Subject: [RFC][PATCH 0/6] Add group fairness to CFS - v1
Posted by Srivatsa Vaddagiri on Mon, 11 Jun 2007 15:47:24 GMT
View Forum Message <> Reply to Message

Ingo,
Here's an update of the group fairness patch I have been working on. Its against CFS v16 (sched-cfs-v2.6.22-rc4-mm2-v16.patch).

The core idea is to reuse much of CFS logic to apply fairness at higher hierarchical levels (user, container etc). In this regard CFS engine has been modified to deal with generic 'schedulable entities'. The patches introduce two essential structures in CFS core:

- struct sched_entity
- represents a schedulable entity in a hierarchy. Task is the lowest element in this hierarchy. Its ancestors could be user, container etc. This structure stores essential attributes/execution-history (wait_runtime etc) which is required by CFS engine to provide fairness between 'struct sched_entities' at the same hierarchy.

- struct lrq
- represents (per-cpu) runqueue in which ready-to-run 'struct sched_entities' are queued. The fair clock calculation is split to be per 'struct lrq'.

Here's a brief description of the patches to follow:

Patches 1-3 introduce the essential changes in CFS core to support this concept. They rework existing code w/o any (intended!) change in functionality.

Patch 4 fixes some bad interaction between SCHED_RT and SCHED_NORMAL tasks in current CFS.

Patch 5 introduces basic changes in CFS core to support group fairness.

Patch 6 hooks up scheduler with container patches in mm (as an interface for task-grouping functionality).

Changes since last version:

- Prelimnary SMP support included (based on the idea outlined at http://lkml.org/lkml/2007/5/25/146)
- Task grouping to which fairness is applied is based on Paul Menage's container patches included in -mm tree. Usage of this feature is described in Patch 6/6
- Fix some real time and SCHED_NORMAL interactions (maintain

separate nr_running/raw_weighted counters for SCHED_NORMAL
tasks)
- Support arbitrary levels of hierarchy. Previous version
supported only 2 levels. Current version makes no assumption
on the number of levels supported.

TODO:

- Weighted fair-share support
Currently each group gets "equal" share. Support
weighted fair-share so that some groups deemed important
get more than this "equal" share. I believe it is
possible to use load_weight to achieve this goal
(similar to how niced tasks use it to get differential
bandwidth)

- Separate out tunables
Right now tunable are same for all layers of scheduling. I
strongly think we will need to separate them, esp
sysctl_sched_runtime_limit.

- Flattening hierarchy
This may be useful if we want to avoid cost of deep
hierarchical scheduling in core scheduler, but at the
same time want deeper hierarchical levels to be supported from
user pov. William Lee Irwin has suggested basic technique at
http://lkml.org/lkml/2007/5/26/81 which I need to
experiment with. With this technique, for ex, it is
possible to have core scheduler support two levels
(container, task) but use weight adjustement
to support more levels from user pov (container, user,
process, task).

- (SMP optimization) during load balance, pick cache-cold tasks
first to migrate

- (optimization) reduce frequency of timer tick processing at
higher levels (similar to how load balancing frequency varies
across scheduling domains).

The patches have been very stable in my tests. There is however one oops I hit
just before sending this (!). I think I know the reason for that (some
cleanup required in RT<->NORMAL switch) and am currently investigating
that.

I am sending the patches largely to get feedback on the direction this
is heading.

Some results of the patches below.

Legends used in the results :-

cfs = base cfs performance (sched-cfs-v2.6.22-rc4-mm2-v16.patch)
cfscc = base cfs + patches 1-3 applied (core changes to cfs core)
cfsccrt = base cfs + patches 1-4 applied (fix RT/NORMAL interactions)
cfsgrpch = base cfs + patches 1-5 applied (group changes applied)
cfsgrpchdi = base cfs + all patches applied (CONFIG_FAIR_GROUP_SCHED disabled)
cfsgrpchen = base cfs + all patches applied (CONFIG_FAIR_GROUP_SCHED enabled)

1. lat_ctx (from lmbench):
==========

Context switching - times in microseconds - smaller is better

---

| Host | OS | 2p/0K ctxsw | 2p/16K ctxsw | 2p/64K ctxsw | 8p/16K ctxsw | 8p/64K ctxsw | 16p/16K ctxsw | 16p/64K ctxsw |
|---|---|---|---|---|---|---|---|---|
| cfs | Linux 2.6.22- | 6.2060 | 7.1200 | 7.7746 | 7.6880 | 11.27 | 8.61400 | 20.68 |
| cfscc | Linux 2.6.22- | 6.3920 | 6.9800 | 7.9320 | 8.5420 | 12.1 | 9.64000 | 20.46 |
| cfsccrt | Linux 2.6.22- | 6.5280 | 7.1600 | 7.7640 | 7.9340 | 11.35 | 9.34000 | 20.34 |
| cfsgrpch | Linux 2.6.22- | 6.9400 | 7.3080 | 8.0620 | 8.5660 | 12.24 | 9.29200 | 21.04 |
| cfsgrpchdi | Linux 2.6.22- | 6.7966 | 7.4033 | 8.1833 | 8.8166 | 11.76 | 9.53667 | 20.33 |
| cfsgrpchen | Linux 2.6.22- | 7.3366 | 7.7666 | 7.9 | 8.8766 | 12.06 | 9.31337 | 21.03 |

Performance of CFS with all patches applied (but with CONFIG_FAIR_GROUP_SCHED disabled) [cfsgrpchdi above] seems to be very close to base cfs performance [cfs above] (delta within tolerable noise level limits?)

2. hackbench
==========

hackbench -pipe 10:

cfs 0.787
cfscc 0.7547
cfsccrt 0.9014
cfsgrpch 0.8691
cfsgrpchdi 0.7864
cfsgrpchen 0.9229

hackbench -pipe 100:

cfs 3.726
cfscc 3.7216
cfsccrt 3.8151
cfsgrpch 3.6107
cfsgrpchdi 3.8468
cfsgrpchen 4.2332

3. Fairness result between users 'vatsa' and 'guest':

The two groups were created as below in container filesystem:

```
# mkdir /dev/cpuctl
# mount -t container -ocpuctl none /dev/cpuctl
# cd /dev/cpuctl
# mkdir vatsa
# mkdir guest

# echo vatsa_shell_pid > vatsa/tasks
# echo guest_shell_pid > guest/tasks

# # Start tests now in the two user's shells
```

hackbench -pipe 10:

vatsa : 1.0186
guest : 1.0449

hackbench -pipe 100:

vatsa : 6.9512
guest : 7.5668

Note: I have noticed that running lat_ctx in a loop for 10 times doesnt give me good results. Basically I expected the loop to take same time for both users (when run simultaneously), whereas it was taking different times for different users. I think this can be solved by increasing sysctl_sched_runtime_limit at group level (to remeber execution history over a longer period).

--
Regards,
vatsa

_____
Containers mailing list
Containers@lists.linux-foundation.org