Portage Versions Bisection Findings

Authors: xcl@
Last Updated: Aug 2, 2022
go/cros-portage-bisection-findings

Introduction

Upgrading to portage-3.0.21 in October 2021 caused a severe performance regression in build time. We want to investigate what caused the regression and explore fixes.

Background

The performance regression was reported as an increase in CQ run time (b/226933790) with the "init SDK", "update SDK", and "install packages" steps seeing the most increase in runtime.

Further investigation revealed what appears to be 2 major regressions:

1. ~10x increase in time spent calculating dependencies.
2. Low load-averages indicating that jobs are not being run in parallel.

We'll dive into what caused each of these regressions below.

Metrics

We used cl-perf to gather metrics and focused on the "install packages" phase of builds to measure performance for consistency.

To bisect portage versions, we merged upstream versions of portage into our current branch (chromeos-2.3.75). Overall, there were few merge conflicts to resolve.

See go/portage-regression-tracking-summary for links to individual spreadsheets for all metrics gathered.

Dependency Calculations (Approx. 800% Increase)

Cause

Commit: 9b755b46f in portage-2.3.99
Bug: https://bugs.gentoo.org/717140
Metrics:

- cl-perf (portage-2.3.98 + ad325eb10..f04960466)
- cl-perf (portage-2.3.98 + ad325eb10..f04960466 + 9b755b46f)
- cl-perf (portage-3.0.21)
- cl-perf (portage-3.0.32)

Analysis

From the build logs of one run that included the regression, we can see that we spend almost 2 hours (start timestamp: 13:09:30.780, end timestamp: 15:00:55.781) calculating dependencies. In contrast, from the build logs of one run for the baseline (portage-2.3.75), calculating dependencies only took 5 minutes (start timestamp: 12:23:05.475, end timestamp: 12:28:38.690).

The commit appears to be a "correctness" fix for an issue introduced in portage-2.2.13. From the build logs for portage-3.0.21, we see that the regression still exists in later versions of portage.

Open Questions

1. Do/will we ever run into this issue in our tree?
    a. portage-2.3.75 seems to be ok. Since we reverted to portage-2.3.75 in early April, there haven't been any reported issues like what was reported upstream.
    b. Tried reverting this commit and did not run into downgrade/upgrade loops like reported upstream though theoretically the reported issue seems like an issue we should run into.
2. Why did this cause such a large performance regression?
    a. The commit changes the logic in `_slot_operator_update_probe()` at the core of portage's dependency calculation logic.
    b. Compared to before where a package was not considered as a package that needed to be rebuilt, the updated logic now considers these packages as needing to be rebuilt.
    c. Larger number of packages that need to be rebuilt --> more dependency calculations --> more time spent "calculating dependencies".

Low Load-Averages (Approx. 50% Increase)

Cause

Commit: 50da2e165 in portage-2.3.93
Bug: https://bugs.gentoo.org/711688
Metrics:

- cl-perf (portage-2.3.89 + 84b3556158d..a287c49f84a)
- cl-perf (portage-2.3.89 + 84b3556158d..a287c49f84a + 50da2e165)

Analysis

There were a series of changes made starting from portage-2.3.90 to rewrite logic around asynchronous operations. These changes caused other issues that were resolved from portage-2.3.91 to portage-2.3.93 (50da2e165 is one of these fixes).

From the baseline build logs, we see that the load average rapidly increases to about 20 with 168 jobs completed. In contrast, from the build logs of one run that included the regression, we don't reach a load average of 20 until 1178 jobs completed despite running 32 jobs. With 168 jobs completed, the load average is about 5 and does not increase significantly until the end.

The regression commit removes `isAlive()` implementations from `AsynchronousTask` subclasses, instead opting to use the default definition of `isAlive()` in `AsynchronousTask`. The default implementation appears to keep various processes alive for longer than before, leading to less parallelization.

Open Questions

1. Can we safely revert some/all of the commit (50da2e165)?
    a. The commit fixes situations where `isAlive()` was returning `False` early in the `CompositeTask` subclass. This terminated processes early that were fetching binpkgs so binpkgs were not being fetched.
    b. Tried reverting this commit and was able to fetch binpkgs. Could consider a partial revert where we use the default implementation of `isAlive()` for the `CompositeTask` subclass but keep the individual subclass implementations for `AbstractPollTask` and `FifoIpcDaemon` (`SubProcess`'s `isAlive()` implementation is already the default).
2. Why did this cause such a large performance regression?
    a. Probably due to our usage of binhosts and binpkgs. Since we generally prefer binpkgs (to building from source), every time we fetch a binpkg, the time the process is kept alive increased. Multiply the increase over the number of packages we have in our tree, we have a large regression.
    b. Upstream bug filed: https://bugs.gentoo.org/866085