(Public)

Browser-side scheduling roadmap

skyostil@
June 8th, 2018

This document presents a series of scenarios where suboptimal task scheduling in the browser process negatively impacts user experience. We propose a high level roadmap of architectural improvements to address these issues.

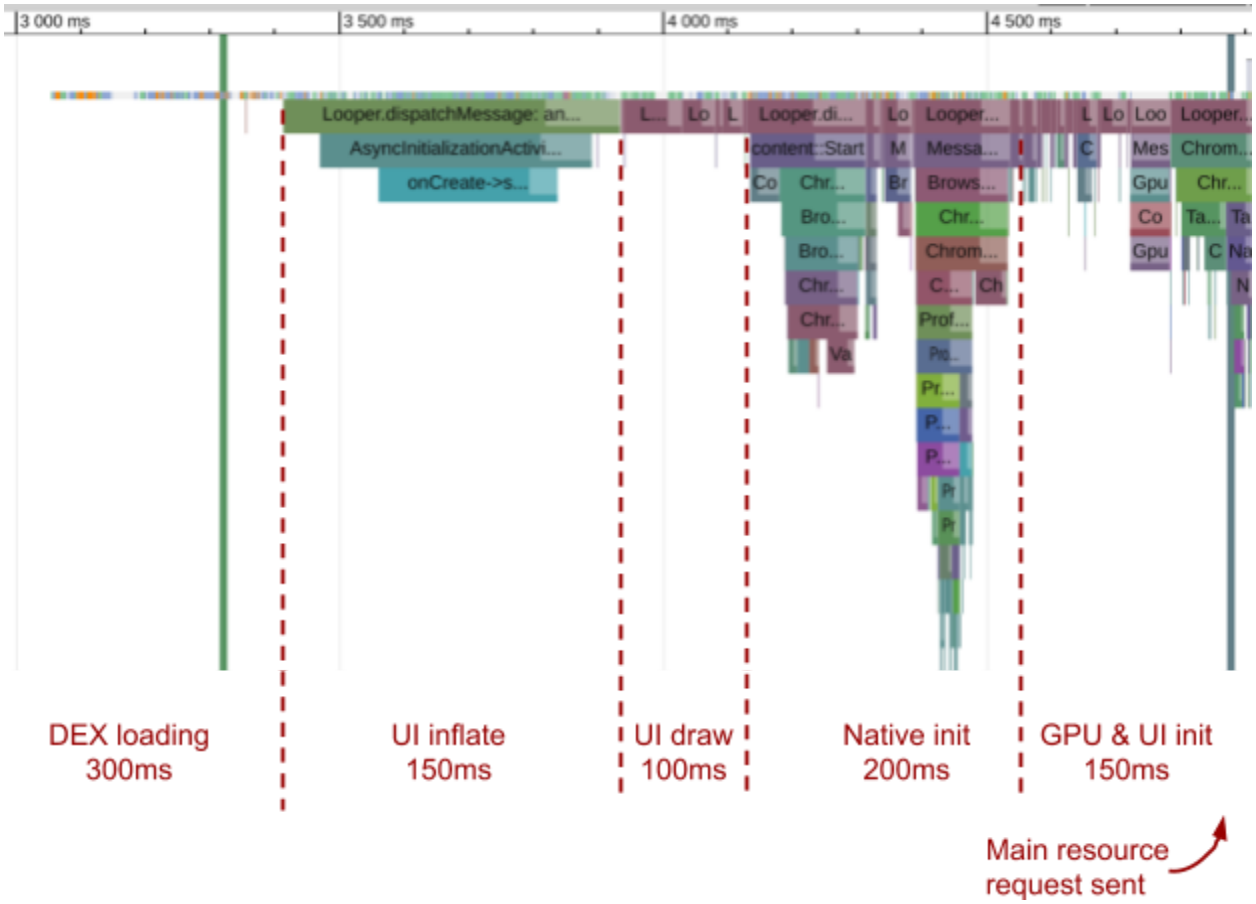
Unless noted otherwise, all data was obtained from a Huawei Y5 lite (Android Go, 512MB configuration) running Android 8.1.0.

Contents

Startup critical path optimization

Tracking bug: crbug.com/835323

During the Hello Chrome hackathon we found that tasks on the critical path to sending the initial network request were being delayed by unrelated work. The following trace shows an overview of the browser process startup sequence:



For a view intent startup scenario (roughly 50% of all intents sent to Chrome), the main goal is to send out the request for the main resource as soon as possible. Examining each of the phases until that point individually, we note both ongoing work to improve their respective runtimes as well as propose some new optimizations:

- **DEX loading:** Java code ordering (crbug.com/849239) should help reduce the time it takes for Chrome's code to start running.
- **UI inflate/draw:** There is an effort to lazily inflate the toolbar (crbug.com/819760) – initially for WebAPKs but later also for Chrome itself – which will drop the time spent in setContentView(). **(New)** As a follow-up experiment, we want to try moving all UI initialization until after the initial navigation request is sent (crbug.com/849324).
- **Native init:** **(New)** Make KeyedService initialization asynchronous (crbug.com/849305) to ensure only necessary services are initialized during startup. Potentially based on C++ promises.
- **GPU & UI init:** **(New)** Similarly, we want to attempt lazy GPU initialization until the main resource request is sent (crbug.com/849327).
- **Overall:** **(New)** A common pattern in startup traces is that code execution is often delayed by synchronous I/O. We propose investigate whether this I/O is caused by code page faults, and if so, use instruction cache misses tracing to investigate which areas of Chrome

are contributing most (crbug.com/849316). (Note that this project also helps define the requirements for sampling tracing in Perfetto.)

Additionally we propose a set of code health improvements to how early startup tasks are scheduled and how Java integrates with the native task scheduler:

- Introduce a dedicated thread for early Java startup tasks (crbug.com/TODO).
- Get all AsyncTasks onto base::TaskScheduler and make sure they are instrumented (crbug.com/825947).
- Assign priorities for Java AsyncTasks (crbug.com/TODO).
- Enable background thread priority on Android (crbug.com/801355).

Beyond these initial improvements, we want to investigate defining a well-lit startup path for the different startup scenarios (e.g., new tab page, tab restore, PWA/TWA, CCT, downloads, WebView) (crbug.com/841403). This ensures only the necessary browser components are being loaded in each case.

Note that this works represents only the initial set of startup work; a more complete backlog is maintained separately.

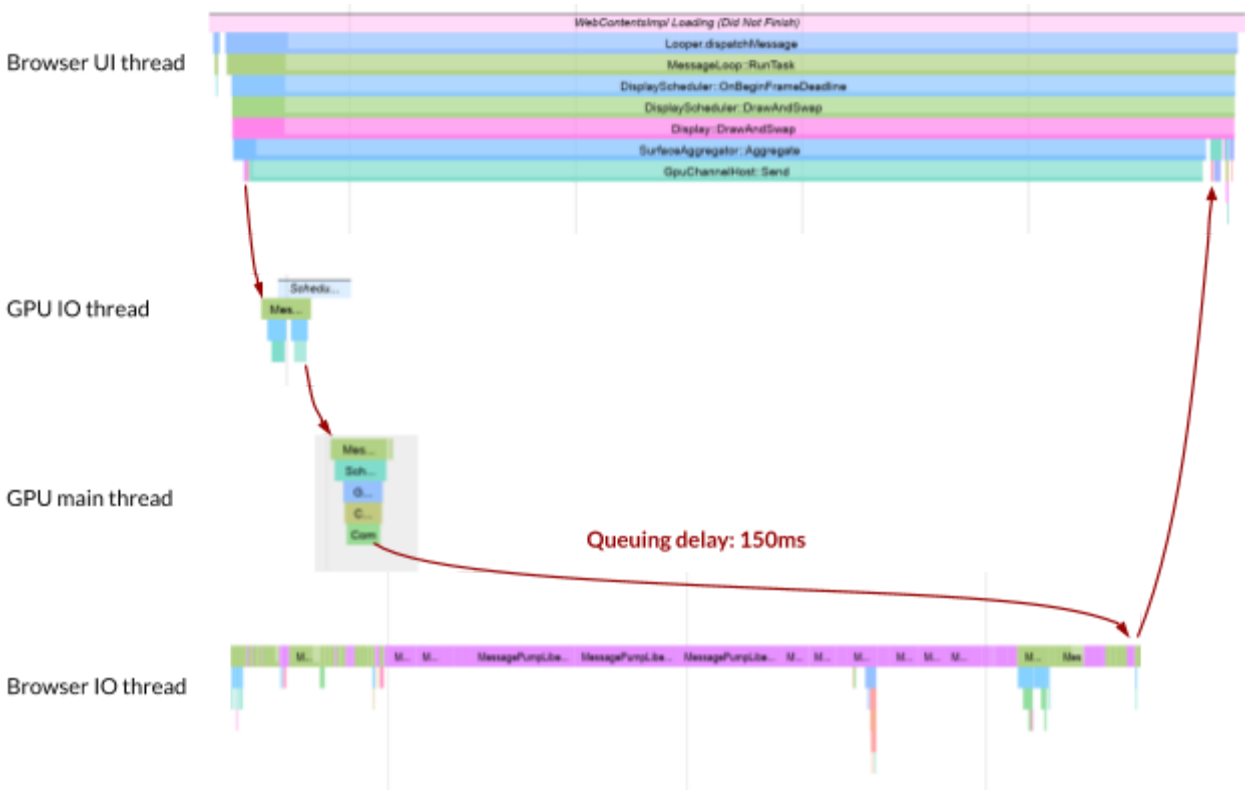
Metrics: Startup time (Startup.Android.Cold.TimeToFirstContentfulPaint.Tabbed)

Congestion on the I/O thread

Tracking bug: crbug.com/861708

The I/O thread is used for unmarshalling and handling mojo/IPC messages on the receiving side. Currently these messages are handled in FIFO order, which means nominally higher priority tasks may get delayed by less important ones.

The following trace on the site golem.de show a scenario where the browser UI thread performs a synchronous IPC to the GPU service during rendering. The message itself is handled promptly on the GPU I/O and main threads, but because the browser I/O thread is busy processing other work (socket notifications in this case), the response unblocking the IPC is delayed by 150 ms, during which the UI appears frozen.



To avoid this problem we propose to introduce priorities to tasks running on the I/O thread. This ensures latency-critical work such as input and GPU messages take priority over other tasks. On a high level, we propose to use a SequenceManager (from the Blink scheduler) to implement task prioritization and route selected mojo/IPC messages to higher priority task queues.

A prerequisite for this is making SequenceManager available for use outside Blink. This involves moving the code to //base/task_scheduler/sequence_manager as outlined in the Jolly Jumper project. SequenceManager also allows us to instrument the I/O thread with performance metrics to help prioritize and track task scheduling progress.

Metrics: Primarily input latency and scroll smoothness (Event.Latency.ScrollBegin.Touch.TimeToScrollUpdateSwapBegin2, Event.Latency.ScrollUpdate.Touch.TimeToScrollUpdateSwapBegin2), but startup latency (Startup.Android.Cold.TimeToFirstContentfulPaint.Tabbed) may also improve.

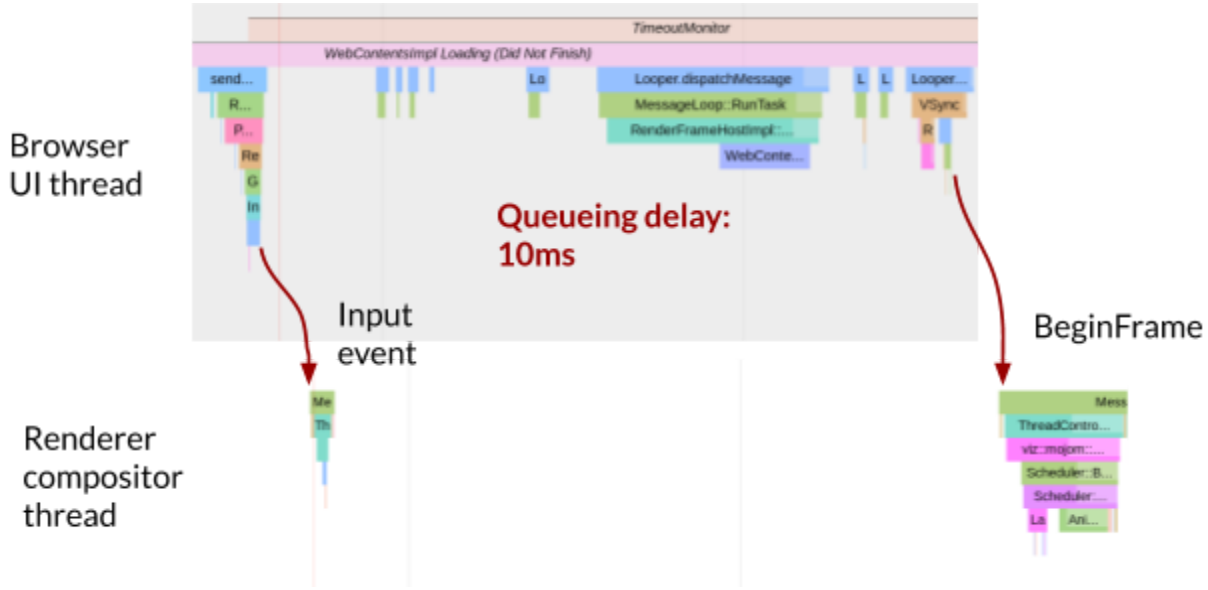
Java vs. C++ prioritization

Tracking bug: crbug.com/863341

On Android, the Browser UI thread multiplexes two types of work: C++ MessageLoop tasks and Java Handler tasks. The Java tasks further consist of Chrome's own Java tasks and native Android

ones. Currently there is no relative prioritization between the two – CPU time is split in a simple round-robin fashion between Java and C++.

The following trace of scrolling the site hs.fi shows an example where this lack of prioritization causes a frame drop. Since touch events and the vsync signal (which triggers rendering) are delivered by separate Java tasks, it's possible for C++ tasks to run between them. In this case, navigation related tasks delayed the the vsync task by 10 ms, causing the renderer to miss the frame deadline.



We propose to eliminate these types of problems by introducing prioritization logic to decide when to swap control between C++ and Java tasks. For example, if a high priority task (e.g., an input event) is pending in the Java Handler queue, C++ tasks should be deferred until it is handled. To enable this, the batching logic of SequenceManager and the Android message pump is made to take pending tasks of the opposite type into account.

In addition to the SequenceManager metrics mentioned above, we also want to investigate capturing Slow Reports based on long tasks detected on the UI thread.

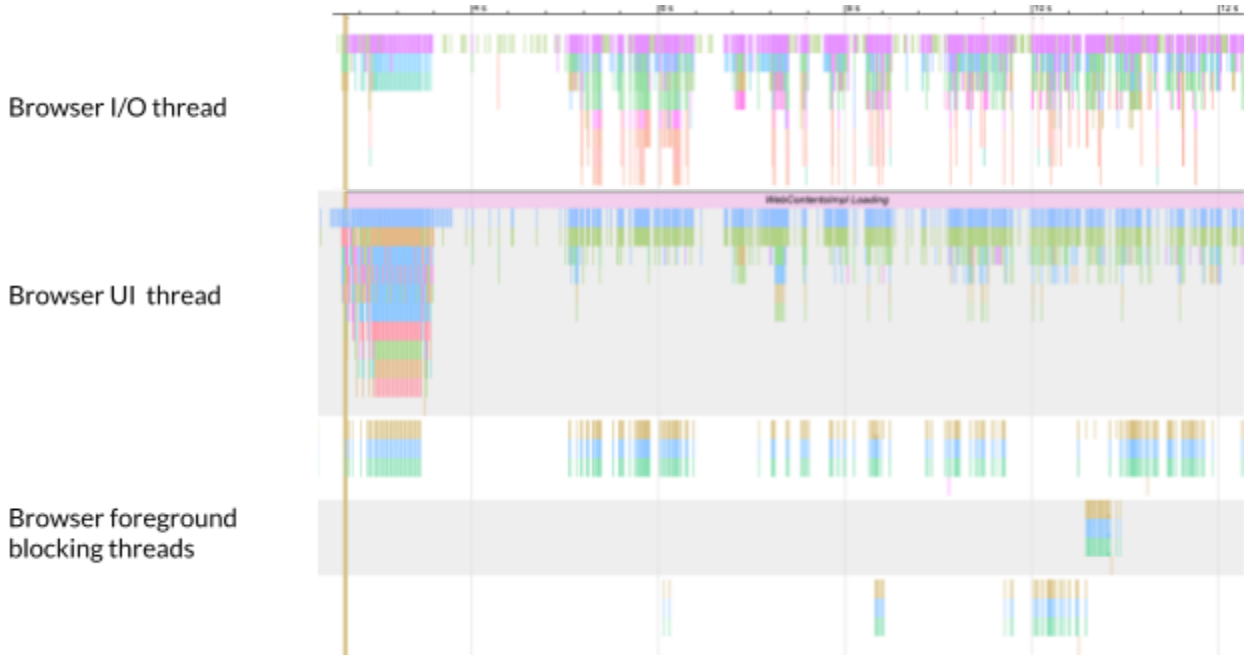
Metrics: Input latency and scroll smoothness (Event.Latency.ScrollBegin.Touch.TimeToScrollUpdateSwapBegin2, Event.Latency.ScrollUpdate.Touch.TimeToScrollUpdateSwapBegin2)

Per-frame scheduling in the browser

Tracking bug: TODO

Currently when the browser performs work on behalf of the renderer (e.g., network loads, database access), each tab and frame is given equal priority. However, the user more likely cares about tasks affecting visible tabs, so we should give them higher priority.

The following trace (Captured on an 1GB Evercoss A65) shows an example of opening a link in a background tab on an otherwise idle page:



Except for the initial burst of activity (tab opening animation), all the shown tasks are are potentially competing with work for the foreground tab. To fix this priority inversion, we propose to:

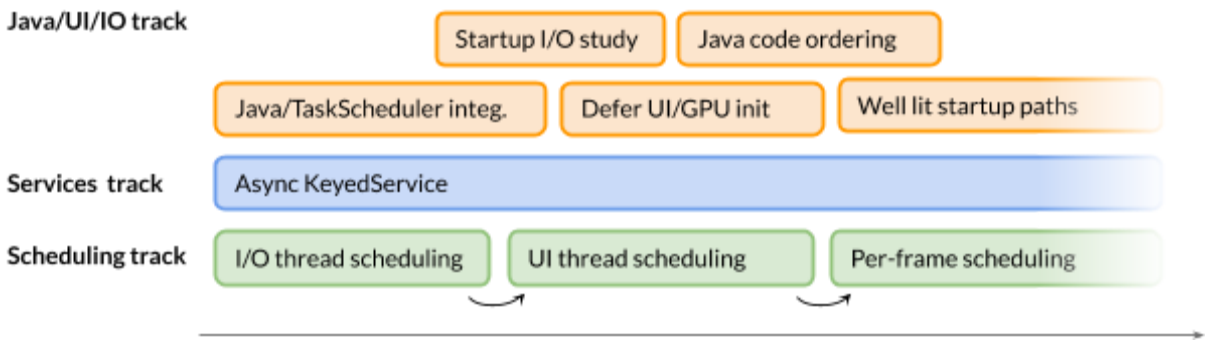
- Pause background renderers while the foreground tab is being interacted with. Initially this means freezing backgrounded pages in the renderer process, but once browser-side tasks are annotated with a render frame, tasks for background pages can also be paused.
- Drop the priority of work triggered by background tabs to BACKGROUND from USER_BLOCKING/USER_VISIBLE.

- Add per-frame task queues on the I/O thread and prioritize visible frames over invisible ones.

Metrics: Time to first meaningful paint (PageLoad.PaintTiming.NavigationToFirstContentfulPaint), input latency and scroll smoothness (Event.Latency.ScrollBegin.Touch.TimeToScrollUpdateSwapBegin2, Event.Latency.ScrollUpdate.Touch.TimeToScrollUpdateSwapBegin2), potentially also session restore time.

Implementation plan

As the listed projects are mostly independent, we have some flexibility in sequencing them. We propose to split the work into three parallel tracks (with arrows designating hard dependencies):



Note that more detailed planning is required to better estimate the sizes of the individual projects.