Next-gen scheduling architecture

altimin@chromium.org
31 Aug 2017

This document provides a brief overview of the current scheduling architecture and a plan to refactor the policy part of the renderer process to a) the browser process for global decisions (to be able to make decisions involving global state of all processes) b) frame schedulers for local decisions (for fine-grain control and performance isolation). This document does not focus on the specifics — there will be another, more detailed document covering this.

For now this document discusses only Blink main thread scheduling, because scheduling requirements for other threads are simple and should not present a problem.

Present situation

RendererScheduler lives on the main thread* of the renderer process. It aggregates a number of signals originating in various places, including the browser process, the main thread of the renderer process and the compositor thread. Based on these signals, RendererScheduler formulates task queue policies (which include things like priorities and whether task queue should be disabled or throttled).

Note that RendererScheduler isn't the only place in the scheduler where policy decisions are made: WebFrameScheduler also can independently trigger task queue throttling when a frame is offscreen or invisible.

Signals used by renderer scheduler include:

- Renderer status (renderer visible/hidden/backgrounded), plumbed via an IPC from the browser process.
- Browser-provided frame status (e.g. whether frame is playing audio)
- Local frame status (frame visibility inside a page).
- Compositor signals (WillBeginMainFrame, BeginMainFrameNotExpectedSoon, etc).
- Input signals (i.e. scrolls and other user gestures).

Limitations

The main drawback of the current approach is that policy decisions are limited to a single renderer and do not take into account global situation. For example, we don't know how busy is the system and can't set priorities based on the state of the other tabs — if there is a heavy page being loaded in the foreground it makes sense to lower priority of the background tabs.

New architecture

It is proposed to move all* policy bits of the renderer scheduler to a browser process (or a service). It makes sense given that most major signals (renderer backgrounding/page backgrounding/audio state/input signals) originate in the browser process and are being plumbed to the renderer scheduler via IPCs.

Renderer scheduler provides a nice layer of abstraction which can facilitate this move — there is a boundary between calculating task queue policies and applying them to task queues. It is proposed to move policy calculation to the browser process and send the calculated task queue policies to the renderer process. Number of IPC involved should not change due to most signals already being plumbed via IPCs to the renderer process.

Challenges

Renderer compositor signals

Some signals used in scheduling originate from compositor thread (e.g. whether a user gesture should be handled by main thread instead of compositor thread etc). It seems possible to handle these signals in the renderer process and send the results of the calculation (use_case and touchstart_expected_soon) to the browser process. The danger here is that this will introduce additional IPC on a input handling critical path and add several milliseconds(?) to scroll latency. It also may be possible to get input-related signals directly from the browser (they originate there before being plumbed via compositor). input-dev@ consultation is needed here.

Per-task decisions

Due to resource limitations (mainly number of IPC messages) it isn't possible to let browser process know durations of all individual tasks. That means that all decisions involving tasks should be handled in the renderer process. There are two big categories of this:

- Determining if the tasks are expensive. At the moment it's done for all timer tasks and all loading tasks. We'll have to estimate duration for each task queue individually, make a high-level decision in the browser process ("should we block long tasks" instead of
- Metrics. Many UMA histograms are tracking total duration per task queue type and renderer state (renderer backgrounded / etc). If we are to continue to have complex logic here, additional metadata about browser state should still be send to the renderer process, but instead of decision making it will be used only for recording metrics.

To make sure that we can make decisions based on renderer load we'll have to architect an efficient way to let the browser process know about load level. Most likely that will involve aggregation (e.g. 10ms chunks) and sending the data back to the browser process on a regular basis (e.g. once a second/10 seconds/minute).