Subject: Re: Supporting overcommit with the memory controller
Posted by Balbir Singh on Thu, 06 Mar 2008 18:42:18 GMT
View Forum Message <> Reply to Message

Paul Menage wrote:
> We want to be able to use the memory controller in the following way,
> and I'd like to know how practical this is currently, and will be in
> the future.
>
> Users are poor at determining how much memory their jobs will actually
> use (partly due to poor estimation, partly due to high variance of
> memory usage on some jobs). So, we want to overcommit machines, i.e.
> we want the total limits granted to all cgroups add up to more than
> the total size of the machine.
>
> Our central scheduler will try to ensure that the jobs that are packed
> on to the same machine are unlikely to all hit their peak usage at
> once, so the machine as a whole is unlikely to actually run out of
> memory. But sometimes it will be over-optimistic, and the machine will
> run out of memory. We will try to ensure that there's a mixture of
> high and low priority jobs on a machine, so that when the machine runs
> out of memory the OOM killer can nuke the low-priority jobs and we can
> reschedule them elsewhere.
>
> The tricky bit is that we don't want this OOM process to impact the
> high-priority jobs on the machine. I.e. even while the low-priority
> job is OOM-killing itself, the high priority job shouldn't have any
> difficulty in doing regular memory allocations. And if the
> high-priority job gets a spike in its memory usage, we want the
> low-priority jobs to get killed quickly and cleanly to free up memory
> for the high-priority job, without stalling the high-priority job.
>
> So for each job we need a (per-job configurable) amount of memory
> that's essentially reserved for that job. That way the high-priority
> job can carry on allocating from its reserved pool even while the
> low-priority job is OOMing; the low-priority job can't touch the
> reserved pool of the high-priority job.
>
> But to make this more interesting, there are plenty of jobs that will
> happily fill as much pagecache as they have available. Even a job
> that's just writing out logs will continually expand its pagecache
> usage without anything to stop it, and so just keeping the reserved
> pool at a fixed amount of free memory will result in the job expanding
> even if it doesn't need to. Therefore we want to be able to include in
> the "reserved" pool, memory that's allocated by the job, but which can
> be freed without causing performance penalties for the job. (e.g. log
> files, or pages from a large on-disk data file with little access
> locality of reference) So suppose we'd decided to keep a reserve of

> 200M for a particular job - if it had 200M of stale log file pages in
> the pagecache then we could treat those as the 200M reserve, and not
> have to keep on expanding the reserve pool.
>
> We've been approximating this reasonably well with a combination of
> cpusets, fake numa, and some hacks to determine how many pages in each
> node haven't been touched recently (this is a bit different from the
> active/inactive distinction). By assigning physical chunks of memory
> (fake numa nodes) to different jobs, we get the pre-reservation that
> we need. But using fake numa is a little inflexible, so it would be
> nice to be able to use a page-based memory controller.
>
> Is this something that would be possible to set up with the current
> memory controller? My impression is that this isn't quite possible
> yet, but maybe I've not just thought hard enough. I suspect that we'd
> need at least the addition of page refault data, and the ability to
> pre-reserve pages for a group.

I have some patches for implementing soft-limits. Have you explored to see if they can sort your problem? I am thinking of adding additional statistics like page-in, page-out rates and eventually refault statistics.

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