Publicly shared

falken@, horo@, kenjibaheux@, kinuko@, shimazu@

April 6th 2018

Context

Adding a Service Worker doesn't magically result in better performance. Web developers need to invest in their service worker to get performance returns. For instance, many web developers of progressive web apps are using their Service Worker to serve a shell while fetching fresh content. On the Chrome side, we are working hard to improve the performance of Service Workers to satisfy even the most performance sensitive websites. Our plan for 2017 resulted in massive improvements for our most demanding partners (e.g cumulative improvements on the order of 20x) and we also refined our understanding of speed opportunities in our implementation. So, it's time for a new roadmap!

Opportunities

Main Thread Contention in the Renderer

Main Thread Contention used to be a generic latency concern. But, because of the work we did in 2017 with respondWith and Fetch API, it is now mainly a **startup latency concern**.

- Creating a "shadow page (hidden document)" is a fundamental architectural problem resulting in extra latency.

IO Thread Contention and Throttling in the Browser

IO Thread Contention is **a generic latency concern** with potential impact at startup and for every Fetch API call.

- SW startup in the browser process goes through multiple thread-hops between UI/IO, which adds 100+ ms overhead on slow devices.
- Resources loaded by SW are throttled by SafeBrowsing and ResourceScheduler in the browser process, but this throttling is not always necessary.

Accelerating Script Execution

Script execution is a significant latency factor in the renderer process. We can reduce its impact by aggressively caching V8 optimized code for the javascript resources that are associated with Service Workers.

Roadmap

Project	crbug	Motivation	Description	ETA / Status
V8 full Code Caching	SW scripts: 788619	Script execution cost	Script execution is the heaviest task in the renderer. We can reduce its impact by eagerly and fully compile	**M65:** approved for launch.

for SW and installed scripts	Installed scripts: 788621		scripts that are stored in Cache Storage or otherwise associated with a Service Worker.	
Service Worker servicification	715640	Remove process round-trips and undesirable throttling for every Fetch calls.	Re-architecturing SW code to let pages directly talk to their SW. Estimated gain: unclear on its own (possibly 10s of ms per Fetch for throttling removal), but this unblocks many other fixes/optimizations. Note: this is a major rewrite of ServiceWorker and will therefore take time to complete.	2018 Q2 P1 in 2018 (tasks tracker)
Move browser-side Service Worker code to UI thread	824858 824840	Remove thread-hops to avoid IO thread contention during startup. And code health.	We can remove the thread-hops by moving all the Service Worker code in the browser process onto the UI thread. Estimated gain: 100+ ms on Android Go	2018 Q3+ Dependency: Service Worker Servicification. Needs hard data from slow windows/CrOS devices.
Bypass/delay shadow page creation for installed SW startup	820329 692909	Main thread latency during startup	The start-up sequence of workers relies on the main thread to create and setup a shadow page. However, the main thread tend to be busy while loading a page which results in startup latencies. Estimated gain: 100s of ms (cold navigation). StartWorker message latency metric shows that we wait 750-1000ms for the message at 95p and 10-30ms at median. Design Doc	2018 Q3+ ? (Not yet started)
Mitigate Shadow Page overhead at startup	719100	Main thread latency during startup	See above for context. There is work done during the shadow page setup that is unnecessary for Service Worker (e.g. style-related initialization).	2018 Q2

			We could skip such unnecessary work but the issue will be irrelevant as we remove the shadow page altogether. Estimated gain: unknown?	
Remove Shadow Page (hidden document)	538751	Main thread latency during startup	See first item in this section for context. For new workers, we still need to create a shadow page in the current architecture.	2018 Q4 Dependencies: 1. Off main-thread import scripts (horo@) 2. Off main-thread WebSocket (nhiroki@)
Off main-thread importScripts	706331	Main thread latency during startup and unblocking shadow page removal.	Fetch API calls are now done off the main-thread, but importScripts aren't yet. Estimated gain: unclear, **but** should also **unblock shadow page removal**.	2018 Q3 (horo@)
Avoid SafeBrowsing throttling for installed resources	817909	Unnecessary throttling	Currently fetching resources always goes through SafeBrowsing throttling even if they come from Cache Storage. We already bypass this for installed SW scripts, but not for other resources. It might be reasonable to do so. Estimated gain: 10s~100s of ms per request. Android Go: may cost a process start when the SB service isn't loaded.	(Not yet decided) (Not yet started) Dependency: SW Servicification (high complexity otherwise) Discussion with SB team.
Skip SW for certain resources	Spec issue	Fetch latency	Provide the ability to skip SW for resources that are known to never be in the cache, e.g. videos. Options: - Fetch API flag - HTML element attribute - Static route (MVP: opt-out routes at install)	TBC

Archive

Roadmap 2017~2018Q1

Green: shipped

Yellow: on-going

Project	crbug	Motivation	Description	ETA / Status
Navigation Preload	661071	Startup cost.	Allow navigation and service worker startup to run in parallel. Estimated gains: on the order of a network RTT.	M59: launched. (origin trial in M58).
Mojo pipe for respondWith()	690795	Main thread latency.	Allow streaming responses without relying on the main thread. Side-benefits: The responses from Navigation Preload will no longer be blocked by the main thread. Increase the potential of the Off the main thread Fetch project. Estimated gains: 100s of ms.	M60: launched.
~~Mitigate the impact of preconnect~~	727544	Startup cost.	A busy IO thread can delay the Service worker startup tasks. We found out that pre-connections can hog the IO thread for a long time. Avoid having pre-connections block Service Worker startup tasks. Estimated gains: 100s of ms.	Cancelled: impact would be minimal with PlzNavigate (shipping soon).
SW Script Streaming	683037	Startup cost.	Remove round-trips between the Renderer and Browser processes when starting up a service worker. Side-benefit: the resource scheduler will no longer throttle the requests for Service Worker scripts. Estimated gains: 10s of ms per script.	M64
UMA breakdown and fixes to Nav		Insights	More UMA breakdown. Fixes to our navigation preload UMAs.	M61.

Preload UMA				
Off the main thread StartWorker	692909	Startup cost (due to main thread latency).	*startWorker* relies on the the main thread to perform its tasks. Unfortunately, the main thread tend to be busy when a website starts to load. Move startWorker off the main thread. Estimated gains: 100s of ms per cold navigation. Design Doc	Proof of concept CL. ETA: late-2017.
Off the main thread Fetch	443374	Main thread latency.	Fetch operations issued from a worker are handled by the main thread. Unfortunately, the main thread tend to be busy with other competing tasks. Move Fetch off the main thread. Estimated gains: 100s of ms.	Enabled by default in M62. Monitoring metrics.
Service Worker servicification	612285	Remove round-trips between browser and renderer. Directly communicate with SW.	Remove more round-trips between browser and renderer processes. Side-benefit: the resource scheduler will no longer throttle requests issued by a service worker. Estimated gains: 10s of ms per requests. Note: this is a major rewrite of ServiceWorker and will therefore take time to complete.	ETA early 2018. Prototyping has started. Behind the experimental flag: late 2017.
Fast Back navigation		Back navigation on Chrome is negatively impacted by Service Worker.	Chrome doesn't have a back/forward cache. Our back/forward navigations are done by issuing a page load with a flag to prefer cached resources. This means that a Service Worker may get involved on back/forward navigations in Chrome. Browsers with a back/forward cache do not have this particular	Approach is still TBD.

			issue.	