Subject: Re: Per container statistics (containerstats)
Posted by Andrew Morton on Thu, 07 Jun 2007 22:54:45 GMT
View Forum Message <> Reply to Message

On Wed, 6 Jun 2007 17:28:13 +0530
Balbir Singh <balbir@linux.vnet.ibm.com> wrote:

```
> Hi, Andrew/Paul,
> 
> Here's the latest version of containerstats ported to v10. Could you
> please consider it for inclusion
> 
> Changelog
> 
> 1. Instead of parsing long container path's use the dentry to match the
>    container for which stats are required. The user space application
>    opens the container directory and passes the file descriptor, which
>    is used to determine the container for which stats are required.
>    This approach was suggested by Paul Menage
> 
> This patch is inspired by the discussion at http://lkml.org/lkml/2007/4/11/187
> and implements per container statistics as suggested by Andrew Morton
> in http://lkml.org/lkml/2007/4/11/263. The patch is on top of 2.6.21-mm1
> with Paul's containers v9 patches (forward ported)
> 
> This patch implements per container statistics infrastructure and re-uses
> code from the taskstats interface. A new set of container operations are
> registered with commands and attributes. It should be very easy to
> *extend* per container statistics, by adding members to the containerstats
> structure.
> 
> The current model for containerstats is a pull, a push model (to post
> statistics on interesting events), should be very easy to add. Currently
> user space requests for statistics by passing the container file descriptor.
> Statistics about the state of all the tasks in the container is returned to
> user space.
> 
> TODO's/NOTE:
> 
> This patch provides an infrastructure for implementing container statistics.
> Based on the needs of each controller, we can incrementally add more statistics,
> event based support for notification of statistics, accumulation of taskstats
> into container statistics in the future.
> 
> Sample output
> 
> # ./containerstats -C /container/a
> sleeping 2, blocked 0, running 1, stopped 0, uninterruptible 0
```

```
>
> # ./containerstats -C /container/
> sleeping 154, blocked 0, running 0, stopped 0, uninterruptible 0
>
> If the approach looks good, I'll enhance and post the user space utility for
> the same
>
> Feedback, comments, test results are always welcome!
>
>
>
> Signed-off-by: Balbir Singh <balbir@linux.vnet.ibm.com>
> ---
>
>  Documentation/accounting/containerstats.txt |   27 ++++++++++
>  include/linux/Kbuild                        |    1
>  include/linux/container.h                   |    8 +++
>  include/linux/containerstats.h              |   70 ++++++++++++++++++++++++++++
>  include/linux/delayacct.h                   |   11 ++++
>  kernel/container.c                          |   63 +++++++++++++++++++++++++
>  kernel/sched.c                              |    4 +
>  kernel/taskstats.c                          |   66 ++++++++++++++++++++++++++
>  8 files changed, 250 insertions(+)
```

I'd have hoped to see containerstats.c in here.

```
> diff -puN /dev/null include/linux/containerstats.h
> --- /dev/null 2007-06-01 20:42:04.000000000 +0530
> +++ linux-2.6.22-rc2-mm1-balbir/include/linux/containerstats.h 2007-06-05 17:23:56.000000000
+0530
> @@ -0,0 +1,70 @@
> +/* containerstats.h - exporting per-container statistics
> + *
> + * __ Copyright IBM Corporation, 2007
> + * Author Balbir Singh <balbir@linux.vnet.ibm.com>
> + *
> + * This program is free software; you can redistribute it and/or modify it
> + * under the terms of version 2.1 of the GNU Lesser General Public License
> + * as published by the Free Software Foundation.
> + *
> + * This program is distributed in the hope that it would be useful, but
> + * WITHOUT ANY WARRANTY; without even the implied warranty of
> + * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
> + */
> +
> +#ifndef _LINUX_CONTAINERSTATS_H
> +#define _LINUX_CONTAINERSTATS_H
> +
> +#include <linux/taskstats.h>
```

I don't understand the relationship between containerstats and taskstats.
afacit it's using the same genetlink channel?

```
> +/*
> + * Data shared between user space and kernel space on a per container
> + * basis. This data is shared using taskstats.
> + *
> + * Most of these states are derived by looking at the task->state value
> + * For the nr_io_wait state, a flag in the delay accounting structure
> + * indicates that the task is waiting on IO
> + *
> + * Each member is aligned to a 8 byte boundary.
> + */
> +struct containerstats {
> + __u64 nr_sleeping;  /* Number of tasks sleeping */
> + __u64 nr_running;  /* Number of tasks running */
> + __u64 nr_stopped;  /* Number of tasks in stopped state */
> + __u64 nr_uninterruptible; /* Number of tasks in uninterruptible */
> +     /* state */
> + __u64 nr_io_wait;  /* Number of tasks waiting on IO */
> +};
> +
> +/*
> + * Commands sent from userspace
> + * Not versioned. New commands should only be inserted at the enum's end
> + * prior to __CONTAINERSTATS_CMD_MAX
> + */
> +
> +enum {
> + CONTAINERSTATS_CMD_UNSPEC = __TASKSTATS_CMD_MAX, /* Reserved */
```

This seems to mean that the containerstats commands all get renumbered if
we add new taskstats commands. That would be bad?

```
> + */
> +int containerstats_build(struct containerstats *stats, struct dentry *dentry)
> +{
> + int ret = -EINVAL;
> + struct task_struct *g, *p;
> + struct container *cont, *root_cont;
> + struct container *src_cont;
> + int subsys_id;
> + struct containerfs_root *root;
> +
> + /*
> +  * Validate dentry by checking the superblock operations
> +  */
> + if (dentry->d_sb->s_op != &container_ops)
> +   goto err;
> +
> + ret = 0;
> + src_cont = (struct container *)dentry->d_fsdata;
```

Unneeded cast.

```
> + rcu_read_lock();
> +
> + for_each_root(root) {
> +  if (!root->subsys_bits)
> +   continue;
> +  root_cont = &root->top_container;
> +  get_first_subsys(root_cont, NULL, &subsys_id);
> +  do_each_thread(g, p) {
```

this needs tasklist_lock?

```
> +   cont = task_container(p, subsys_id);
> +   if (cont == src_cont) {
> +    switch (p->state) {
> +    case TASK_RUNNING:
> +     stats->nr_running++;
> +     break;
> +    case TASK_INTERRUPTIBLE:
> +     stats->nr_sleeping++;
> +     break;
> +    case TASK_UNINTERRUPTIBLE:
> +     stats->nr_uninterruptible++;
> +     break;
> +    case TASK_STOPPED:
> +     stats->nr_stopped++;
> +     break;
> +    default:
> +     if (delayacct_is_task_waiting_on_io(p))
> +      stats->nr_io_wait++;
> +     break;
> +    }
> +   }
> +  } while_each_thread(g, p);
> + }
> + rcu_read_unlock();
> +err:
> + return ret;
> +}
> +
>  static int cmppid(const void *a, const void *b)
>  {
>   return *(pid_t *)a - *(pid_t *)b;
> diff -puN kernel/sched.c~containers-taskstats kernel/sched.c
> --- linux-2.6.22-rc2-mm1/kernel/sched.c~containers-taskstats 2007-06-05 17:21:57.000000000
+0530
> +++ linux-2.6.22-rc2-mm1-balbir/kernel/sched.c 2007-06-05 17:21:57.000000000 +0530
> @@ -4280,11 +4280,13 @@ void __sched io_schedule(void)
>  {
>   struct rq *rq = &__raw_get_cpu_var(runqueues);
>
> + delayacct_set_flag(DELAYACCT_PF_BLKIO);
>   delayacct_blkio_start();
```

Would it be suitable and appropriate to embed the delayacct_set_flag() call
inside delayacct_blkio_start()?