Manage Performance with Cgroups and Projects

David Collier-Brown
Agenda

• Introduction
• Resource Managers for Linux and Unix
• Containers, cgroups and the Completely Fair Scheduler
  > CPU management
  > Memory management
  > I/O
  > Network
• Cgroups in Practice
• Conclusions
Some Things Won't Fit

• For example, a fridge in a French minicar

• Or an ad-hoc reporting program in a small, fast website
Imagine for a Moment

• You're the sysadmin for a successful LAMP-based web site
• Management has just gifted you with a new ad-hoc reporting program.
• You now have to somehow shoehorn it in
  > Without affecting the performance

• Insoluble problem?
• Not a bit of it!
Limit Reporting to Spare Cycles

• Production is more important, so reporting should take a second place
• It should only get CPU, I/O or network bandwidth when production doesn't need it
Add Resource Management

• What's needed is a resource manager like cgroups
• Assign most of the CPU, memory or I/O bandwidth to the more important programs
• And then give any other programs, including reporting, a fair share of what's left.
A Classic Capacity problem

• This is the performance *management* problem.

• Make sure non-critical programs don't interfere with important ones,

• And ensure there is a bit of CPU for root to use to stop and diagnose runaway programs.

• You can't create resources out of nothing

• You can do a lot more than you'd think
My Laptop is Too Slow!

• As a practical example, I have a CPU- and disk-hog Solaris data-analysis batch program.

• My laptop is just a bit too slow for it

• Whenever I run it, even with nice -20, it takes 100% of the CPU and a big chunk of the disk bandwidth.

• That's so much that I can't keep up with my everyday work.
  > I can't even read email when it's hogging the machine.
So I Ration It

• To solve this, I give
  > one share of the CPU to the background program,
  > three shares to everything else.
• Now I can run the job and the only indication is the Gnome perfmeter is pegged at 100%:
• my interactive programs run at full speed.
What's Happening

• The background program is cut to 25% whenever other programs need CPU,
• But is allowed to use all the CPU it wants when I'm thinking and neither typing nor moving the mouse.
• This keeps it from issuing enough I/Os to make my disk non-responsive.
  > That's a lucky side-effect
• It's enforcing a least upper bound
My process status looked like this:

<table>
<thead>
<tr>
<th>PROJID</th>
<th>NPROC</th>
<th>SIZE</th>
<th>RSS</th>
<th>MEMORY</th>
<th>TIME</th>
<th>CPU</th>
<th>PROJECT</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>2</td>
<td>2304K</td>
<td>1232K</td>
<td>0.2%</td>
<td>00:02:26</td>
<td>92.0%</td>
<td>bg</td>
</tr>
<tr>
<td>101</td>
<td>32</td>
<td>919M</td>
<td>277M</td>
<td>56%</td>
<td>00:02:54</td>
<td>6.5%</td>
<td>user.davecb</td>
</tr>
<tr>
<td>0</td>
<td>39</td>
<td>164M</td>
<td>62M</td>
<td>13%</td>
<td>00:00:28</td>
<td>0.1%</td>
<td>system</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>14M</td>
<td>3264K</td>
<td>1%</td>
<td>00:00:00</td>
<td>0.00%</td>
<td>default</td>
</tr>
</tbody>
</table>

- The project containing the background job averaged 92% of the CPU, and I averaged only 6.5% over 10 seconds,
- But whenever user davecb ran anything, *that* program got at least 25% of the CPU.
- That was for more than I needed for Open Office.
Interactive editing was fast...

• Well, as fast as OO ever is.
• I literally needed to keep checking the perfmeter to see when the batch job was done.
Resource Managers for Linux and Unix

• Back in the days of the IBM mainframes, resource management was critical: hundreds of users need to be able to get their fair share a single machine.

• Nor could they afford to let a program with a memory leak steal all the memory from everyone else.
Minis and Micros

• For many years, mini and microcomputers had so little performance that you only ran one program per machine, and didn't have to worry about sharing.

• Now, however, the average Linux machine
  > has the power of an older mainframe and
  > will be running seventy-odd processes by the time the first user has logged on.
Resource Managers Redux

• Because of this, resource managers are coming back, starting with the commercial Unixes like Solaris and AIX.

• The 2.6 kernel has a fair-share scheduler and "cgroups", the performance management infrastructure for Linux containers.

• Linux is now a hotbed of resource management research
Control Groups and the CF Scheduler

• Cgroups are “control groups”, an elegant extension to ordinary groups to allow resource management.
  > If a program is in a particular control group, it will given a specific share of the resources of the machine

• For example, consider an overloaded disk
  > all the programs in a group with a guarantee of 10% of the bandwidth of a particular disk will share that 10 percent whenever the disk is busy.
  > When no-one else is using the disk, the members of that same group could take
Control Groups

- Control groups are the low-level mechanism in Linux to provide resource guarantees to containers and virtual machines.
- They are artifacts of the “Completely Fair Scheduler”, CFS.
- Like many Unix constructs, they are organized in a virtual filesystem under /cgroups (or /containers, this is still being debated)
A Container Example

• Assuming you're creating containers, you write and read files to define them

• For example:
  
  # mount -t container -o cpu /containers/cpu
  # mount -t container -o net /containers/net
  # mkdir /containers/new_container
  # echo $$ > /containers/new_container/tasks
And They're Recursive

• They started out as a way to manage resources for
  > containers (lightweight virtual machines) or
  > conventional VMs,
• They allow you to give VMs shares of the real physical machines.
• They aren't restricted to virtual machines though:
  > they can manage resources and therefore the performance of ordinary processes, too.
  > Either inside VMs or under the native OS.
CPU Management

- CPU Management is pretty easy
  > done by scheduling the CPU
  > Hierarchical or recursive

- Strengths & weaknesses
  > Very simple and low-cost
  > Only affects other resources by accident

- My Solaris example “works by accident”
  > It prevents the programs from issuing enough reads
  > Nice -40 would have worked, if it existed
  > But this trick works for diets, so it's not
Memory Management

• Open Office could use this feature
  > It keeps pushing everything else out of memory
  > It really doesn't need to

• Current Linux experiments:
  > keep RSS at a specified level
  > weakly tracks shared libraries, first cgroup to use one gets billed for it
  > Don't use least upper bound

• Future experiments may do it differently
  > “soft limits” are least upper bounds
I/O Management

• Done via the Linux I/O scheduler
  > Solaris lacks this (and I really want it)
  > Doesn't dispatch I/Os if it will exceed the cgroup's ration of bandwidth
  > Equivalent to adding think time (Z)
  > Can set a bound on total bandwidth
    – We'll see this in a moment...

• Uses a virtual filesystem too
  # /bin/echo <device>:<bandwidth> \n     <cgroup>/blockio.bandwidth
I/O Management II [Updated]

• Solaris lacks a scheduler
• But ZFS has one now
  > Implemented the same way as the Linux I/O bandwidth management, by adding sleeps before the write attempts
A Extended Example

• Mount the cgroup filesystem (blockio subsystem):
  # mkdir /mnt/cgroup
  # mount -t cgroup -obo blockio blockio
  /mnt/cgroup
  – Note that this developer put them under a different directory

• Instantiate the new cgroup "foo":
  # mkdir /mnt/cgroup/foo

• Add the current shell process to the cgroup "foo":
  # /bin/echo $$ > /mnt/cgroup/foo/tasks
I/O Example II

• +Give maximum 1MB/s of I/O bandwidth on /dev/sda1 to the cgroup "foo":
  # /bin/echo /dev/sda1:1M \
  >/mnt/cgroup/foo/blockio.bandwidth

• And 8MB/s of I/O bandwidth on /dev/sdb f
  # /bin/echo /dev/sda5:8M \
  >/mnt/cgroup/foo/blockio.bandwidth

• Start a subshell in cgroup "foo" for the test
  > it can use a maximum I/O bandwidth of 1MB/s on /dev/sda1 and 8 MB/s on sda5
I/O Example III

• Now run a benchmark doing I/O on sda1 and 5

• I/O limits and usage are readable from the VFS:
  
  ```
  # cat /mnt/cgroup/foo/blockio.bandwidth
  === device (8,1) ===
    bandwidth limit: 1024 KiB/sec
    current i/o usage: 819 KiB/sec
  === device (8,5) ===
    bandwidth limit: 1024 KiB/sec
    current i/o usage: 3102 KiB/sec
  ```

• Default units are KB/S
Why I/O Limiting Works

Read Response Time vs Effective IOPS Per Drive
6140 10k RPM FC RAID 5

- This is the measured performance of a disk array

Note the degradation after 400 start-I/Os per second.
Why I/O Limiting Works II

- At 400 start-IoFs/S
  > 40 milliseconds
- At 500
  > 80 milliseconds
- This disk is in pain

- So insert 10 milliseconds delay
  > Same as reducing demand to 400 IOPS
  > Service time falls back to 40 milliseconds
  > 30 milliseconds better than without the delay
This Applies to Most Limits

• Anything that can build up a queue
• That's why Apache has a tunable to reject more than N connections
  > N at 40 milliseconds is better than n+25% at 80 Milliseconds
• CPU, Disk and I/O apply
• Memory and caches don't
  > until they're full and degrade to the speed of the paging disk or the bus
Networks Too

• Make a cgroup of file transfer processes and assign it a unique classid # of 0x1234
  > mkdir -p /dev/cgroup
  > mount -t cgroup tc -otc /dev/cgroup
  > mkdir /dev/cgroup/file_transfer
  > echo 0x1234 > /dev/cgroup/file_transfer/tc.classid
  > echo $$ > /dev/cgroup/file_transfer/tasks
Networks Too, II

• Now create a HTB class that rate limits traffic to 100mbits and filter to direct all traffic from cgroup file_transfer to this new class.

  # tc qdisc add dev eth0 root handle 1: htb
  # tc class add dev eth0 parent 1: classid 1:2 htb rate 100mbit ceil 100mbit
  # tc filter add dev eth0 parent 1: protocol ip prio 1 handle 800 cgroup value 0x1234 classid 1:2

• Yet another variant on the syntax...
Networks Too [Updated]

• Solaris added “Virtual NIC Cards”
  > December 4th, 2008
  > Can be assigned to any zone/VM
• They can limit & prioritize “flows”
  > Where a flow is a group of port and IP addresses
• You could have a priority flow for all https
  > And a low-priority one for http & ftp
Cooperative Networking with Trickle

- Already available for OpenBSD, Linux i386, Solaris SPARC, NetBSD/Alpha and FreeBSD
- Implemented as a shared library
  > Doesn't require OS support
  > Can't impose itself, though
  > The BOFH can, though
- Has a management daemon
  > You can trickle all the machine in your home net
- http://monkey.org/~marius/pages/?page=trickle
This is an Immature Area

• Measurement tools are missing
  > No pmap -x
  > No by-cgroup stats

• To find libraries, for example,
  > # strace -p <pid> | grep 'open.*\.so'

• But give them about a week (;-))
Cgroups in Practice

• Already in Fedora
• There, you can:
  > Set up CPU groups for main apps
  > Create a new one for reporting
• See what it bottlenecks -
  > it may be I/O
  > set limits
• Stress-test the new program
  > with normal production unaffected
• This is also handy for developers
Conclusions

• Cgroups are here now, in Fedora
  > There are upsides and downsides
  > There's LOTS of room to improve
Downsides

• Least upper bounds are weird
  > They're not well-studied, so they can be contra-intuitive

• It's hard to measure per-cgroup CPU
  > This week, anyway

• Users WILL get it wrong
  > The Bob Sneed story
Upsides

• Containers and application to VMs coming

• IBM is doing magic with “goal mode”
  > A daemon manages whole mainframe VMs

• You can guarantee
  > production gets 3 second response time minimums
  > Reporting gets 40 seconds minimum

• If the production VM falls below it's goal,
  > it gets more resources at the expense of
**Do Try This At Home**

- Try Trickle almost anywhere
  > Especially for those &###%! downloads
- Or if you're running Fedora
  > At the very least, it will make your laptop or desktop survive CPU hogs
- Recommend it to any production folks running Solaris