Subject: [PATCH 14/29] add containerstats v3
Posted by Paul Menage on Tue, 11 Sep 2007 19:52:53 GMT
View Forum Message <> Reply to Message

From: Balbir Singh <balbir@linux.vnet.ibm.com>

This patch is inspired by the discussion at
http://lkml.org/lkml/2007/4/11/187 and implements per cgroup 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 cgroups v9 patches (forward
ported)

This patch implements per cgroup statistics infrastructure and re-uses
code from the taskstats interface. A new set of cgroup operations are
registered with commands and attributes. It should be very easy to
extend per cgroup statistics, by adding members to the cgroupstats
structure.

The current model for cgroupstats 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 cgroup file
descriptor. Statistics about the state of all the tasks in the cgroup
is returned to user space.

TODO's/NOTE:

This patch provides an infrastructure for implementing cgroup 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 cgroup statistics in the future.

Sample output

./cgroupstats -C /cgroup/a
sleeping 2, blocked 0, running 1, stopped 0, uninterruptible 0

./cgroupstats -C /cgroup/
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>
Cc: Paul Menage <menage@google.com>
Cc: Jay Lan <jlan@engr.sgi.com>
Signed-off-by: Andrew Morton <akpm@linux-foundation.org>

```
---

 Documentation/accounting/cgroupstats.txt |   27 ++++++
 include/linux/Kbuild                     |    1
 include/linux/cgroup.h                   |    8 ++
 include/linux/cgroupstats.h              |   70 ++++++++++++++++++++
 include/linux/delayacct.h                |   13 +++
 kernel/cgroup.c                          |   55 ++++++++++++++++
 kernel/taskstats.c                       |   66 ++++++++++++++++++++
 7 files changed, 240 insertions(+)

diff -puN /dev/null Documentation/accounting/cgroupstats.txt
--- /dev/null
+++ a/Documentation/accounting/cgroupstats.txt
@@ -0,0 +1,27 @@
+Control Groupstats is inspired by the discussion at
+http://lkml.org/lkml/2007/4/11/187 and implements per cgroup statistics as
+suggested by Andrew Morton in http://lkml.org/lkml/2007/4/11/263.
+
+Per cgroup statistics infrastructure re-uses code from the taskstats
+interface. A new set of cgroup operations are registered with commands
+and attributes specific to cgroups. It should be very easy to
+extend per cgroup statistics, by adding members to the cgroupstats
+structure.
+
+The current model for cgroupstats 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 cgroup path.
+Statistics about the state of all the tasks in the cgroup 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 cgroup statistics a utility very similar to getdelays.c
+has been developed, the sample output of the utility is shown below
+
+~/balbir/cgroupstats # ./getdelays  -C "/cgroup/a"
+sleeping 1, blocked 0, running 1, stopped 0, uninterruptible 0
+~/balbir/cgroupstats # ./getdelays  -C "/cgroup"
+sleeping 155, blocked 0, running 1, stopped 0, uninterruptible 2
diff -puN include/linux/Kbuild~add-cgroupstats-v3 include/linux/Kbuild
--- a/include/linux/Kbuild~add-cgroupstats-v3
+++ a/include/linux/Kbuild
@@ -47,6 +47,7 @@ header-y += coda_psdev.h
 header-y += coff.h
 header-y += comstats.h
```

```
 header-y += const.h
+header-y += cgroupstats.h
 header-y += cycx_cfm.h
 header-y += dlm_device.h
 header-y += dlm_netlink.h
diff -puN include/linux/cgroup.h~add-cgroupstats-v3 include/linux/cgroup.h
--- a/include/linux/cgroup.h~add-cgroupstats-v3
+++ a/include/linux/cgroup.h
@@ -13,6 +13,7 @@
 #include <linux/cpumask.h>
 #include <linux/nodemask.h>
 #include <linux/rcupdate.h>
+#include <linux/cgroupstats.h>

 #ifdef CONFIG_CGROUPS

@@ -29,6 +30,8 @@ extern void cgroup_fork(struct task_s
 extern void cgroup_fork_callbacks(struct task_struct *p);
 extern void cgroup_post_fork(struct task_struct *p);
 extern void cgroup_exit(struct task_struct *p, int run_callbacks);
+extern int cgroupstats_build(struct cgroupstats *stats,
+    struct dentry *dentry);

 extern struct file_operations proc_cgroup_operations;

@@ -305,6 +308,11 @@ static inline void cgroup_exit(struct

 static inline void cgroup_lock(void) {}
 static inline void cgroup_unlock(void) {}
+static inline int cgroupstats_build(struct cgroupstats *stats,
+     struct dentry *dentry)
+{
+ return -EINVAL;
+}

 #endif /* !CONFIG_CGROUPS */

diff -puN /dev/null include/linux/cgroupstats.h
--- /dev/null
+++ a/include/linux/cgroupstats.h
@@ -0,0 +1,70 @@
+/* cgroupstats.h - exporting per-cgroup 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_CGROUPSTATS_H
+#define _LINUX_CGROUPSTATS_H
+
+#include <linux/taskstats.h>
+
+/*
+ * Data shared between user space and kernel space on a per cgroup
+ * 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 cgroupstats {
+ __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 __CGROUPSTATS_CMD_MAX
+ */
+
+enum {
+ CGROUPSTATS_CMD_UNSPEC = __TASKSTATS_CMD_MAX, /* Reserved */
+ CGROUPSTATS_CMD_GET,  /* user->kernel request/get-response */
+ CGROUPSTATS_CMD_NEW,  /* kernel->user event */
+ __CGROUPSTATS_CMD_MAX,
+};
+
+#define CGROUPSTATS_CMD_MAX (__CGROUPSTATS_CMD_MAX - 1)
+
+enum {
+ CGROUPSTATS_TYPE_UNSPEC = 0, /* Reserved */
```

```
+ CGROUPSTATS_TYPE_CGROUP_STATS, /* contains name + stats */
+ __CGROUPSTATS_TYPE_MAX,
+};
+
+#define CGROUPSTATS_TYPE_MAX (__CGROUPSTATS_TYPE_MAX - 1)
+
+enum {
+ CGROUPSTATS_CMD_ATTR_UNSPEC = 0,
+ CGROUPSTATS_CMD_ATTR_FD,
+ __CGROUPSTATS_CMD_ATTR_MAX,
+};
+
+#define CGROUPSTATS_CMD_ATTR_MAX (__CGROUPSTATS_CMD_ATTR_MAX - 1)
+
+#endif /* _LINUX_CGROUPSTATS_H */
diff -puN include/linux/delayacct.h~add-cgroupstats-v3 include/linux/delayacct.h
--- a/include/linux/delayacct.h~add-cgroupstats-v3
+++ a/include/linux/delayacct.h
@@ -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)
@@ -71,6 +80,7 @@ static inline void delayacct_tsk_free(st

 static inline void delayacct_blkio_start(void)
 {
+ delayacct_set_flag(DELAYACCT_PF_BLKIO);
  if (current->delays)
   __delayacct_blkio_start();
```

```
 }
@@ -79,6 +89,7 @@ static inline void delayacct_blkio_end(v
 {
  if (current->delays)
   __delayacct_blkio_end();
+ delayacct_clear_flag(DELAYACCT_PF_BLKIO);
 }

 static inline int delayacct_add_tsk(struct taskstats *d,
@@ -116,6 +127,8 @@ 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/cgroup.c~add-cgroupstats-v3 kernel/cgroup.c
--- a/kernel/cgroup.c~add-cgroupstats-v3
+++ a/kernel/cgroup.c
@@ -42,6 +42,9 @@
 #include <linux/spinlock.h>
 #include <linux/string.h>
 #include <linux/sort.h>
+#include <linux/delayacct.h>
+#include <linux/cgroupstats.h>
+
 #include <asm/atomic.h>

 static DEFINE_MUTEX(cgroup_mutex);
@@ -1739,6 +1742,58 @@ static int pid_array_load(pid_t *pidarra
  return n;
 }

+/**
+ * Build and fill cgroupstats so that taskstats can export it to user
+ * space.
+ *
+ * @stats: cgroupstats to fill information into
+ * @dentry: A dentry entry belonging to the cgroup for which stats have
+ * been requested.
+ */
+int cgroupstats_build(struct cgroupstats *stats, struct dentry *dentry)
+{
+ int ret = -EINVAL;
+ struct cgroup *cont;
+ struct cgroup_iter it;
```

```
+ struct task_struct *tsk;
+ /*
+  * Validate dentry by checking the superblock operations
+  */
+ if (dentry->d_sb->s_op != &cgroup_ops)
+   goto err;
+
+ ret = 0;
+ cont = dentry->d_fsdata;
+ rcu_read_lock();
+
+ cgroup_iter_start(cont, &it);
+ while ((tsk = cgroup_iter_next(cont, &it))) {
+  switch (tsk->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(tsk))
+    stats->nr_io_wait++;
+   break;
+  }
+ }
+ cgroup_iter_end(cont, &it);
+
+ 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/taskstats.c~add-cgroupstats-v3 kernel/taskstats.c
--- a/kernel/taskstats.c~add-cgroupstats-v3
+++ a/kernel/taskstats.c
@@ -22,6 +22,9 @@
 #include <linux/delayacct.h>
 #include <linux/cpumask.h>
```

```
 #include <linux/percpu.h>
+#include <linux/cgroupstats.h>
+#include <linux/cgroup.h>
+#include <linux/file.h>
 #include <net/genetlink.h>
 #include <asm/atomic.h>

@@ -49,6 +52,11 @@ __read_mostly = {
  [TASKSTATS_CMD_ATTR_REGISTER_CPUMASK] = { .type = NLA_STRING },
  [TASKSTATS_CMD_ATTR_DEREGISTER_CPUMASK] = { .type = NLA_STRING },};

+static struct nla_policy
+cgroupstats_cmd_get_policy[CGROUPSTATS_CMD_ATTR_MAX+1] __read_mostly = {
+ [CGROUPSTATS_CMD_ATTR_FD] = { .type = NLA_U32 },
+};
+
 struct listener {
  struct list_head list;
  pid_t pid;
@@ -374,6 +382,51 @@ err:
  return NULL;
 }

+static int cgroupstats_user_cmd(struct sk_buff *skb, struct genl_info *info)
+{
+ int rc = 0;
+ struct sk_buff *rep_skb;
+ struct cgroupstats *stats;
+ struct nlattr *na;
+ size_t size;
+ u32 fd;
+ struct file *file;
+ int fput_needed;
+
+ na = info->attrs[CGROUPSTATS_CMD_ATTR_FD];
+ if (!na)
+  return -EINVAL;
+
+ fd = nla_get_u32(info->attrs[CGROUPSTATS_CMD_ATTR_FD]);
+ file = fget_light(fd, &fput_needed);
+ if (file) {
+  size = nla_total_size(sizeof(struct cgroupstats));
+
+  rc = prepare_reply(info, CGROUPSTATS_CMD_NEW, &rep_skb,
+     size);
+  if (rc < 0)
+   goto err;
+
```

```
+  na = nla_reserve(rep_skb, CGROUPSTATS_TYPE_CGROUP_STATS,
+     sizeof(struct cgroupstats));
+  stats = nla_data(na);
+  memset(stats, 0, sizeof(*stats));
+
+  rc = cgroupstats_build(stats, file->f_dentry);
+  if (rc < 0)
+   goto err;
+
+  fput_light(file, fput_needed);
+  return send_reply(rep_skb, info->snd_pid);
+ }
+
+err:
+ if (file)
+  fput_light(file, fput_needed);
+ nlmsg_free(rep_skb);
+ return rc;
+}
+
 static int taskstats_user_cmd(struct sk_buff *skb, struct genl_info *info)
 {
  int rc = 0;
@@ -524,6 +577,12 @@ static struct genl_ops taskstats_ops = {
  .policy  = taskstats_cmd_get_policy,
 };

+static struct genl_ops cgroupstats_ops = {
+ .cmd  = CGROUPSTATS_CMD_GET,
+ .doit  = cgroupstats_user_cmd,
+ .policy  = cgroupstats_cmd_get_policy,
+};
+
 /* Needed early in initialization */
 void __init taskstats_init_early(void)
 {
@@ -548,8 +607,15 @@ static int __init taskstats_init(void)
  if (rc < 0)
   goto err;

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

```
+ genl_unregister_ops(&family, &taskstats_ops);
 err:
  genl_unregister_family(&family);
  return rc;
```

–

--

_____
Containers mailing list
Containers@lists.linux-foundation.org
https://lists.linux-foundation.org/mailman/listinfo/containers