Revisiting Process-per-Site

This Document is Public

Author: haraken@

2023 Feb

Motivation

Currently each renderer process has its own blink::MemoryCache. When you open three tabs for three Google Docs, Chrome creates three same-site renderer processes, each of which has its own blink::MemoryCache.

When you open the first Google Docs, the renderer process creates a blink::MemoryCache from scratch. It loads scripts, fonts, stylesheets, images etc from the disk cache or the network and decodes the resources into the blink::MemoryCache. The cached, decoded resources can be reused only for the upcoming same-site navigation that happens in the same tab. When you click a link of another Google Docs, it opens in another tab, which is hosted by a different renderer. The renderer needs to create its blink::MemoryCache from scratch.

If we can share blink::MemoryCache among all same-site renderers, the second and later same-site renderers can load pages faster. This improves the global LCP. Also it reduces Memory.Renderer.PrivateMemoryFootprint because the same-site renderers can share one cache. This enables us to cache more resources without regressing Memory.Renderer.PrivateMemoryFootprint, which contributes to improving the global LCP more.

With this motivation, we are brainstorming the idea of sharing blink::MemoryCache among same-site renderers.

However, this brings one question: Can we go beyond? Instead of sharing blink::MemoryCache, **what happens if we load same-site main documents in the same renderer process (until the number of main documents in the process reaches some threshold)?**

Proposal

The proposal is to **run a Finch experiment to host same-site main documents in the same renderer process until the number of main documents in the process reaches some threshold and measure the performance / memory impact using UKM**.

This is close to the Process-per-Site mode. According to this document:

> *Process-per-site: This model consolidated all instances of a given site into a single process (per profile), to reduce the process count. It generally led to poor usability when a single process was used for too many tabs. This mode is still used for certain limited cases (e.g., the New Tab Page) to reduce the process count and process creation latency. It is also used for extensions to allow synchronous scripting from a background page. Note that having a single process for a site might not be guaranteed (e.g., due to multiple profiles, or races).*

The difference between the proposal and the Process-per-Site mode is that the proposal enforces a limit about the number of main documents hosted by one renderer to mitigate the performance concerns. We can run Finch experiments for multiple thresholds (e.g., 2, 4, 6) and measure the performance / memory impact using UKM.

Evaluating the performance / memory impact using UKM is important because it may improve performance of some websites but regress performance of other websites. For example, for Google Workspace, performance isolation matters for their responsiveness and they enable Origin Isolation. If we host two Google Docs in one renderer, it may improve LCP but is likely to regress responsiveness. However, the proposal may be a performance / memory win for most websites in the long tail. UKM will tell us the result.

Specifically:

- Introduce a new ProcessReusePolicy to share a renderer process among same-site main documents until the number of the main documents reaches a threshold.
- Tweak the process selection policy in RenderProcessHostImpl::GetProcessHostForSiteInstance and enable the experiment.
- Exclude Origin Isolation and COOP cases from the experiment.

The expected results are:

- For some heavy websites: LCP improves. PMF improves. FID and responsiveness metrics regress. CLS stays the same.
- Most websites: LCP improves. PMF improves. FID and responsiveness metrics slightly regress (how much?). CLS stays the same.