/WangKozdemba_f_1130_cgroups14.pdf

RPM-GPG-KEY-koz.zip
1.4 K

cgbuddy-1.0-1.i686.rpm
8.5 K

smpray-1.0-2.i686.rpm
9.3 K

cgbuddy-1.0-1.s390x.rpm
9.6 K

smpray-1.0-2.s390x.rpm
10.3 K

cgdemos-1.0-1.x86_64.rpm
16.8 K

1365 Views Categories: Tags: cgroups

Average User Rating

(1 rating)

My Rating:

4 Comments 0 Author comments

Anthony Green Oct 18, 2010 6:49 PM (in response to)

This is great Bob. It's just what I was looking for. Thanks!

AG

Actions Like (0)

Robin Price Oct 19, 2010 12:50 PM (in response to Anthony Green)

Same here! This will be great for our partners!

Actions Like (0)

Alexander Pierce Oct 21, 2010 6:04 PM (in response to Anthony Green)

This is excellent! I have had much interest in this from RHEL 6 roadmaps, and the demonstration really drives the point home.

Actions Like (0)

Frank Weyns Dec 1, 2010 5:49 AM (in response to Anthony Green)

Nice stuff... looks much better then my "time echo "scale=50000; 4*a(1)" | bc -l -q" or the simple version "cat /dev/zero > /dev/null" ;-)
But do you have the code of "smpray" ... it does not work on a 32bit RHEL 6 desktop, Thanks.

Actions Like (0)