
Subject: Re: [RFC][mm] [0/2] Basic stats for cgroups V2
Posted by [Balbir Singh](#) on Tue, 08 Apr 2008 10:30:10 GMT
[View Forum Message](#) <> [Reply to Message](#)

Paul Menage wrote:

> On Sat, Apr 5, 2008 at 11:09 AM, Balaji Rao <balajirao@gmail.com> wrote:
>> V1->V2
>> - Fixed a possible race in cpu_cgroup_read_stat. Thank you Paul for pointing this out.
>> - A few other naming changes.
>>
>> This patchset is a first step towards implementing stats for cgroup
>> subsystems. Only a few trivial stats for cpu and memory resource controller
>> have been implemented for now. Please provide comments on the general
>> direction and any suggestions on how you would like the cgroupstats framework
>> to be implemented.
>
> This is sort of heading in the same way as the cgroup binary stats API
> that I mentioned a couple of months ago (when I proposed the
> "cgroup.api" file).
>
> Since the cgroup file API encourages subsystems to export values via
> abstract methods such as read_s64() or read_map() rather than having
> them handle the file I/O themselves, this gives the basis for a binary
> stats API - the same methods can be used to retrieve the information
> in a binary form rather than from regular ASCII-based file reads, and
> the subsystem doesn't have to care which is being used.
>
> I was originally thinking along the lines of having a special mode in
> which you could obtain a cgroupfs binary file for a cgroup directory
> that would report a requested set of binary stats each time it was
> read, but using the netlink/taskstats API might be a good approach
> too.
>
> One of the important API choices would be whether the stats API was
> fixed in header files shared with userspace, or whether it would be
> possible for stats to be added and dynamically discovered/used by
> userspace without needing fixed header file descriptions.
>
> The difference would be a bit like the old sysctl API (where each
> sysctl entry had to be enumerated in a header file) versus the newer
> /proc/sys approach where numerical values aren't used and userspace
> can determine which entries are supported at runtime, and even access
> new previously-unknown entries.
>
> Here's one possible way to do it:
>
> With the taskstats interface, we could have operations to:
>

- > - describe the API exported by a given subsystem (automatically
- > generated, based on its registered control files and their access
- > methods)
- >
- > - retrieve a specified set of stats in a binary format
- >
- > So as a concrete example, with the memory, cpuacct and cpu subsystems
- > configured, the reported API might look something like (in pseudo-code
- > form)
- >
- > 0 : memory.usage_in_bytes : u64
- > 1 : memory.limit_in_bytes : u64
- > 2 : memory.failcnt : u64
- > 3 : memory.stat : map
- > 4 : cpuacct.usage : u64
- > 5 : cpu.shares : u64
- > 6 : cpu.rt_runtime_ms : s64
- > 7 : cpu.stat : map
- >
- > This list would be auto-generated by cgroups based on inspection of
- > the control files.
- >
- > The user could then request stats 0, 3 and 7 for a cgroup to get the
- > memory.usage_in_bytes, memory.stat and cpu.stat statistics.
- >

The user needs to inspect the reported API before requesting for statistics?

- > The stats could be returned in a binary format; the format for each
- > individual stat would depend on the type of that stat, and these could
- > be simply concatenated together.
- >
- > A u64 or s64 stat would simply be a 64-bit value in the data stream
- >
- > A map stat would be represented as a sequence of 64-bit values,
- > representing the values in the map. There would be no need to include
- > the size of the map or the key ordering in the binary format, since
- > userspace could determine that by reading the ASCII version of the map
- > control file once at startup.
- >
- > So in the case of the request above for stats 0, 3 & 7, the binary
- > stats stream would be a sequence of 64-bit values consisting of:
- >
- > <memory.usage>
- > <memory.stat.cache>
- > <memory.stat.rss>
- > <memory.stat.active>
- > <memory.stat.inactive>

> <cpu.stat.uptime>
> <cpu.stat.stime>
>
> If more stats were added to memory.stat or cpu.stat by a future
> version of the code, then they would automatically appear; any that
> userspace didn't understand it could ignore.
>
> The userspace side of this could be handled by libcg.
>

That sounds nice.

> Thoughts?

I like the overall approach, do you have a prototype implementation?

--

Warm Regards,
Balbir Singh
Linux Technology Center
IBM, ISTL

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