Subject: [RFC][PATCH] Per container statistics
Posted by Balbir Singh on Mon, 14 May 2007 17:44:54 GMT
View Forum Message <> Reply to Message

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 path, if
a container is found to belong to the specified hierarchy, then statistics
about the state of all the tasks in the container is returned to user space.

TODO's

1. All data is shared as filesystem paths, it involves a lot of string
   and path name handling. Simplify and make the mechanism more elegant
2. Cache the entire container path instead of just the mount point path

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                   |   10 +
 include/linux/containerstats.h              |   72 ++++++++++++
 include/linux/delayacct.h                   |   11 +
 kernel/container.c                          |  158 +++++++++++++++++++++++++-
 kernel/cpuset.c                             |    2
 kernel/sched.c                              |    4
 kernel/taskstats.c                          |   70 ++++++++++++
 9 files changed, 349 insertions(+), 6 deletions(-)
```

diff -puN kernel/container.c~containers-taskstats kernel/container.c
--- linux-2.6.21-mm1/kernel/container.c~containers-taskstats 2007-05-11 08:50:29.000000000 +0530
+++ linux-2.6.21-mm1-balbir/kernel/container.c 2007-05-14 22:32:05.000000000 +0530
@@ -59,8 +59,11 @@

```
 #include <asm/uaccess.h>
 #include <asm/atomic.h>
 #include <linux/mutex.h>
+#include <linux/delayacct.h>
+#include <linux/containerstats.h>

 #define CONTAINER_SUPER_MAGIC  0x27e0eb
+#define MAX_MNT_PATH_LEN  256

 /* Generate an array of container subsystem pointers */
 #define SUBSYS(_x) &_x ## _subsys,
@@ -89,6 +92,10 @@ struct containerfs_root {

  /* A list running through the mounted hierarchies */
  struct list_head root_list;
+
+ char *mnt_name;
+ int mnt_name_len;
+ struct vfsmount *mnt;
 };


@@ -574,6 +581,9 @@ static void container_put_super(struct s
  ret = rebind_subsystems(root, 0);
  BUG_ON(ret);

+ kfree(root->mnt_name);
+ root->mnt_name_len = -1;
+ root->mnt = NULL;
  kfree(root);
  mutex_unlock(&container_mutex);
 }
@@ -694,7 +704,8 @@ static int container_fill_super(struct s
  return 0;
 }

-static void init_container_root(struct containerfs_root *root) {
+static void init_container_root(struct containerfs_root *root)
+{
  struct container *cont = &root->top_container;
  INIT_LIST_HEAD(&root->subsys_list);
  root->number_of_containers = 1;
@@ -705,6 +716,44 @@ static void init_container_root(struct c
  list_add(&root->root_list, &roots);
 }

+int container_get_mnt_path(struct containerfs_root *root)
+{
```

```
+ char *start;
+ char *buf = root->mnt_name;
+ int buflen = MAX_MNT_PATH_LEN;
+ struct vfsmount *mnt = root->mnt;
+ struct dentry *dentry;
+ int len;
+
+ if (root->mnt_name_len > 0)
+  return 0;
+
+ start = buf + buflen;
+
+ *--start = '\0';
+ for (;;) {
+  dentry = mnt->mnt_mountpoint;
+  do {
+   if (IS_ROOT(dentry) || (dentry == mnt->mnt_root))
+    break;
+   len = dentry->d_name.len;
+   if ((start -= len) < buf)
+    return -ENAMETOOLONG;
+   memcpy(start, dentry->d_name.name, len);
+   dentry = dentry->d_parent;
+   if (--start < buf)
+    return -ENAMETOOLONG;
+   *start = '/';
+  } while (1);
+  mnt = mnt->mnt_parent;
+  if (mnt == mnt->mnt_parent)
+   break;
+ }
+ memmove(buf, start, buf + buflen - start);
+ root->mnt_name_len = (unsigned long)(buf - start) + buflen - 1;
+ return 0;
+}
+
 static int container_get_sb(struct file_system_type *fs_type,
     int flags, const char *unused_dev_name,
     void *data, struct vfsmount *mnt)
@@ -744,6 +793,12 @@ static int container_get_sb(struct file_
    goto out_unlock;
   }
   init_container_root(root);
+  root->mnt_name = kzalloc(MAX_MNT_PATH_LEN, GFP_KERNEL);
+  if (!root->mnt_name) {
+   ret = -ENOMEM;
+   kfree(root);
+   goto out_unlock;
```

```
+  }
  }

  if (!root->sb) {
@@ -776,6 +831,7 @@ static int container_get_sb(struct file_
   if (!ret)
    atomic_inc(&root->sb->s_active);
  }
+ root->mnt = mnt;

  out_unlock:
  mutex_unlock(&container_mutex);
@@ -803,11 +859,18 @@ static inline struct cftype *__d_cft(str
  * Returns 0 on success, -errno on error.
  */

-int container_path(const struct container *cont, char *buf, int buflen)
+int container_path(const struct container *cont, char *buf, int buflen,
+   bool absolute)
 {
  char *start;
+ int ret;
+ struct containerfs_root *root;
+
+ if (!cont)
+  return -EINVAL;

  start = buf + buflen;
+ root = cont->root;

  *--start = '\0';
  for (;;) {
@@ -824,6 +887,17 @@ int container_path(const struct containe
    return -ENAMETOOLONG;
   *start = '/';
  }
+ if (!absolute)
+  goto copy_out;
+
+ ret = container_get_mnt_path(root);
+ if (ret)
+  return -EINVAL;
+
+ if ((start -= (root->mnt_name_len)) < buf)
+  return -ENAMETOOLONG;
+ memcpy(start, root->mnt_name, root->mnt_name_len);
+copy_out:
  memmove(buf, start, buf + buflen - start);
```

```
  return 0;
 }
@@ -1264,6 +1338,84 @@ array_full:
  return n;
 }

+static inline char* compress_path(char *path)
+{
+ char *tmp;
+ char *start;
+
+ tmp = kmalloc(strlen(path) + 1, GFP_KERNEL);
+ start = tmp;
+ if (!tmp)
+  return NULL;
+
+ while (*path) {
+  while (*path && *path == '/')
+   path++;
+  *tmp++ = '/';
+  while (*path && *path != '/')
+   *tmp++ = *path++;
+ }
+ *tmp++ = '\0';
+ return start;
+}
+
+/*
+ * Build and fill containerstats so that taskstats can export it to user
+ * space
+ */
+int containerstats_build(struct containerstats *stats, char *path)
+{
+ int ret = 0;
+ struct task_struct *g, *p;
+ char *buf;
+ struct container *cont, *root_cont;
+ int subsys_id;
+ struct containerfs_root *root;
+
+ buf = kmalloc(MAX_MNT_PATH_LEN, GFP_KERNEL);
+ if (!buf)
+  return -ENOMEM;
+
+ 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) {
+   cont = task_container(p, subsys_id);
+   ret = container_path(cont, buf, MAX_MNT_PATH_LEN, true);
+   if (ret)
+    goto err;
+   if (strncmp(path, buf, strlen(path)) == 0) {
+    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);
+ }
+err:
+ rcu_read_unlock();
+ kfree(buf);
+ return ret;
+}
+
 static int cmppid(const void *a, const void *b)
 {
  return *(pid_t *)a - *(pid_t *)b;
@@ -1725,7 +1877,7 @@ static int proc_container_show(struct se
   seq_putc(m, ':');
   get_first_subsys(&root->top_container, NULL, &subsys_id);
   cont = task_container(tsk, subsys_id);
-  retval = container_path(cont, buf, PAGE_SIZE);
+  retval = container_path(cont, buf, PAGE_SIZE, false);
   if (retval < 0)
    goto out_unlock;
   seq_puts(m, buf);
```

```
diff -puN include/linux/container.h~containers-taskstats include/linux/container.h
---  linux-2.6.21-mm1/include/linux/container.h~containers-taskst ats 2007-05-11
08:50:29.000000000 +0530
+++ linux-2.6.21-mm1-balbir/include/linux/container.h 2007-05-11 08:50:30.000000000 +0530
@@ -12,6 +12,7 @@
 #include <linux/kref.h>
 #include <linux/cpumask.h>
 #include <linux/nodemask.h>
+#include <linux/containerstats.h>

 #ifdef CONFIG_CONTAINERS

@@ -21,6 +22,7 @@ extern void container_init_smp(void);
 extern void container_fork(struct task_struct *p);
 extern void container_fork_callbacks(struct task_struct *p);
 extern void container_exit(struct task_struct *p, int run_callbacks);
+extern int containerstats_build(struct containerstats *stats, char *path);

 extern struct file_operations proc_container_operations;

@@ -162,7 +164,8 @@ int container_add_files(struct container

 int container_is_removed(const struct container *cont);

-int container_path(const struct container *cont, char *buf, int buflen);
+int container_path(const struct container *cont, char *buf, int buflen,
+   bool absolute);

 int container_task_count(const struct container *cont);

@@ -220,7 +223,8 @@ static inline struct container* task_con
  return task_subsys_state(task, subsys_id)->container;
 }

-int container_path(const struct container *cont, char *buf, int buflen);
+int container_path(const struct container *cont, char *buf, int buflen,
+   bool absolute);

 int container_clone(struct task_struct *tsk, struct container_subsys *ss);

@@ -235,6 +239,8 @@ static inline void container_exit(struct

 static inline void container_lock(void) {}
 static inline void container_unlock(void) {}
+static inline int containerstats_build(struct containerstats *stats,
+     char *path) { return -EINVAL; }

 #endif /* !CONFIG_CONTAINERS */
```

```
diff -puN include/linux/taskstats.h~containers-taskstats include/linux/taskstats.h
diff -puN /dev/null include/linux/containerstats.h
--- /dev/null 2007-02-02 22:51:23.000000000 +0530
+++ linux-2.6.21-mm1-balbir/include/linux/containerstats.h 2007-05-14 13:48:46.000000000
+0530
@@ -0,0 +1,72 @@
+/* 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>
+
+/*
+ * 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 */
+ CONTAINERSTATS_CMD_GET,  /* user->kernel request/get-response */
+ CONTAINERSTATS_CMD_NEW,  /* kernel->user event */
+ __CONTAINERSTATS_CMD_MAX,
+};
+
+#define CONTAINERSTATS_CMD_MAX (__CONTAINERSTATS_CMD_MAX - 1)
+
+enum {
+ CONTAINERSTATS_TYPE_UNSPEC = 0, /* Reserved */
+ CONTAINERSTATS_TYPE_CONTAINER_STATS, /* contains name + stats */
+ __CONTAINERSTATS_TYPE_MAX,
+};
+
+#define CONTAINERSTATS_TYPE_MAX (__CONTAINERSTATS_TYPE_MAX - 1)
+
+enum {
+ CONTAINERSTATS_CMD_ATTR_UNSPEC = 0,
+ CONTAINERSTATS_CMD_ATTR_CONTAINER_NAME,
+ __CONTAINERSTATS_CMD_ATTR_MAX,
+};
+
+#define CONTAINERSTATS_CMD_ATTR_MAX (__CONTAINERSTATS_CMD_ATTR_MAX - 1)
+
+/* NETLINK_GENERIC related info */
+
+#endif /* _LINUX_CONTAINERSTATS_H */
diff -puN kernel/taskstats.c~containers-taskstats kernel/taskstats.c
--- linux-2.6.21-mm1/kernel/taskstats.c~containers-taskstats 2007-05-11 08:50:29.000000000
+0530
+++ linux-2.6.21-mm1-balbir/kernel/taskstats.c 2007-05-14 22:44:16.000000000 +0530
@@ -23,6 +23,8 @@
 #include <linux/tsacct_kern.h>
 #include <linux/cpumask.h>
 #include <linux/percpu.h>
+#include <linux/containerstats.h>
+#include <linux/container.h>
 #include <net/genetlink.h>
 #include <asm/atomic.h>

@@ -31,6 +33,7 @@
  * the TASKSTATS_CMD_ATTR_REGISTER/DEREGISTER_CPUMASK attribute
  */
 #define TASKSTATS_CPUMASK_MAXLEN (100+6*NR_CPUS)
```

```
+#define CONTAINER_STATS_MAX_PATH_LEN 256

 static DEFINE_PER_CPU(__u32, taskstats_seqnum) = { 0 };
 static int family_registered;
@@ -50,6 +53,11 @@ __read_mostly = {
  [TASKSTATS_CMD_ATTR_REGISTER_CPUMASK] = { .type = NLA_STRING },
  [TASKSTATS_CMD_ATTR_DEREGISTER_CPUMASK] = { .type = NLA_STRING },};

+static struct nla_policy
 +containerstats_cmd_get_policy[CONTAINERSTATS_CMD_ATTR_MAX+1 ] __read_mostly = {
+ [CONTAINERSTATS_CMD_ATTR_CONTAINER_NAME] = { .type = NLA_STRING },
+};
+
 struct listener {
  struct list_head list;
  pid_t pid;
@@ -369,6 +377,55 @@ err:
  return NULL;
 }

+static int containerstats_user_cmd(struct sk_buff *skb, struct genl_info *info)
+{
+ int rc = 0;
+ struct sk_buff *rep_skb;
+ struct containerstats *stats;
+ struct nlattr *na;
+ char name[CONTAINER_STATS_MAX_PATH_LEN];
+ int len;
+ size_t size;
+
+ na = info->attrs[CONTAINERSTATS_CMD_ATTR_CONTAINER_NAME];
+ if (!na)
+  return -EINVAL;
+
+ len = nla_len(na);
+ if (len > CONTAINER_STATS_MAX_PATH_LEN)
+  return -E2BIG;
+
+ nla_strlcpy(name, na, len);
+ len = strlen(name);
+
+ /*
+  * Remove trailing '/'s
+  */
+ while (name[len - 1] == '/')
+  len--;
+ name[len] = '\0';
+
```

```
+ size = nla_total_size(sizeof(struct containerstats));
+
+ rc = prepare_reply(info, CONTAINERSTATS_CMD_NEW, &rep_skb, size);
+ if (rc < 0)
+  return rc;
+
+ na = nla_reserve(rep_skb, CONTAINERSTATS_TYPE_CONTAINER_STATS,
+    sizeof(struct containerstats));
+ stats = nla_data(na);
+ memset(stats, 0, sizeof(*stats));
+
+ rc = containerstats_build(stats, name);
+ if (rc < 0)
+  goto err;
+
+ return send_reply(rep_skb, info->snd_pid);
+err:
+ nlmsg_free(rep_skb);
+ return rc;
+}
+
 static int taskstats_user_cmd(struct sk_buff *skb, struct genl_info *info)
 {
  int rc = 0;
@@ -519,6 +576,12 @@ static struct genl_ops taskstats_ops = {
  .policy  = taskstats_cmd_get_policy,
 };

+static struct genl_ops containerstats_ops = {
+ .cmd  = CONTAINERSTATS_CMD_GET,
+ .doit  = containerstats_user_cmd,
+ .policy  = containerstats_cmd_get_policy,
+};
+
 /* Needed early in initialization */
 void __init taskstats_init_early(void)
 {
@@ -543,8 +606,15 @@ static int __init taskstats_init(void)
  if (rc < 0)
   goto err;

+ rc = genl_register_ops(&family, &containerstats_ops);
+ if (rc < 0)
+  goto err_container_ops;
+
  family_registered = 1;
+ printk("registered taskstats version %d\n", TASKSTATS_GENL_VERSION);
  return 0;
```

```
+err_container_ops:
+ genl_unregister_ops(&family, &taskstats_ops);
 err:
  genl_unregister_family(&family);
  return rc;
diff -puN include/linux/sched.h~containers-taskstats include/linux/sched.h
diff -puN kernel/sched.c~containers-taskstats kernel/sched.c
--- linux-2.6.21-mm1/kernel/sched.c~containers-taskstats 2007-05-11 08:50:30.000000000 +0530
+++ linux-2.6.21-mm1-balbir/kernel/sched.c 2007-05-11 08:50:30.000000000 +0530
@@ -4817,11 +4817,13 @@ void __sched io_schedule(void)
 {
  struct rq *rq = &__raw_get_cpu_var(runqueues);

+ delayacct_set_flag(DELAYACCT_PF_BLKIO);
  delayacct_blkio_start();
  atomic_inc(&rq->nr_iowait);
  schedule();
  atomic_dec(&rq->nr_iowait);
  delayacct_blkio_end();
+ delayacct_clear_flag(DELAYACCT_PF_BLKIO);
 }
 EXPORT_SYMBOL(io_schedule);

@@ -4830,11 +4832,13 @@ long __sched io_schedule_timeout(long ti
  struct rq *rq = &__raw_get_cpu_var(runqueues);
  long ret;

+ delayacct_set_flag(DELAYACCT_PF_BLKIO);
  delayacct_blkio_start();
  atomic_inc(&rq->nr_iowait);
  ret = schedule_timeout(timeout);
  atomic_dec(&rq->nr_iowait);
  delayacct_blkio_end();
+ delayacct_clear_flag(DELAYACCT_PF_BLKIO);
  return ret;
 }

diff -puN include/linux/delayacct.h~containers-taskstats include/linux/delayacct.h
---  linux-2.6.21-mm1/include/linux/delayacct.h~containers-taskst ats 2007-05-11
08:50:30.000000000 +0530
+++ linux-2.6.21-mm1-balbir/include/linux/delayacct.h 2007-05-11 08:56:49.000000000 +0530
@@ -26,6 +26,7 @@
  * Used to set current->delays->flags
  */
 #define DELAYACCT_PF_SWAPIN 0x00000001 /* I am doing a swapin */
+#define DELAYACCT_PF_BLKIO 0x00000002 /* I am waiting on IO */

 #ifdef CONFIG_TASK_DELAY_ACCT
```

```
@@ -39,6 +40,14 @@ extern void __delayacct_blkio_end(void);
 extern int __delayacct_add_tsk(struct taskstats *, struct task_struct *);
 extern __u64 __delayacct_blkio_ticks(struct task_struct *);

+static inline int delayacct_is_task_waiting_on_io(struct task_struct *p)
+{
+ if (p->delays)
+  return (p->delays->flags & DELAYACCT_PF_BLKIO);
+ else
+  return 0;
+}
+
 static inline void delayacct_set_flag(int flag)
 {
  if (current->delays)
@@ -116,6 +125,8 @@ static inline int delayacct_add_tsk(stru
 { return 0; }
 static inline __u64 delayacct_blkio_ticks(struct task_struct *tsk)
 { return 0; }
+static inline int delayacct_is_task_waiting_on_io(struct task_struct *p)
+{ return 0; }
 #endif /* CONFIG_TASK_DELAY_ACCT */

 #endif
diff -puN kernel/containerstats.c~containers-taskstats kernel/containerstats.c
diff -puN include/linux/taskstats_kern.h~containers-taskstats include/linux/taskstats_kern.h
diff -puN init/Kconfig~containers-taskstats init/Kconfig
diff -puN kernel/cpuset.c~containers-taskstats kernel/cpuset.c
--- linux-2.6.21-mm1/kernel/cpuset.c~containers-taskstats 2007-05-14 11:31:37.000000000 +0530
+++ linux-2.6.21-mm1-balbir/kernel/cpuset.c 2007-05-14 11:31:46.000000000 +0530
@@ -1788,7 +1788,7 @@ static int proc_cpuset_show(struct seq_f
  retval = -EINVAL;
  container_lock();
  css = task_subsys_state(tsk, cpuset_subsys_id);
- retval = container_path(css->container, buf, PAGE_SIZE);
+ retval = container_path(css->container, buf, PAGE_SIZE, false);
  if (retval < 0)
   goto out_unlock;
  seq_puts(m, buf);
diff -puN include/linux/Kbuild~containers-taskstats include/linux/Kbuild
--- linux-2.6.21-mm1/include/linux/Kbuild~containers-taskstats 2007-05-14 13:52:53.000000000
+0530
+++ linux-2.6.21-mm1-balbir/include/linux/Kbuild 2007-05-14 13:53:14.000000000 +0530
@@ -47,6 +47,7 @@ header-y += coff.h
 header-y += comstats.h
 header-y += consolemap.h
 header-y += const.h
```

```
+header-y += containerstats.h
 header-y += cycx_cfm.h
 header-y += dlm_device.h
 header-y += dm-ioctl.h
diff -puN /dev/null Documentation/accounting/containerstats.txt
--- /dev/null 2007-02-02 22:51:23.000000000 +0530
+++  linux-2.6.21-mm1-balbir/Documentation/accounting/containerst ats.txt 2007-05-14
23:11:35.000000000 +0530
@@ -0,0 +1,27 @@
+Containerstats 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.
+
+Per container statistics infrastructure re-uses code from the taskstats
+interface. A new set of container operations are registered with commands
+and attributes specific to containers. 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 path, if
+a container is found to belong to the specified hierarchy, then statistics
+about the state of all the tasks in the container is returned to user space.
+
+NOTE: We currently rely on delay accounting for extracting information
+about tasks blocked on I/O. If CONFIG_TASK_DELAY_ACCT is disabled, this
+information will not be available.
+
+To extract container statistics a utility very similar to getdelays.c
+has been developed, the sample output of the utility is shown below
+
+~/balbir/containerstats # ./containerstats  -C "/container/a"
+sleeping 1, blocked 0, running 1, stopped 0, uninterruptible 0
+~/balbir/containerstats # ./containerstats  -C "/container"
+sleeping 155, blocked 0, running 1, stopped 0, uninterruptible 2

_

--
 Warm Regards,
 Balbir Singh
 Linux Technology Center
 IBM, ISTL
```

Subject: Re: [RFC][PATCH] Per container statistics
Posted by dev on Tue, 15 May 2007 07:12:12 GMT

Balbir Singh wrote:
> 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 path, if
> a container is found to belong to the specified hierarchy, then statistics
> about the state of all the tasks in the container is returned to user space.
>
> TODO's
>
> 1. All data is shared as filesystem paths, it involves a lot of string
>   and path name handling. Simplify and make the mechanism more elegant
> 2. Cache the entire container path instead of just the mount point path
>
> 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 | 10 +
> include/linux/containerstats.h | 72 ++++++++++++
> include/linux/delayacct.h | 11 +
> kernel/container.c | 158 ++++++++++++++++++++++++++++-
> kernel/cpuset.c | 2
> kernel/sched.c | 4
> kernel/taskstats.c | 70 ++++++++++++
> 9 files changed, 349 insertions(+), 6 deletions(-)
>
> diff -puN kernel/container.c~containers-taskstats kernel/container.c
> --- linux-2.6.21-mm1/kernel/container.c~containers-taskstats 2007-05-11 08:50:29.000000000
+0530
> +++ linux-2.6.21-mm1-balbir/kernel/container.c 2007-05-14 22:32:05.000000000 +0530
> @@ -59,8 +59,11 @@
> #include <asm/uaccess.h>
> #include <asm/atomic.h>

> #include <linux/mutex.h>
> +#include <linux/delayacct.h>
> +#include <linux/containerstats.h>
>
> #define CONTAINER_SUPER_MAGIC 0x27e0eb
> +#define MAX_MNT_PATH_LEN 256
>
> /* Generate an array of container subsystem pointers */
> #define SUBSYS(_x) &_x ## _subsys,
> @@ -89,6 +92,10 @@ struct containerfs_root {
>
>   /* A list running through the mounted hierarchies */
>   struct list_head root_list;
> +
> + char *mnt_name;
> + int mnt_name_len;
> + struct vfsmount *mnt;
> };
>
>
> @@ -574,6 +581,9 @@ static void container_put_super(struct s
>   ret = rebind_subsystems(root, 0);
>   BUG_ON(ret);
>
> + kfree(root->mnt_name);
> + root->mnt_name_len = -1;
> + root->mnt = NULL;
>   kfree(root);
>   mutex_unlock(&container_mutex);
> }
> @@ -694,7 +704,8 @@ static int container_fill_super(struct s
>   return 0;
> }
>
> -static void init_container_root(struct containerfs_root *root) {
> +static void init_container_root(struct containerfs_root *root)
> +{
>   struct container *cont = &root->top_container;
>   INIT_LIST_HEAD(&root->subsys_list);
>   root->number_of_containers = 1;
> @@ -705,6 +716,44 @@ static void init_container_root(struct c
>   list_add(&root->root_list, &roots);
> }
>
> +int container_get_mnt_path(struct containerfs_root *root)
> +{
> + char *start;
> + char *buf = root->mnt_name;

```
> + int buflen = MAX_MNT_PATH_LEN;
> + struct vfsmount *mnt = root->mnt;
> + struct dentry *dentry;
> + int len;
> +
> + if (root->mnt_name_len > 0)
> +  return 0;
> +
> + start = buf + buflen;
> +
> + *--start = '\0';
> + for (;;) {
> +  dentry = mnt->mnt_mountpoint;
> +  do {
> +   if (IS_ROOT(dentry) || (dentry == mnt->mnt_root))
> +    break;
> +   len = dentry->d_name.len;
> +   if ((start -= len) < buf)
> +    return -ENAMETOOLONG;
> +   memcpy(start, dentry->d_name.name, len);
> +   dentry = dentry->d_parent;
> +   if (--start < buf)
> +    return -ENAMETOOLONG;
> +   *start = '/';
> +  } while (1);
> +  mnt = mnt->mnt_parent;
> +  if (mnt == mnt->mnt_parent)
> +   break;
> + }
> + memmove(buf, start, buf + buflen - start);
> + root->mnt_name_len = (unsigned long)(buf - start) + buflen - 1;
> + return 0;
> +}
> +
>  static int container_get_sb(struct file_system_type *fs_type,
>      int flags, const char *unused_dev_name,
>      void *data, struct vfsmount *mnt)
> @@ -744,6 +793,12 @@ static int container_get_sb(struct file_
>     goto out_unlock;
>    }
>    init_container_root(root);
> +  root->mnt_name = kzalloc(MAX_MNT_PATH_LEN, GFP_KERNEL);
> +  if (!root->mnt_name) {
> +   ret = -ENOMEM;
> +   kfree(root);
> +   goto out_unlock;
> +  }
>    }
```

```
>
>   if (!root->sb) {
> @@ -776,6 +831,7 @@ static int container_get_sb(struct file_
>     if (!ret)
>       atomic_inc(&root->sb->s_active);
>   }
> + root->mnt = mnt;
>
>   out_unlock:
>   mutex_unlock(&container_mutex);
> @@ -803,11 +859,18 @@ static inline struct cftype *__d_cft(str
>   * Returns 0 on success, -errno on error.
>   */
>
> -int container_path(const struct container *cont, char *buf, int buflen)
> +int container_path(const struct container *cont, char *buf, int buflen,
> +    bool absolute)
>  {
>   char *start;
> + int ret;
> + struct containerfs_root *root;
> +
> + if (!cont)
> +  return -EINVAL;
>
>   start = buf + buflen;
> + root = cont->root;
>
>   *--start = '\0';
>   for (;;) {
> @@ -824,6 +887,17 @@ int container_path(const struct containe
>       return -ENAMETOOLONG;
>     *start = '/';
>   }
> + if (!absolute)
> +  goto copy_out;
> +
> + ret = container_get_mnt_path(root);
> + if (ret)
> +  return -EINVAL;
> +
> + if ((start -= (root->mnt_name_len)) < buf)
> +  return -ENAMETOOLONG;
> + memcpy(start, root->mnt_name, root->mnt_name_len);
> +copy_out:
>   memmove(buf, start, buf + buflen - start);
>   return 0;
>  }
```

```
> @@ -1264,6 +1338,84 @@ array_full:
>   return n;
>  }
>
> +static inline char* compress_path(char *path)
> +{
> + char *tmp;
> + char *start;
> +
> + tmp = kmalloc(strlen(path) + 1, GFP_KERNEL);
> + start = tmp;
> + if (!tmp)
> +  return NULL;
> +
> + while (*path) {
> +  while (*path && *path == '/')
> +   path++;
> +  *tmp++ = '/';
> +  while (*path && *path != '/')
> +   *tmp++ = *path++;
> + }
> + *tmp++ = '\0';
> + return start;
> +}
> +
> +/*
> + * Build and fill containerstats so that taskstats can export it to user
> + * space
> + */
> +int containerstats_build(struct containerstats *stats, char *path)
> +{
> + int ret = 0;
> + struct task_struct *g, *p;
> + char *buf;
> + struct container *cont, *root_cont;
> + int subsys_id;
> + struct containerfs_root *root;
> +
> + buf = kmalloc(MAX_MNT_PATH_LEN, GFP_KERNEL);
> + if (!buf)
> +  return -ENOMEM;
> +
> + 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) {
> +   cont = task_container(p, subsys_id);
> +   ret = container_path(cont, buf, MAX_MNT_PATH_LEN, true);
> +   if (ret)
> +    goto err;
> +   if (strncmp(path, buf, strlen(path)) == 0) {
> +    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);
```

oh, please no... Andrew, this loop can be very very long when having > 10,000 tasks on the machine...
we have had enough such issues in OpenVZ and just don't want to come through this again.
Also sum of RUNNING + UNINTERRUPTIBLE is required for per-container loadavg calculations.

```
> + }
> +err:
> + rcu_read_unlock();
> + kfree(buf);
> + return ret;
> +}
> +
>  static int cmppid(const void *a, const void *b)
>  {
>   return *(pid_t *)a - *(pid_t *)b;
> @@ -1725,7 +1877,7 @@ static int proc_container_show(struct se
>    seq_putc(m, ':');
>    get_first_subsys(&root->top_container, NULL, &subsys_id);
>    cont = task_container(tsk, subsys_id);
> -  retval = container_path(cont, buf, PAGE_SIZE);
```

> + retval = container_path(cont, buf, PAGE_SIZE, false);
>   if (retval < 0)
>   goto out_unlock;
>   seq_puts(m, buf);
> diff -puN include/linux/container.h~containers-taskstats include/linux/container.h
> --- linux-2.6.21-mm1/include/linux/container.h~containers-taskst ats 2007-05-11
08:50:29.000000000 +0530
> +++ linux-2.6.21-mm1-balbir/include/linux/container.h 2007-05-11 08:50:30.000000000 +0530
> @@ -12,6 +12,7 @@
> #include <linux/kref.h>
> #include <linux/cpumask.h>
> #include <linux/nodemask.h>
> +#include <linux/containerstats.h>
>
> #ifdef CONFIG_CONTAINERS
>
> @@ -21,6 +22,7 @@ extern void container_init_smp(void);
> extern void container_fork(struct task_struct *p);
> extern void container_fork_callbacks(struct task_struct *p);
> extern void container_exit(struct task_struct *p, int run_callbacks);
> +extern int containerstats_build(struct containerstats *stats, char *path);
>
> extern struct file_operations proc_container_operations;
>
> @@ -162,7 +164,8 @@ int container_add_files(struct container
>
> int container_is_removed(const struct container *cont);
>
> -int container_path(const struct container *cont, char *buf, int buflen);
> +int container_path(const struct container *cont, char *buf, int buflen,
> +   bool absolute);
>
> int container_task_count(const struct container *cont);
>
> @@ -220,7 +223,8 @@ static inline struct container* task_con
>   return task_subsys_state(task, subsys_id)->container;
> }
>
> -int container_path(const struct container *cont, char *buf, int buflen);
> +int container_path(const struct container *cont, char *buf, int buflen,
> +   bool absolute);
>
> int container_clone(struct task_struct *tsk, struct container_subsys *ss);
>
> @@ -235,6 +239,8 @@ static inline void container_exit(struct
>
> static inline void container_lock(void) {}
> static inline void container_unlock(void) {}

> +static inline int containerstats_build(struct containerstats *stats,
> +   char *path) { return -EINVAL; }
>
> #endif /* !CONFIG_CONTAINERS */
>
> diff -puN include/linux/taskstats.h~containers-taskstats include/linux/taskstats.h
> diff -puN /dev/null include/linux/containerstats.h
> --- /dev/null 2007-02-02 22:51:23.000000000 +0530
> +++ linux-2.6.21-mm1-balbir/include/linux/containerstats.h 2007-05-14 13:48:46.000000000
+0530
> @@ -0,0 +1,72 @@
> +/* 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>
> +
> +/*
> + * 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 */
> + CONTAINERSTATS_CMD_GET, /* user->kernel request/get-response */
> + CONTAINERSTATS_CMD_NEW, /* kernel->user event */
> + __CONTAINERSTATS_CMD_MAX,
> +};
> +
> +#define CONTAINERSTATS_CMD_MAX (__CONTAINERSTATS_CMD_MAX - 1)
> +
> +enum {
> + CONTAINERSTATS_TYPE_UNSPEC = 0, /* Reserved */
> + CONTAINERSTATS_TYPE_CONTAINER_STATS, /* contains name + stats */
> + __CONTAINERSTATS_TYPE_MAX,
> +};
> +
> +#define CONTAINERSTATS_TYPE_MAX (__CONTAINERSTATS_TYPE_MAX - 1)
> +
> +enum {
> + CONTAINERSTATS_CMD_ATTR_UNSPEC = 0,
> + CONTAINERSTATS_CMD_ATTR_CONTAINER_NAME,
> + __CONTAINERSTATS_CMD_ATTR_MAX,
> +};
> +
> +#define CONTAINERSTATS_CMD_ATTR_MAX (__CONTAINERSTATS_CMD_ATTR_MAX - 1)
> +
> +/* NETLINK_GENERIC related info */
> +
> +#endif /* _LINUX_CONTAINERSTATS_H */
> diff -puN kernel/taskstats.c~containers-taskstats kernel/taskstats.c
> --- linux-2.6.21-mm1/kernel/taskstats.c~containers-taskstats 2007-05-11 08:50:29.000000000 +0530
> +++ linux-2.6.21-mm1-balbir/kernel/taskstats.c 2007-05-14 22:44:16.000000000 +0530
> @@ -23,6 +23,8 @@
> #include <linux/tsacct_kern.h>
> #include <linux/cpumask.h>
> #include <linux/percpu.h>
> +#include <linux/containerstats.h>
> +#include <linux/container.h>
> #include <net/genetlink.h>
> #include <asm/atomic.h>

```
>
> @@ -31,6 +33,7 @@
>   * the TASKSTATS_CMD_ATTR_REGISTER/DEREGISTER_CPUMASK attribute
>   */
>  #define TASKSTATS_CPUMASK_MAXLEN (100+6*NR_CPUS)
> +#define CONTAINER_STATS_MAX_PATH_LEN 256
>
>  static DEFINE_PER_CPU(__u32, taskstats_seqnum) = { 0 };
>  static int family_registered;
> @@ -50,6 +53,11 @@ __read_mostly = {
>   [TASKSTATS_CMD_ATTR_REGISTER_CPUMASK] = { .type = NLA_STRING },
>   [TASKSTATS_CMD_ATTR_DEREGISTER_CPUMASK] = { .type = NLA_STRING },};
>
> +static struct nla_policy
>  +containerstats_cmd_get_policy[CONTAINERSTATS_CMD_ATTR_MAX+1 ] __read_mostly =
{
> + [CONTAINERSTATS_CMD_ATTR_CONTAINER_NAME] = { .type = NLA_STRING },
> +};
> +
>  struct listener {
>   struct list_head list;
>   pid_t pid;
> @@ -369,6 +377,55 @@ err:
>   return NULL;
>  }
>
> +static int containerstats_user_cmd(struct sk_buff *skb, struct genl_info *info)
> +{
> + int rc = 0;
> + struct sk_buff *rep_skb;
> + struct containerstats *stats;
> + struct nlattr *na;
> + char name[CONTAINER_STATS_MAX_PATH_LEN];
> + int len;
> + size_t size;
> +
> + na = info->attrs[CONTAINERSTATS_CMD_ATTR_CONTAINER_NAME];
> + if (!na)
> +  return -EINVAL;
> +
> + len = nla_len(na);
> + if (len > CONTAINER_STATS_MAX_PATH_LEN)
> +  return -E2BIG;
> +
> + nla_strlcpy(name, na, len);
> + len = strlen(name);
> +
> + /*
```

```
> +  * Remove trailing '/'s
> +  */
> + while (name[len - 1] == '/')
> +  len--;
> + name[len] = '\0';
> +
> + size = nla_total_size(sizeof(struct containerstats));
> +
> + rc = prepare_reply(info, CONTAINERSTATS_CMD_NEW, &rep_skb, size);
> + if (rc < 0)
> +  return rc;
> +
> + na = nla_reserve(rep_skb, CONTAINERSTATS_TYPE_CONTAINER_STATS,
> +    sizeof(struct containerstats));
> + stats = nla_data(na);
> + memset(stats, 0, sizeof(*stats));
> +
> + rc = containerstats_build(stats, name);
> + if (rc < 0)
> +  goto err;
> +
> + return send_reply(rep_skb, info->snd_pid);
> +err:
> + nlmsg_free(rep_skb);
> + return rc;
> +}
> +
>  static int taskstats_user_cmd(struct sk_buff *skb, struct genl_info *info)
>  {
>   int rc = 0;
> @@ -519,6 +576,12 @@ static struct genl_ops taskstats_ops = {
>   .policy  = taskstats_cmd_get_policy,
>  };
>
> +static struct genl_ops containerstats_ops = {
> + .cmd  = CONTAINERSTATS_CMD_GET,
> + .doit  = containerstats_user_cmd,
> + .policy  = containerstats_cmd_get_policy,
> +};
> +
>  /* Needed early in initialization */
>  void __init taskstats_init_early(void)
>  {
> @@ -543,8 +606,15 @@ static int __init taskstats_init(void)
>   if (rc < 0)
>    goto err;
>
> + rc = genl_register_ops(&family, &containerstats_ops);
```

> + if (rc < 0)
> + goto err_container_ops;
> +
>   family_registered = 1;
> + printk("registered taskstats version %d\n", TASKSTATS_GENL_VERSION);
>   return 0;
> +err_container_ops:
> + genl_unregister_ops(&family, &taskstats_ops);
> err:
>   genl_unregister_family(&family);
>   return rc;
> diff -puN include/linux/sched.h~containers-taskstats include/linux/sched.h
> diff -puN kernel/sched.c~containers-taskstats kernel/sched.c
> --- linux-2.6.21-mm1/kernel/sched.c~containers-taskstats 2007-05-11 08:50:30.000000000
+0530
> +++ linux-2.6.21-mm1-balbir/kernel/sched.c 2007-05-11 08:50:30.000000000 +0530
> @@ -4817,11 +4817,13 @@ void __sched io_schedule(void)
> {
>   struct rq *rq = &__raw_get_cpu_var(runqueues);
>
> + delayacct_set_flag(DELAYACCT_PF_BLKIO);
>   delayacct_blkio_start();
>   atomic_inc(&rq->nr_iowait);
>   schedule();
>   atomic_dec(&rq->nr_iowait);
>   delayacct_blkio_end();
> + delayacct_clear_flag(DELAYACCT_PF_BLKIO);
> }
> EXPORT_SYMBOL(io_schedule);
>
> @@ -4830,11 +4832,13 @@ long __sched io_schedule_timeout(long ti
>   struct rq *rq = &__raw_get_cpu_var(runqueues);
>   long ret;
>
> + delayacct_set_flag(DELAYACCT_PF_BLKIO);
>   delayacct_blkio_start();
>   atomic_inc(&rq->nr_iowait);
>   ret = schedule_timeout(timeout);
>   atomic_dec(&rq->nr_iowait);
>   delayacct_blkio_end();
> + delayacct_clear_flag(DELAYACCT_PF_BLKIO);
>   return ret;
> }
>
> diff -puN include/linux/delayacct.h~containers-taskstats include/linux/delayacct.h
> --- linux-2.6.21-mm1/include/linux/delayacct.h~containers-taskst ats 2007-05-11
08:50:30.000000000 +0530
> +++ linux-2.6.21-mm1-balbir/include/linux/delayacct.h 2007-05-11 08:56:49.000000000 +0530

```
> @@ -26,6 +26,7 @@
>   * Used to set current->delays->flags
>   */
>  #define DELAYACCT_PF_SWAPIN 0x00000001 /* I am doing a swapin */
> +#define DELAYACCT_PF_BLKIO 0x00000002 /* I am waiting on IO */
>
>  #ifdef CONFIG_TASK_DELAY_ACCT
>
> @@ -39,6 +40,14 @@ extern void __delayacct_blkio_end(void);
>  extern int __delayacct_add_tsk(struct taskstats *, struct task_struct *);
>  extern __u64 __delayacct_blkio_ticks(struct task_struct *);
>
> +static inline int delayacct_is_task_waiting_on_io(struct task_struct *p)
> +{
> + if (p->delays)
> +  return (p->delays->flags & DELAYACCT_PF_BLKIO);
> + else
> +  return 0;
> +}
> +
>  static inline void delayacct_set_flag(int flag)
>  {
>   if (current->delays)
> @@ -116,6 +125,8 @@ static inline int delayacct_add_tsk(stru
>  { return 0; }
>  static inline __u64 delayacct_blkio_ticks(struct task_struct *tsk)
>  { return 0; }
> +static inline int delayacct_is_task_waiting_on_io(struct task_struct *p)
> +{ return 0; }
>  #endif /* CONFIG_TASK_DELAY_ACCT */
>
>  #endif
> diff -puN kernel/containerstats.c~containers-taskstats kernel/containerstats.c
> diff -puN include/linux/taskstats_kern.h~containers-taskstats include/linux/taskstats_kern.h
> diff -puN init/Kconfig~containers-taskstats init/Kconfig
> diff -puN kernel/cpuset.c~containers-taskstats kernel/cpuset.c
> --- linux-2.6.21-mm1/kernel/cpuset.c~containers-taskstats 2007-05-14 11:31:37.000000000
+0530
> +++ linux-2.6.21-mm1-balbir/kernel/cpuset.c 2007-05-14 11:31:46.000000000 +0530
> @@ -1788,7 +1788,7 @@ static int proc_cpuset_show(struct seq_f
>   retval = -EINVAL;
>   container_lock();
>   css = task_subsys_state(tsk, cpuset_subsys_id);
> - retval = container_path(css->container, buf, PAGE_SIZE);
> + retval = container_path(css->container, buf, PAGE_SIZE, false);
>   if (retval < 0)
>    goto out_unlock;
>   seq_puts(m, buf);
```

> diff -puN include/linux/Kbuild~containers-taskstats include/linux/Kbuild
> --- linux-2.6.21-mm1/include/linux/Kbuild~containers-taskstats 2007-05-14 13:52:53.000000000 +0530
> +++ linux-2.6.21-mm1-balbir/include/linux/Kbuild 2007-05-14 13:53:14.000000000 +0530
> @@ -47,6 +47,7 @@ header-y += coff.h
> header-y += comstats.h
> header-y += consolemap.h
> header-y += const.h
> +header-y += containerstats.h
> header-y += cycx_cfm.h
> header-y += dlm_device.h
> header-y += dm-ioctl.h
> diff -puN /dev/null Documentation/accounting/containerstats.txt
> --- /dev/null 2007-02-02 22:51:23.000000000 +0530
> +++ linux-2.6.21-mm1-balbir/Documentation/accounting/containerst ats.txt 2007-05-14 23:11:35.000000000 +0530
> @@ -0,0 +1,27 @@
> +Containerstats 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.
> +
> +Per container statistics infrastructure re-uses code from the taskstats
> +interface. A new set of container operations are registered with commands
> +and attributes specific to containers. 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 path, if
> +a container is found to belong to the specified hierarchy, then statistics
> +about the state of all the tasks in the container is returned to user space.
> +
> +NOTE: We currently rely on delay accounting for extracting information
> +about tasks blocked on I/O. If CONFIG_TASK_DELAY_ACCT is disabled, this
> +information will not be available.
> +
> +To extract container statistics a utility very similar to getdelays.c
> +has been developed, the sample output of the utility is shown below
> +
> +~/balbir/containerstats # ./containerstats -C "/container/a"
> +sleeping 1, blocked 0, running 1, stopped 0, uninterruptible 0
> +~/balbir/containerstats # ./containerstats -C "/container"
> +sleeping 155, blocked 0, running 1, stopped 0, uninterruptible 2
> _
>

Subject: Re: [RFC][PATCH] Per container statistics
Posted by Balbir Singh on Tue, 15 May 2007 07:23:00 GMT
View Forum Message <> Reply to Message

Kirill Korotaev wrote:
> oh, please no... Andrew, this loop can be very very long when having > 10,000 tasks on the machine...
> we have had enough such issues in OpenVZ and just don't want to come through this again.
> Also sum of RUNNING + UNINTERRUPTIBLE is required for per-container loadavg calculations.
>

Hi, Kirill,

I've asked Paul Menage to help us with this scenario by adding a per-container task list. Would you be ok, if we walked the per-container task list. We could further optimize the routine by adding

1. A container id (avoid parsing file paths to uniquely identify a container)
2. If we cannot do (1), then cache container paths

The other option is of-course what Pavel posted, but I think the amortized cost of tracking task state at the time of task state change or looping over tasks on demand should be the same.

Do you agree in general with containerstats? How we build (walk the task list or at the time of task state change) is flexible with containerstats.

Thanks for your comments!

--
Warm Regards,
Balbir Singh
Linux Technology Center
IBM, ISTL

Subject: Re: [RFC][PATCH] Per container statistics
Posted by Paul Menage on Mon, 21 May 2007 20:56:31 GMT
View Forum Message <> Reply to Message

Hi Balbir,

On 5/14/07, Balbir Singh <balbir@linux.vnet.ibm.com> wrote:
>
> 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.

Sorry for the delay in looking at this. (I've been travelling a bit).

The basic idea of being able to get stats on a per-container basis
seems good, but I've got some suggestions on the API/implementation:

- saving a mount pointer in the containerfsroot structure won't work
because a hierarchy (superblock) can be mounted in more than one
place, or even in zero places (if you unmount a hierarchy with active
containers, the superblock and the containers stay active). Also it
might be possible to move a mounted container hierarchy via mount
--move, although I've not actually tried that.

- a cleaner way to pass in a container id would be to pass a file
descriptor on a container directory. The dentry associated with this
fd would unambiguously identify the hierarchy and the container, so
then even if we didn't maintain a per-container task likst, the
for_each_thread() loop would involve just a single comparison per task
to see if the task was in the desired container.

- if we're trying to integrate this with taskstats, then it would be
nice to support retrieving all the other taskstats values (where they
make sense) on a per-container basis (in the same way that we can
currently request them on a per-thread or a per-process basis). i.e.
don't create a new container_taskstats structure, but instead augment
the current taskstats structure, and allow the user to retrieve the
aggregate of that for the entire container. Similarly, being able to
get taskstats notifications based on the container memberships of a
task might be nice.

Paul

Subject: Re: [RFC][PATCH] Per container statistics
Posted by Balbir Singh on Thu, 24 May 2007 15:59:36 GMT
View Forum Message <> Reply to Message

Paul Menage wrote:
> Hi Balbir,
>
> On 5/14/07, Balbir Singh <balbir@linux.vnet.ibm.com> wrote:
>>
>> 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.
>
> Sorry for the delay in looking at this. (I've been travelling a bit).
>
> The basic idea of being able to get stats on a per-container basis
> seems good, but I've got some suggestions on the API/implementation:
>
> - saving a mount pointer in the containerfsroot structure won't work
> because a hierarchy (superblock) can be mounted in more than one
> place, or even in zero places (if you unmount a hierarchy with active
> containers, the superblock and the containers stay active). Also it
> might be possible to move a mounted container hierarchy via mount
> --move, although I've not actually tried that.

Yes, Good catch!

>
> - a cleaner way to pass in a container id would be to pass a file
> descriptor on a container directory. The dentry associated with this
> fd would unambiguously identify the hierarchy and the container, so
> then even if we didn't maintain a per-container task likst, the
> for_each_thread() loop would involve just a single comparison per task
> to see if the task was in the desired container.

I thought about this approach, but did not implement the code this way
because a system could have thousands of containers and expecting a
statistics application to open a file descriptor each time for each
container will turn out to be an expensive operation

overhead = 2 syscalls (open + close) * number of containers * frequency
of stats collections

I guess for now this is a good choice

>
> - if we're trying to integrate this with taskstats, then it would be
> nice to support retrieving all the other taskstats values (where they
> make sense) on a per-container basis (in the same way that we can
> currently request them on a per-thread or a per-process basis). i.e.
> don't create a new container_taskstats structure, but instead augment
> the current taskstats structure, and allow the user to retrieve the
> aggregate of that for the entire container. Similarly, being able to
> get taskstats notifications based on the container memberships of a
> task might be nice.
>

The reason why I did not do so (augmenting container stats) is that
we send out an event on every exit and that might be heavy for the

container. The current implementation does not support the push
model (where data is pushed from the kernel), but that should be
easy to support for container events (as and when we find a useful
event to support).

> Paul

--
Warm Regards,
Balbir Singh
Linux Technology Center
IBM, ISTL

Subject: Re: [RFC][PATCH] Per container statistics
Posted by Paul Menage on Thu, 24 May 2007 16:12:41 GMT
View Forum Message <> Reply to Message

On 5/24/07, Balbir Singh <balbir@linux.vnet.ibm.com> wrote:
>
> I thought about this approach, but did not implement the code this way
> because a system could have thousands of containers and expecting a
> statistics application to open a file descriptor each time for each
> container will turn out to be an expensive operation
>
> overhead = 2 syscalls (open + close) * number of containers * frequency
>   of stats collections

But you can cache the file handles open between stats calls, assuming
you're doing them repeatedly.

Paul
