Subject: [PATCH 0/2] Add group awareness to CFS - v2
Posted by Srivatsa Vaddagiri on Sat, 23 Jun 2007 13:15:45 GMT
View Forum Message <> Reply to Message

Hi Ingo,
 Here's an update for the group-aware CFS scheduler that I have been
working on.

(For those reading these patches for the first time:)

The basic idea is to reuse CFS core and other pieces of scheduler like
smpnice-driven load balance for driving fairness between 'schedulable entities'
other than tasks, for ex: users or containers.

The time-sorted rb-tree and nanosecond accurate accounting aspects of
CFS are "repeated" for schedulable entities other than tasks.

For ex: there could be N task-level rb-trees for N users (which stores
tasks) and a single user-level rb-tree which stores user-level entities.
CFS operations on each user's task-level rb-tree drives fairness between tasks
of that user, while CFS operations on user-level rb-tree drives
fairness between users.

v17 CFS introduced basic changes in CFS to support group scheduling.
The two patches to follow build upon them as follows:

Patch 1 => introduces a notion of scheduler hierarchy (of entities) and
    applies CFS operations at all levels of this hierarchy.

Patch 2 => hooks up the cpu scheduler with task grouping feature in mm tree
    (CONFIG_CONTAINERS) as an interface to task-grouping functionality.

A single config option CONFIG_FAIR_GROUP_SCHED allows the group-scheduling
feature to be turned on/off at compile time.

I have tried my best to ensure there is no impact to existing CFS performance
when CONFIG_FAIR_GROUP_SCHED is disabled. Some results in this regard are
provided at the end.

One noticeable change in functionality may be the /proc/sched_debug output
(I had to rearrange that code a bit to dump group cfs_rq information also).

Changes since last version:

 - Fixed some bugs in SMP load balance (pointed by Dmitry)
 - Modified sched_debug.c to dump all cfs_rq stats

Todo:

- Weighted fair-share
  Currently all groups get "equal" cpu bandwidth. I plan
  to support weighted fair-sharing on the lines of task
  niceness.
- Separate out tunable
  Right now tunable are same for all layers of scheduling.
  I strongly think we will need to separate them, esp
  sysctl_sched_runtime_limit.
- Optimization
- reduce frequency of timer tick processing at higher levels
- during load balance, pick cache-cold tasks first to migrate
- hierarchy flattening
  Experiment with this (to reduce number of hierarchical levels)
  as per http://lkml.org/lkml/2007/5/26/81

Some results follows. Legends used in them are:

cfs = base cfs performance (sched-cfs-v2.6.22-rc4-mm2-v18.patch)
cfsgrpdi = base cfs + patches 1-2 applied (CONFIG_FAIR_GROUP_SCHED disabled)
cfsgrpdi = base cfs + patches 1-2 applied (CONFIG_FAIR_GROUP_SCHED enabled)

All tests run on a 4-cpu Intel Xeon (x86_64) box:

A. Overhead Test

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-	6.7400	7.8200	8.0100	8.7900	10.90	8.20000	19.88
cfsgrpdi	Linux 2.6.22-	6.7000	7.6700	8.0700	9.0100	11.54	9.34000	18.71
cfsgrpen	Linux 2.6.22-	7.8600	7.8700	8.6500	9.4600	10.27	9.44000	19.74

hackbench -pipe 100
==========
Average of 10 runs was taken. Smaller numbers are better.

cfs 4.0171
cfsgrpdi 4.154
cfsgrpen 4.7749

B. UP Group fairness test
 These tests were forced to run on a single CPU by making using
 of exclusive cpusets.

hackbench
=========

The two user's shell were put in different groups (as explained in Patch 2/2).
Each user then ran this script:

```
i=0
while [ $i -lt 10 ]
do
./hackbench -pipe 100 >> log
i=`expr $i + 1`
done
```

Time taken to complete this script was measured as follows (note that
both the scripts were made to run simultaneously on /same/ cpu).

vatsa 103.51 s (real)
guest 103.37 s (real)

Inference: Both users completed the same amount of work in (nearly) same time.

kernel compilation
==================

Again the two user's shell were put in different groups.

User vatsa ran "make -s -j4 bzImage", while
User guest ran "make -s -j20 bzImage"

Both are compiling the same sources (and hence should effectively be
doing the same amount of work). Time taken to complete kernel-compile by
both users:

vatsa 777.46 s (real)
guest 778.30 s (real)

Inference: Both users completed the same amount of work in nearly same
time, even though one had higher number of threads dedicated to the job.

C. SMP Fairness test
====================

I used a simple cpu-intensive program which measures how much CPU time it got (using getrusage) over a minute. N (=4*NUM_CPUS) such tasks were spawned with N/2 in one group and N/2 in another group. Total CPU time obtained by one group was compared with total cpu time obtained by another group. While the test was running, I observed distribution of all tasks across CPUs. I am quite happy with the results obtained and with the load distribution. I can share the sources/results of the program/script upon request.

Looking forward to your feedback on these patches!

[P.S : Since I am travelling this weekend, I may not respond promptly]

--
Regards,
vatsa

_____