CPU Caps

Alexander Kolbasov (alexander.kolbasov@sun.com)

January 10, 2007

1 Summary

CPU caps provide absolute fine-grained limits on the amount of CPU resources that can be consumed by a project or a zone. It is provided as an extension to `resource_controls`(5) . This document provides a short description specifically for PSARC. A full description is available in the design document [1] and the implementation is discussed in the implementation guide [2]. The on-line version of this document is available in [5].

CPU caps extend existing facilities that control CPU usage on the system – processor binding, processor sets and the Fair Share Scheduler. Unlike processor binding and processor sets, CPU caps allow applications to run on any CPUs while limiting their CPU usage and provide fine-grained (specified in fractions of a CPU) limit. In contrast to the `FSS`(7) , CPU caps provide absolute usage limit that does not depend on other applications running in the system. CPU caps can be used together with CPU binding, processor sets and FSS. When used in conjunction with processor sets, CPU caps limit CPU usage within a set. When used with `FSS`(7) , CPU resources defined by caps are further subdivided using `FSS`(7) shares.

CPU caps provide several advantages over the mechanisms described above:

- Allow customers more flexibility in managing CPU resources than processor sets.
- Easier to manage and use than shares.
- Provide simpler semantics compared to CPU shares.

The CPU caps are administered using the `resource_controls`(5) framework or the `zonecfg`(1M) command (see section 3). Two new resource controls are introduced to set per-project and per-zone CPU caps (see section 6.1).

System administrators can use kstats to observe impact of CPU caps at the zone or project level (see 5.1). The DTrace `sched` provider is extended with new probes which can be used for per-process or per-thread analysis (see 5.3).

All extensions described here are *Committed* except for the kstats which are *Uncommitted*. Patch binding is requested for all.

2 Motivation

Customers provided the following major reasons for wanting CPU caps:

- Selling CPU resources.

  Customers selling their CPU resources want to provide usage limits based on the amount of CPU resources purchased.

- Managing expectations.

  Customers want the clients buying CPU resources to have the same experience in terms of their application performance independent of the machine load. In particular, they do not want clients running on an idle system experience better performance than when running in parallel with other clients applications. Note that such behavior can not be achieved with the FSS since it does not limit CPU resources when idle CPUs are present.

- Over-subscription

  Some customers want to use hard limits as a mechanism to over-subscribe users. They may sell more CPU resource than what is available and they will be able to provide the level of service unless they run out of CPU cycles in which case there will be some decline in performance.

In some form, such mechanism for limiting CPU resources is provided by other operating systems. It became one of many check-box items which customers look at when comparing resource management solutions from Sun and other vendors.

2.1 Related CPU resource management tools

Solaris provides different mechanisms which can be used to control CPU usage of applications to some extent:

- Processor binding (see `pbind`(1M)) provides a way to limit process or a set of processes to a single CPU. It allows other unbound threads to run on the same CPU and compete for CPU cycles with bound threads.

- Processor sets (see `psrset`(1M)) provide a way to limit process execution to a set of CPUs. It also prohibits threads not belonging to the set from running within the set.

- Dynamic resource pools (see `pooladm`(1M)) integrate processor sets with Zones.

- Fair Share Scheduler (see `FSS`(7)) provides a mechanism to share available CPUs within given proportions (shares). The *Fair Share Scheduler* only controls CPU usage relative to CPU usage of other applications. CPU shares do not limit CPU usage when there are idle CPU resources.

CPU binding and processor sets provide *hard* limits for applications since they only run on a bound CPU or within a set, but the minimum granularity for both is a single CPU. The `FSS`(7) provides *soft* limits which are enforced only when there is enough competition for CPU resources and is specified in units expressing relationships of processes to each other.

3 Administrative Interface

CPU caps are represented by two new resource controls. The cap value is specified in units of one per cent of a CPU regardless of how many CPUs are available or their characteristics. The value should be greater than zero.

3.1 Project CPU caps

A project CPU cap is represented by the `project.cpu-cap` resource control. It is associated with the *privileged level* and can be modified only by privileged (superuser) callers (see `resource_controls`(5) and `prctl`(1) for the description of *privileged level*).

The `project.cpu-cap` resource limit can be set statically for projects in `project`(4) file. For example, the following line in `project`(4) file sets persistent CPU cap of 6 CPUs for user akolb:

```
user.akolb:1234::::project.cpu-cap=(privileged,600,none)
```

Project CPU cap can be also dynamically modified or removed on a running system using `prctl`(1) utility. For example, the following command modifies the CPU cap to limit user akolb to 3 CPUs:

```
$ prctl -r -t privileged -n project.cpu-cap -v 300 -i project user.akolb
```

3.2 Zone CPU caps

A zone CPU caps is represented by the `zone.cpu-cap` resource control. Similar to the project cap it is associated with the *privileged level* and can be modified only by privileged (superuser) callers.

The `zone.cpu-cap` resource can be set for a zone using `zonecfg`(1M) command. The following example configures CPU cap for a zone to 3 CPUs:

```
zonecfg:myzone> add rctl
zonecfg:myzone:rctl> set name=zone.cpu-cap
zonecfg:myzone:rctl> add value (priv=privileged,limit=300,action=none)
zonecfg:myzone:rctl> end
```

3.2.1 zonecfg(1M) extensions for CPU caps

The zonecfg(1M) is extended with a new resource called capped-cpu, as described in PSARC/2006/496 [3]. The resource value, called ncpus, maps to the zone.cpu-cap rctl.

The following example sets zone CPU cap using the capped-cpu resource:

```
zonecfg:myzone> add capped-cpu
zonecfg:myzone>capped-cpu> set ncpus=3
zonecfg:myzone>capped-cpu>capped-cpu> end
```

The PSARC/2006/496 [3] describes the capped-cpu resource in the following way:

> The capped-cpu resource has a single ncpus property which is a positive decimal with two digits to the right of the decimal. This property is implemented as a special case of the zonecfg cpu-cap alias. The special case handling of this property normalizes the value so that it corresponds to units of cpus and is similar to the ncpus property under the dedicated-cpu resource group. Unlike dedicated-cpu it will not accept a range and it will accept a decimal number. For example, when using ncpus in the dedicated-cpu resource group, a value of 1 means one dedicated cpu. When using ncpus in the capped-cpu resource group, a value of 1 means 100% of a cpu as the zone.cpu-cap setting. A value of 1.25 means 125%, since 100% corresponds to one full cpu on the system when using cpu caps. The intention here is to align the ncpus units as closely as possible in these two cases (dedicated-cpu vs. capped-cpu), given the limitations and capabilities of the two underlying mechanisms (pset vs. rctl).

See PSARC/2006/496 [3] for a description of the dedicated-cpu resource and ncpu property.

Here we commit to the capped-cpu resource.

4 Implementation Overview

For all threads running in capped projects or zones, the system keeps track of their CPU usage over short periods of time. When CPU usage of projects or zones reaches specified caps, threads in them do not get scheduled and instead are placed on the special *wait queues* in the kernel. These threads will become runnable again only when CPU usage drops below the cap level.

The time spent by threads on *wait queues* is combined with the time spent on *run queues* and is reported as part of the *"wait-cpu"* (latency) time by procfs. The *"wait-cpu"* time is also used to report time spent waiting on *run queues* (see pr_wtime field of struct prusage in proc(4)). Wait times can be seen in the

`LAT` column when `prstat`(1M) is invoked with the `-m` option. CPU time spent by threads on wait queues is also accumulated at the `LMS_WAIT_CPU` micro-state accounting state which is also used to account for time spent waiting on *run queues*.

We decided to combine the wait times that threads spent waiting on *run queues* and *wait queues* into a single bucket because currently there is a fixed set of micro-state accounting types. Any extension of this set will cause offsets of fields in data structures, embedding micro-state data, to change. This, in turn, implies that all `proc`(4) consumers should be recompiled. We feel that CPU wait time is general enough to include time spent waiting on both *run queues* and *wait queues* and it is reasonable to combine them together until micro-state accounting framework is extended.

CPU caps are enforced only for threads running in *TS*, *IA*, *FX*, and *FSS* scheduling classes and have no effect on threads running in *RT* (Real-Time) scheduling class.

All CPU usage accounting data is collected using micro-state accounting facility, so the accuracy of CPU caps depends on the accuracy of the micro-state data.

A more detailed description of the implementation is available in the implementation guide [2].

5 CPU Caps Observability

There are several ways to observe the impact of CPU caps at a zone or project level and at a process or thread level. Each project or zone CPU cap exports information via kstats (see 5.1).

The DTrace `sched` provider is extended with two new probes which can be used for gathering detailed data for times spent on wait queues (see 5.3). These probes provide thread and process level observability for CPU caps.

5.1 Zones and Project observability

Zone and project CPU cap kstats contain the following information:

value – the cap value in percentages of a single CPU

usage – current aggregated CPU usage for all threads belonging to a capped project or zone in percentages of a single CPU

maxusage – maximum observed CPU usage

above_sec – total time in seconds spent above the cap

below_sec – total time in seconds spent below the cap

nwait – number of threads on cap wait queue

For example, when a project CPU cap is set to 50% for project 1234, the following command will show cap information:

```
$ kstat -m caps
module: caps instance: 0
name:   cpucaps_project_1234 class: project_caps
        above_sec    787
        below_sec    260551
        nwait        0
        usage        1
        maxusage     51
        value        50
```

The following example is for zone kstats:

```
module: caps instance: 14
name:   cpucaps_zone_14 class: zone_caps
        above_sec  0
        below_sec  3
        maxusage   255
        nwait      0
        usage      19
        value      300
```

For both zone and project kstats, the kstat instance is the same as zone ID. The PSARC 2006/598 *Swap resource control* case[4] provides a precedents for such kstats and establishes the naming conventions.

The maximum usage statistics provides a good way for users to estimate how to set up their CPU caps. They can set the cap to a very high value and observe the maximum usage while their application is running for a while. This gives an estimate of maximum CPU requirements for the workload.

5.2 Thread level observability

The `ps`(1) command shows threads on the wait queue by displaying "W" for their state. For example:

```
$ ps -o pid,s,comm -p 101262
   PID S COMMAND
101262 W /usr/perl5/bin/perl
```

The `prstat`(1M) command shows *"wait"* state for threads, sitting on wait queues. For example:

```
   PID USERNAME SIZE  RSS  STATE  PRI NICE      TIME  CPU PROCESS/NLWP
100686 akolb   4272K 1516K wait     0    0   0:51:20  25% perl/1
```

5.3 DTrace changes for CPU caps

CPU caps provide two new DTrace `sched` provider probes for observing scheduling impact for threads and processes:

cpucaps-sleep Probe that fires immediately before the current thread is placed on a wait queue. The `lwpsinfo_t` of the waiting thread is pointed to by `args[0]`. The `psinfo_t` of the process containing the waiting thread is pointed to by `args[1]`.

cpucaps-wakeup Probe that fires immediately after a thread is removed from a wait queue. The `lwpsinfo_t` of the waiting thread is pointed to by `args[0]`. The `psinfo_t` of the process containing the waiting thread is pointed to by `args[1]`.

6 Public Documentation Changes

6.1 `resource_controls`(5)

The following description should be added to `resource_controls`(5) man page:

project.cpu-cap Maximum amount of CPU resources that a project can use. The unit used is the percentage of a single CPU (an integer). The cap does not apply to threads running in real-time scheduling class.

zone.cpu-cap Sets a limit on amount of CPU time that can be used by a zone. The cap does not apply to threads running in real-time scheduling class. Projects within the zone can have their own CPU caps. The minimum cap value takes precedence. Expressed as an integer, denoting the percentage of a single CPU that can be used by all user threads in a zone.

6.2 Solaris Dynamic Tracing Guide

The Solaris Dynamic Tracing Guide should be updated to include information about new process states and two new `sched` provider probes.

6.2.1 `proc` Provider

The `proc` *Provider* section of the guide should be updated to reflect the addition of the new `SWAIT` processor state (table `25-5`):

SWAIT(W) The thread is waiting on wait queue. The `sched:::cpucaps-sleep` probe will fire immediately before a thread state is transitioned to `SWAIT`.

6.2.2 Probes

The *Probes* section of the guide (table `26-1`) should be updated with information about two new probes:

cpucaps-sleep Probe that fires immediately before the current thread is placed on a wait queue. The `lwpsinfo_t` of the waiting thread is pointed to by `args[0]`. The `psinfo_t` of the process containing the waiting thread is pointed to by `args[1]`.

cpucaps-wakeup Probe that fires immediately after a thread is removed from a wait queue. The `lwpsinfo_t` of the waiting thread is pointed to by `args[0]`. The `psinfo_t` of the process containing the waiting thread is pointed to by `args[1]`.

6.2.3 Arguments

The *Arguments* section of the guide (table `26-2`) should be updated with information in following table, describing arguments for CPU Caps probe arguments:

Probe	args[0]	args[1]
`cpucaps-sleep`	`lwpsinfo_t *`	`psinfo_t *`
`cpucaps-wakeup`	`lwpsinfo_t *`	`psinfo_t *`

6.2.4 `cpucaps-sleep` and `cpucaps-wakeup` Examples

The *Examples* section should be updated with examples for CPU Caps probe arguments.

A Summary of exported interfaces

Here is a summary of proposed interface changes. All of them, except kstats are *Committed*.

- New resource controls:
  - `project.cpu-cap`
  - `zone.cpu-cap`
- `zonecfg`(1M) extensions:
  - `capped-cpu` resource
  - `ncpus` property
- zone and project kstats for CPU caps
- `proc`(4) consumers:

  - "`W`" state shown by `ps`(1) for threads on wait queues
  - "`sleep`" state shown by `prstat`(1M) for threads on wait queues

- DTrace extensions:
  - `cpucaps-sleep` sched provider probe
  - `cpucaps-wakeup` sched provider probe
  - `SWAIT` processor state for threads on wait queues

References

[1] CPU caps web page on OpenSolaris.org
`http://www.opensolaris.org/os/project/rm/rctls/cpu-caps/`

[2] Implementation description
`http://www.opensolaris.org/os/project/rm/rctls/cpu-caps/caps_implementation/`.

[3] PSARC/2006/496 Improved Zones/RM Integration
`http://sac.sfbay.sun.com/PSARC/2006/496`
`http://www.opensolaris.org/os/community/arc/caselog/2006/496`

[4] PSARC 2006/598 Swap resource control; locked memory RM improvements.
`http://sac.sfbay.sun.com/PSARC/2006/598`
`http://www.opensolaris.org/os/community/arc/caselog/2006/598`

[5] On-line version of CPU Caps PSARC case
`http://www.opensolaris.org/os/project/rm/rctls/cpu-caps/caps_psarc/`