Subject: [PATCH 0/5] per-cpu/cpuacct cgroup scheduler statistics
Posted by Glauber Costa on Thu, 02 Feb 2012 14:19:27 GMT
View Forum Message <> Reply to Message

Hi,

Here is my new attempt to get a per-container version of some
/proc data such as /proc/stat and /proc/uptime.

In this series I solved the visibility problem, which is,
the problem of how and when to show /proc/stat data per-cgroup,
by declaring it not a problem.

This can probably be done in userspace with other aids, like mounting
a fuse overlay that simulates /proc from outside a container, to a
container location.

Here, we should have most of the data needed to do that. They are drawn
from both the cpu cgroup, and cpuacct. Each cgroup exports the data it
knows better, and I am not really worried here about bindings between them.

In this first version, I am using clock_t units, being quite proc-centric.
It made my testing easier, but I am happy to show any units you guys would
prefer.

Besides that, it still has some other minor issues to be sorted out.
But I verified the general direction to be working, and would like to know
what you think.

Thanks

Glauber Costa (5):
  make steal time's to-tick routine generic
  store number of iowait events in a task_group
  account guest time per-cgroup as well.
  expose fine-grained per-cpu data for cpuacct stats
  expose per-taskgroup schedstats in cgroup

 include/linux/sched.h | 1 +
 kernel/sched/core.c | 207 ++----
 kernel/sched/fair.c | 45 ++++++++++
 kernel/sched/sched.h | 3 +
 4 files changed, 242 insertions(+), 14 deletions(-)

--
1.7.7.4

Subject: [PATCH 2/5] store number of iowait events in a task_group
Posted by Glauber Costa on Thu, 02 Feb 2012 14:19:29 GMT
View Forum Message <> Reply to Message

Instead of just having the rq to hold them, this patch stores
the nr_iowait figures for each task_group, except for the root
task group. That one is kept using the numbers originating from
the rq.

Signed-off-by: Glauber Costa <glommer@parallels.com>
---
 include/linux/sched.h | 1 +
 kernel/sched/core.c | 42 ++++++++++++++++++++++++++++++++++++++----
 2 files changed, 39 insertions(+), 4 deletions(-)

diff --git a/include/linux/sched.h b/include/linux/sched.h
index 5b8ff53..b629c1e 100644
--- a/include/linux/sched.h
+++ b/include/linux/sched.h
@@ -1207,6 +1207,7 @@ struct sched_entity {

  u64   nr_migrations;

+ atomic_t   nr_iowait;
 #ifdef CONFIG_SCHEDSTATS
  struct sched_statistics statistics;
 #endif
diff --git a/kernel/sched/core.c b/kernel/sched/core.c
index 455810f..fe35316 100644
--- a/kernel/sched/core.c
+++ b/kernel/sched/core.c
@@ -2665,7 +2665,41 @@ static inline void task_group_account_field(struct task_struct *p, int
index,
 #endif
 }

+static void task_group_inc_nr_iowait(struct task_struct *p, int cpu)
+{
+ struct task_group *tg;
+ struct rq *rq = cpu_rq(cpu);
+
+ rcu_read_lock();
+ tg = task_group(p);
+
+ atomic_inc(&rq->nr_iowait);
+
+ while (tg && tg != &root_task_group) {
+ atomic_inc(&tg->se[cpu]->nr_iowait);
+ tg = tg->parent;

```
+ }
+ rcu_read_unlock();
+
+}
+
+static void task_group_dec_nr_iowait(struct task_struct *p, int cpu)
+{
+ struct task_group *tg;
+ struct rq *rq = cpu_rq(cpu);
+
+ rcu_read_lock();
+ tg = task_group(p);
+
+ atomic_dec(&rq->nr_iowait);
+
+ while (tg && tg != &root_task_group) {
+  atomic_dec(&tg->se[cpu]->nr_iowait);
+  tg = tg->parent;
+ }
+ rcu_read_unlock();

+}
 /*
  * Account user cpu time to a process.
  * @p: the process that the cpu time gets accounted to
@@ -4677,12 +4711,12 @@ void __sched io_schedule(void)
  struct rq *rq = raw_rq();

  delayacct_blkio_start();
- atomic_inc(&rq->nr_iowait);
+ task_group_inc_nr_iowait(current, cpu_of(rq));
  blk_flush_plug(current);
  current->in_iowait = 1;
  schedule();
  current->in_iowait = 0;
- atomic_dec(&rq->nr_iowait);
+ task_group_dec_nr_iowait(current, cpu_of(rq));
  delayacct_blkio_end();
 }
 EXPORT_SYMBOL(io_schedule);
@@ -4693,12 +4727,12 @@ long __sched io_schedule_timeout(long timeout)
  long ret;

  delayacct_blkio_start();
- atomic_inc(&rq->nr_iowait);
+ task_group_inc_nr_iowait(current, cpu_of(rq));
  blk_flush_plug(current);
  current->in_iowait = 1;
```

```
  ret = schedule_timeout(timeout);
  current->in_iowait = 0;
- atomic_dec(&rq->nr_iowait);
+ task_group_dec_nr_iowait(current, cpu_of(rq));
  delayacct_blkio_end();
  return ret;
 }
--
1.7.7.4
```

Subject: [PATCH 3/5] account guest time per-cgroup as well.
Posted by Glauber Costa on Thu, 02 Feb 2012 14:19:30 GMT
View Forum Message <> Reply to Message

We already track multiple tick statistics per-cgroup, using
the task_group_account_field facility. This patch accounts
guest_time in that manner as well.

Signed-off-by: Glauber Costa <glommer@parallels.com>
---
 kernel/sched/core.c | 10 ++++------
 1 files changed, 4 insertions(+), 6 deletions(-)

```
diff --git a/kernel/sched/core.c b/kernel/sched/core.c
index fe35316..91ea913 100644
--- a/kernel/sched/core.c
+++ b/kernel/sched/core.c
@@ -2734,8 +2734,6 @@ void account_user_time(struct task_struct *p, cputime_t cputime,
 static void account_guest_time(struct task_struct *p, cputime_t cputime,
          cputime_t cputime_scaled)
 {
- u64 *cpustat = kcpustat_this_cpu->cpustat;
-
  /* Add guest time to process. */
  p->utime += cputime;
  p->utimescaled += cputime_scaled;
@@ -2744,11 +2742,11 @@ static void account_guest_time(struct task_struct *p, cputime_t
cputime,

  /* Add guest time to cpustat. */
  if (TASK_NICE(p) > 0) {
-  cpustat[CPUTIME_NICE] += (__force u64) cputime;
-  cpustat[CPUTIME_GUEST_NICE] += (__force u64) cputime;
+  task_group_account_field(p, CPUTIME_NICE, (__force u64) cputime);
+  task_group_account_field(p, CPUTIME_GUEST, (__force u64) cputime);
  } else {
-  cpustat[CPUTIME_USER] += (__force u64) cputime;
```

```
-  cpustat[CPUTIME_GUEST] += (__force u64) cputime;
+  task_group_account_field(p, CPUTIME_USER, (__force u64) cputime);
+  task_group_account_field(p, CPUTIME_GUEST, (__force u64) cputime);
  }
 }

--
1.7.7.4
```

Subject: [PATCH 4/5] expose fine-grained per-cpu data for cpuacct stats
Posted by Glauber Costa on Thu, 02 Feb 2012 14:19:31 GMT
View Forum Message <> Reply to Message

The cpuacct cgroup already exposes user and system numbers in a per-cgroup fashion. But they are a summation along the whole group, not a per-cpu figure. Also, they are coarse-grained version of the stats usually shown at places like /proc/stat.

I want to have enough cgroup data to emulate the /proc/stat interface. To achieve that, I am creating a new file "stat_percpu" that displays the fine-grained per-cpu data. The original data is left alone.

Note that in this first version, I am using clock_t units, being quite proc-centric. It made my testing easier, but I am happy to show any units you guys would prefer.

Signed-off-by: Glauber Costa <glommer@parallels.com>

```
---
 kernel/sched/core.c |   28 ++++++++++++++++++++++++++++
 1 files changed, 28 insertions(+), 0 deletions(-)

diff --git a/kernel/sched/core.c b/kernel/sched/core.c
index 91ea913..013ca9c 100644
--- a/kernel/sched/core.c
+++ b/kernel/sched/core.c
@@ -8308,6 +8308,29 @@ static int cpuacct_stats_show(struct cgroup *cgrp, struct cftype *cft,
  return 0;
 }

+static int cpuacct_stats_percpu_show(struct cgroup *cgrp, struct cftype *cft,
+         struct seq_file *m)
+{
+ struct cpuacct *ca = cgroup_ca(cgrp);
+ int cpu;
+
+ for_each_online_cpu(cpu) {
+  struct kernel_cpustat *kcpustat = per_cpu_ptr(ca->cpustat, cpu);
```

```
+  seq_printf(m,
+  "cpu%d %llu %llu %llu %llu %llu %llu %llu\n", cpu,
+  (unsigned long long)cputime_to_clock_t(kcpustat->cpustat[CPUTIME_USER]),
+  (unsigned long long)cputime_to_clock_t(kcpustat->cpustat[CPUTIME_NICE]),
+  (unsigned long long)cputime_to_clock_t(kcpustat->cpustat[CPUTIME_SYSTEM]),
+  (unsigned long long)cputime_to_clock_t(kcpustat->cpustat[CPUTIME_IRQ]),
+  (unsigned long long)cputime_to_clock_t(kcpustat->cpustat[CPUTIME_SOFTIRQ]),
+  (unsigned long long)cputime_to_clock_t(kcpustat->cpustat[CPUTIME_GUEST]),
+  (unsigned long long)cputime_to_clock_t(kcpustat->cpustat[CPUTIME_GUEST_NICE])
+  );
+ }
+
+ return 0;
+}
+
 static struct cftype files[] = {
  {
   .name = "usage",
@@ -8322,6 +8345,11 @@ static struct cftype files[] = {
   .name = "stat",
   .read_map = cpuacct_stats_show,
  },
+ {
+  .name = "stat_percpu",
+  .read_seq_string = cpuacct_stats_percpu_show,
+ },
+
 };

 static int cpuacct_populate(struct cgroup_subsys *ss, struct cgroup *cgrp)
--
1.7.7.4
```

Subject: [PATCH 5/5] expose per-taskgroup schedstats in cgroup
Posted by Glauber Costa on Thu, 02 Feb 2012 14:19:32 GMT
View Forum Message <> Reply to Message

This patch aims at exposing stat information per-cgroup, such as:
 * idle time,
 * iowait time,
 * steal time,
and friends. The ultimate goal is to be able to present a per-container view of
/proc/stat inside a container. With this patch, everything that is needed to do
that is in place, except for number of switches and number of tasks.

I achieve that by hooking into the schedstats framework, so although the
overhead of that is prone to discussion, I am not adding anything, but reusing

what's already there instead. The exception being that the data is now computed
and stored in non-task se's as well, instead of entity_is_task() branches.
However, I expect this to be minimum comparing to the alternative of adding new
hierarchy walks. Those are kept intact.

Note that in this first version, I am using clock_t units, being quite
proc-centric. It made my testing easier, but I am happy to show any units
you guys would prefer.

Signed-off-by: Glauber Costa <glommer@parallels.com>

```
---
 kernel/sched/core.c  |  114 ++++++++++++++++++++++++++++++++++++++++++++++++++
 kernel/sched/fair.c  |   45 ++++++++++++++++++++
 kernel/sched/sched.h |    3 +
 3 files changed, 162 insertions(+), 0 deletions(-)

diff --git a/kernel/sched/core.c b/kernel/sched/core.c
index 013ca9c..fc2b9ed 100644
--- a/kernel/sched/core.c
+++ b/kernel/sched/core.c
@@ -7975,6 +7975,107 @@ static u64 cpu_rt_period_read_uint(struct cgroup *cgrp, struct cftype *cft)
 }
 #endif /* CONFIG_RT_GROUP_SCHED */

+#ifdef CONFIG_SCHEDSTATS
+
+#ifdef CONFIG_FAIR_GROUP_SCHED
+#define fair_rq(field, tg, i)  tg->cfs_rq[i]->field
+#define fair_se(field, tg, i)  tg->se[i]->statistics.field
+#else
+#define fair_rq(field, tg, i)  0
+#endif
+
+#ifdef CONFIG_RT_GROUP_SCHED
+#define rt_rq(field, tg, i)  tg->rt_rq[i]->field
+#else
+#define rt_rq(field, tg, i)  0
+#endif
+
+static u64 tg_nr_running(struct task_group *tg, int cpu)
+{
+ /*
+  * because of autogrouped groups in root_task_group, the
+  * following does not hold.
+  */
+ if (tg != &root_task_group)
+  return rt_rq(rt_nr_running, tg, cpu) + fair_rq(nr_running, tg, cpu);
```

```
+
+ return cpu_rq(cpu)->nr_running;
+}
+
+static u64 tg_idle(struct task_group *tg, int cpu)
+{
+ u64 *cpustat = kcpustat_this_cpu->cpustat;
+ u64 val;
+
+ if (tg != &root_task_group) {
+  val = cfs_read_sleep(tg->se[cpu]);
+  /* If we have rt tasks running, we're not really idle */
+  val -= rt_rq(exec_clock, tg, cpu);
+  val = nsec_to_tick(val);
+ } else
+  val = cpustat[CPUTIME_IDLE];
+
+ return cputime_to_clock_t(val);
+}
+
+static u64 tg_steal(struct task_group *tg, int cpu)
+{
+ u64 *cpustat = kcpustat_this_cpu->cpustat;
+ u64 val = cpustat[CPUTIME_STEAL];
+
+ if (tg != &root_task_group)
+  val = nsec_to_tick(cfs_read_wait(tg->se[cpu]));
+ else
+  val = cpustat[CPUTIME_STEAL];
+
+ return cputime_to_clock_t(val);
+}
+
+static u64 tg_nr_iowait(struct task_group *tg, int cpu)
+{
+ if (tg != &root_task_group)
+  return atomic_read(&tg->se[cpu]->nr_iowait);
+
+ return atomic_read(&cpu_rq(cpu)->nr_iowait);
+}
+
+static u64 tg_iowait(struct task_group *tg, int cpu)
+{
+ u64 *cpustat = kcpustat_this_cpu->cpustat;
+ u64 val = 0;
+
+ if (tg != &root_task_group)
+  val = nsec_to_tick(cfs_read_iowait(tg->se[cpu]));
```

```
+ else
+  val = cpustat[CPUTIME_IOWAIT];
+
+ return cputime_to_clock_t(val);
+}
+
+static int cpu_schedstats_show(struct cgroup *cgrp, struct cftype *cft,
+          struct seq_file *m)
+{
+ struct task_group *tg = cgroup_tg(cgrp);
+ int cpu;
+ /*
+  * TODO: nr_switches is one of the statistics we're interested in, but
+  * it is potentially too heavy on the scheduler.
+  */
+ u64 nr_switches = 0;
+
+ for_each_online_cpu(cpu) {
+  seq_printf(m,
+  "cpu%d %llu %llu %llu %llu %llu\n",
+  cpu, tg_idle(tg, cpu), tg_iowait(tg, cpu), tg_steal(tg, cpu),
+  tg_nr_iowait(tg,cpu), nr_switches,
+  tg_nr_running(tg, cpu)
+  );
+ }
+
+ return 0;
+}
+#endif
+
 static struct cftype cpu_files[] = {
 #ifdef CONFIG_FAIR_GROUP_SCHED
  {
@@ -7982,6 +8083,19 @@ static struct cftype cpu_files[] = {
   .read_u64 = cpu_shares_read_u64,
   .write_u64 = cpu_shares_write_u64,
  },
+/*
+ * In theory, those could be done using the rt tasks as a basis
+ * as well. Since we're interested in figures like idle, iowait, etc
+ * for the whole cgroup, the results should be the same.
+ * But that only complicates the code, and I doubt anyone using !FAIR_GROUP_SCHED
+ * is terribly interested in those.
+ */
+#ifdef CONFIG_SCHEDSTATS
+ {
+  .name = "schedstat_percpu",
+  .read_seq_string = cpu_schedstats_show,
```

```
+ },
+#endif
 #endif
 #ifdef CONFIG_CFS_BANDWIDTH
  {
diff --git a/kernel/sched/fair.c b/kernel/sched/fair.c
index e301ba4..5305bb1 100644
--- a/kernel/sched/fair.c
+++ b/kernel/sched/fair.c
@@ -721,6 +721,41 @@ update_stats_wait_start(struct cfs_rq *cfs_rq, struct sched_entity *se)
  schedstat_set(se->statistics.wait_start, rq_of(cfs_rq)->clock);
 }

+#ifdef CONFIG_SCHEDSTATS
+u64 cfs_read_sleep(struct sched_entity *se)
+{
+ struct cfs_rq *cfs_rq = se->cfs_rq;
+ u64 value = se->statistics.sum_sleep_runtime;
+
+ if (!se->statistics.sleep_start)
+  return value;
+
+ return value + rq_of(cfs_rq)->clock - se->statistics.sleep_start;
+}
+
+u64 cfs_read_iowait(struct sched_entity *se)
+{
+ struct cfs_rq *cfs_rq = se->cfs_rq;
+ u64 value = se->statistics.iowait_sum;
+
+ if (!se->statistics.block_start)
+  return value;
+
+ return value + rq_of(cfs_rq)->clock - se->statistics.block_start;
+}
+
+u64 cfs_read_wait(struct sched_entity *se)
+{
+ struct cfs_rq *cfs_rq = se->cfs_rq;
+ u64 value = se->statistics.wait_sum;
+
+ if (!se->statistics.wait_start)
+  return value;
+
+ return value + rq_of(cfs_rq)->clock - se->statistics.wait_start;
+}
+#endif
+
```

```
 /*
  * Task is being enqueued - update stats:
  */
@@ -1046,6 +1081,10 @@ static void enqueue_sleeper(struct cfs_rq *cfs_rq, struct sched_entity
*se)
    }
    account_scheduler_latency(tsk, delta >> 10, 0);
   }
+  else if (atomic_read(&se->nr_iowait)) {
+   se->statistics.iowait_sum += delta;
+   se->statistics.iowait_count++;
+  }
  }
 #endif
 }
@@ -1199,6 +1238,12 @@ dequeue_entity(struct cfs_rq *cfs_rq, struct sched_entity *se, int
flags)
     se->statistics.sleep_start = rq_of(cfs_rq)->clock;
    if (tsk->state & TASK_UNINTERRUPTIBLE)
     se->statistics.block_start = rq_of(cfs_rq)->clock;
+  } else {
+   if (atomic_read(&se->nr_iowait))
+    se->statistics.block_start = rq_of(cfs_rq)->clock;
+   else
+    se->statistics.sleep_start = rq_of(cfs_rq)->clock;
+
   }
 #endif
  }
diff --git a/kernel/sched/sched.h b/kernel/sched/sched.h
index 53d13dd..7ec2482 100644
--- a/kernel/sched/sched.h
+++ b/kernel/sched/sched.h
@@ -1156,6 +1156,9 @@ extern void init_rt_rq(struct rt_rq *rt_rq, struct rq *rq);
 extern void unthrottle_offline_cfs_rqs(struct rq *rq);

 extern void account_cfs_bandwidth_used(int enabled, int was_enabled);
+extern u64 cfs_read_sleep(struct sched_entity *se);
+extern u64 cfs_read_iowait(struct sched_entity *se);
+extern u64 cfs_read_wait(struct sched_entity *se);

 #ifdef CONFIG_NO_HZ
 enum rq_nohz_flag_bits {
--
1.7.7.4
```

Subject: Re: [PATCH 0/5] per-cpu/cpuacct cgroup scheduler statistics
Posted by Glauber Costa on Tue, 14 Feb 2012 11:16:27 GMT
View Forum Message <> Reply to Message

On 02/02/2012 06:19 PM, Glauber Costa wrote:
> Hi,
>
> Here is my new attempt to get a per-container version of some
> /proc data such as /proc/stat and /proc/uptime.
>
> In this series I solved the visibility problem, which is,
> the problem of how and when to show /proc/stat data per-cgroup,
> by declaring it not a problem.
>
> This can probably be done in userspace with other aids, like mounting
> a fuse overlay that simulates /proc from outside a container, to a
> container location.
>
> Here, we should have most of the data needed to do that. They are drawn
> from both the cpu cgroup, and cpuacct. Each cgroup exports the data it
> knows better, and I am not really worried here about bindings between them.
>
> In this first version, I am using clock_t units, being quite proc-centric.
> It made my testing easier, but I am happy to show any units you guys would
> prefer.
>
> Besides that, it still has some other minor issues to be sorted out.
> But I verified the general direction to be working, and would like to know
> what you think.
>

Hi,

Did someone had any chance to take a look at this already?

Thanks

Subject: Re: [PATCH 0/5] per-cpu/cpuacct cgroup scheduler statistics
Posted by Serge E. Hallyn on Tue, 14 Feb 2012 22:31:40 GMT
View Forum Message <> Reply to Message

Quoting Glauber Costa (glommer@parallels.com):
> On 02/02/2012 06:19 PM, Glauber Costa wrote:
> >Hi,
> >
> >Here is my new attempt to get a per-container version of some
> >/proc data such as /proc/stat and /proc/uptime.

> >
> >In this series I solved the visibility problem, which is,
> >the problem of how and when to show /proc/stat data per-cgroup,
> >by declaring it not a problem.
> >
> >This can probably be done in userspace with other aids, like mounting
> >a fuse overlay that simulates /proc from outside a container, to a
> >container location.
> >
> >Here, we should have most of the data needed to do that. They are drawn
> >from both the cpu cgroup, and cpuacct. Each cgroup exports the data it
> >knows better, and I am not really worried here about bindings between them.
> >
> >In this first version, I am using clock_t units, being quite proc-centric.
> >It made my testing easier, but I am happy to show any units you guys would
> >prefer.
> >
> >Besides that, it still has some other minor issues to be sorted out.
> >But I verified the general direction to be working, and would like to know
> >what you think.
> >
>
> Hi,
>
> Did someone had any chance to take a look at this already?
>
> Thanks

Hi,

By declaring proc visibility not a problem and sticking to io stats,
you sort of left me where I don't know what I'm talking about :) So
let me just say, on patch 2, "store number of iowait events in a task_group",
my initial reaction is "boy that's a lot more work. What is the performance
impact?"

It'd be possible to move the extra processing out of the hot-path by
only changing the # for the deepest cgroup, and pulling it into
ancestor cgroups only when someone is viewing the stats or the child
cgroup goes away. But if you have #s showing statistically negligable
performance impact anyway then that wouldn't be worth it.

-serge

Subject: Re: [PATCH 0/5] per-cpu/cpuacct cgroup scheduler statistics
Posted by Glauber Costa on Thu, 16 Feb 2012 10:06:11 GMT

On 02/15/2012 02:31 AM, Serge Hallyn wrote:
> Quoting Glauber Costa (glommer@parallels.com):
>> On 02/02/2012 06:19 PM, Glauber Costa wrote:
>>> Hi,
>>>
>>> Here is my new attempt to get a per-container version of some
>>> /proc data such as /proc/stat and /proc/uptime.
>>>
>>> In this series I solved the visibility problem, which is,
>>> the problem of how and when to show /proc/stat data per-cgroup,
>>> by declaring it not a problem.
>>>
>>> This can probably be done in userspace with other aids, like mounting
>>> a fuse overlay that simulates /proc from outside a container, to a
>>> container location.
>>>
>>> Here, we should have most of the data needed to do that. They are drawn
>> >from both the cpu cgroup, and cpuacct. Each cgroup exports the data it
>>> knows better, and I am not really worried here about bindings between them.
>>>
>>> In this first version, I am using clock_t units, being quite proc-centric.
>>> It made my testing easier, but I am happy to show any units you guys would
>>> prefer.
>>>
>>> Besides that, it still has some other minor issues to be sorted out.
>>> But I verified the general direction to be working, and would like to know
>>> what you think.
>>>
>>
>> Hi,
>>
>> Did someone had any chance to take a look at this already?
>>
>> Thanks
>
> Hi,
>
> By declaring proc visibility not a problem and sticking to io stats,
> you sort of left me where I don't know what I'm talking about :)

Heh. Do you at least agree with the approach of just dumping the
information in cgroup files, so we can join them later? (be it via
userspace or in a follow up kernel patch if the need really arises from
real workloads?)

Do you have any comments on any preferred format?

> So
> let me just say, on patch 2, "store number of iowait events in a task_group",
> my initial reaction is "boy that's a lot more work. What is the performance
> impact?"
Yeah, The first thing I need to do if I'm carrying this forward is to
measure that.

>
> It'd be possible to move the extra processing out of the hot-path by
> only changing the # for the deepest cgroup, and pulling it into
> ancestor cgroups only when someone is viewing the stats or the child
> cgroup goes away.

In principle, should be doable. We discussed this briefly (me and Peter)
once, and the problem is that it of course imposes a hit on the readers.
If you're reading often enough (can be the case for things polling
/proc/stat), this can be a problem.

But if we are really doing this, we can very well do it for all stats,
not only iowait...

> But if you have #s showing statistically negligable
> performance impact anyway then that wouldn't be worth it.
>
Need to work on that.
