Isolating performance of third-party iframes

haraken@chromium.org (inspired by esprehn@chromium.org)
2016 May 31
STATUS: PUBLIC

Summary: Third-party iframes can cause a big jank on a top-level frame. This document discusses strategies for isolating performance of third-party iframes. We have already invested on iframe-throttling, per-iframe scheduler, TDI etc. This document tries to correlate the efforts in one picture toward the unified goal of removing janks caused by third-party iframes.

Problem

Third-party iframes can cause a big jank on a top-level frame. They may run synchronous document.write, which blocks until the script resource is fetched from the network and completely loaded into the page. They may run JavaScript that takes a couple of seconds. They may load many cross-origin iframes. For example, theverge.com loads 70 different origins (not including sub-origins). The top-level frame must be blocked while a third-party iframe is doing work. The top-level frame cannot handle scrolling, touch input events or other user interactions.

Strategies

Overview

To improve the user experience on the top-level frame, it is important to **isolate performance between the top-level frame and third-party iframes**. More specifically, the engine should provide the following two mechanisms:

(a) A mechanism to de-prioritize tasks of badly performing iframes
(b) A mechanism to interrupt iframe's long-running task by a top-level frame's task

(a) enables the engine to prioritize tasks of the top-level frame. (b) enables the engine to improve responsiveness of the top-level frame by interleaving third-party iframes (i.e., improves "R" of the RAIL). The platform should provide an infrastructure to smartly schedule third-party iframes by combining (a) and (b).

(a) De-prioritize tasks of badly performing iframes

The platform should provide a ***per-iframe task scheduler*** to schedule tasks per iframe (this is a natural extension of Sami's iframe-throttling). Actually we already have the mechanism in WebFrameScheduler, which provides an ability to throttle tasks of iframes that are not in a visible page. The problem is, however, that not all Blink tasks are posted to the WebFrameScheduler. Most Blink tasks are posted by Timers and Platform::current()->mainThread()->getWebTaskRunner(). They post the tasks to the default task queue of the renderer process, not the per-iframe task queue of the WebFrameScheduler. To get full benefit of the WebFrameScheduler, we need to associate the tasks with iframes so that the tasks are posted to the WebFrameScheduler of proper iframes.

Specifically, we need to make the following changes:

- Associate postTasks with frames: We need to replace Platform::current()->mainThread()->getWebTaskRunner()->postTask() with m_frame->frameScheduler()->timerTaskRunner()->postTask().
- Remove Timers: Blink has 70 different Timer members to schedule various things, and they're used pretty randomly. Most of these Timers should be replaced with postTasks associated with frames, idle tasks or requestAnimationFrame().
- Remove EventSenders: Blink has 8 EventSenders but they are a broken scheduling primitive that sends events asynchronously in a batch, causing event listener floods. Since the events are inherently asynchronous, we should just post the events by postTasks.

The goal is to post as many tasks as possible to the per-iframe task queue of the WebFrameScheduler, instead of the default task queue. Once it's done, the next step is to decide a policy to de-prioritize tasks of third-party iframes. The details about the policy is out of the scope of this document, but for example, we can de-prioritize third-party iframes that meet the following conditions. FrameBlamer could be extended to blame those iframes:

- The iframe ran long-running JavaScript.
- The iframe ran synchronous document.write.
- The iframe loaded many cross-origin iframes.

- The iframe triggered many layouts.

The main goal of the de-prioritization is to improve the user experience on the top-level frame. However, the de-prioritization will also work as a way to **give a right level of "penalty" to badly performing third-party contents, giving the third-parties good incentive to improve the contents.** This would be key to keep the ecosystem of the web and the third-party contents. See this document (Google-internal) for more details.

Finally, remember that the per-iframe scheduler is useful regardless of OOPIF (Out-Of-Process IFrame) or TDI (Top Document Isolation).

(b) Interrupt iframe's long-running task

The platform should provide a mechanism to interrupt long-running tasks of third-party iframes. Given that most long-running tasks in the web are JavaScript, the question boils down to **how to provide a mechanism to yield iframe's JavaScript to a top-level frame's task** like this:

1. Run iframe's JavaScript
2. Suspend the iframe's JavaScript
3. Run a task of the top-level frame
4. Resume the iframe's JavaScript

Remember that the yielding doesn't break correctness because third-party iframes are inherently asynchronous per the spec.

There are multiple ways to make JavaScript yieldable:

- **Multi-processing the engine**
  - Idea: Put the top-level frame and third-party iframes in different processes (i.e., ask the OS to interrupt iframe's JavaScript).
  - Pros: We can reuse the infrastructure for OOPIF/TDI.
  - Cons: Memory bloats. It would be hard to ship it to low-end Android devices.
- **Multi-threading the engine**
  - Idea: Put the top-level frame and third-party iframes in different threads (i.e., ask the OS to interrupt iframe's JavaScript).
  - Pros: Memory bloat will be smaller than multi-processing.
  - Cons: It would be hard to multi-thread Blink without introducing any performance regression because the current Blink heavily assumes a single-threaded architecture. Also memory bloat will be still problematic because a renderer process creates many data structures per thread (e.g., CC's texture).
- **Restricted nested message loop**
  - Idea: Run a top-level frame's task inside a nested message loop of iframe's JavaScript. To mitigate the security risk of the nested message loop, define safe

points and restrict places that can fire nested message loops. (See this document for more details)
    - Pros: Memory doesn't bloat. It's easier to implement than multi-processing/threading. We can provide a more fine-grained, user-level scheduling.
    - Cons: It's not trivial to make the restricted nested message loop workable.

All of the approaches have pros and cons and are hard for different reasons. We need more discussion.

How the strategies are related to each other

As described above, we have multiple strategies to isolate performance of third-party iframes. Let me clarify how the strategies are related to each other.

First of all, to isolate performance between the top-level frame and third-party iframes, we must have the following two mechanisms:

(a) A mechanism to de-prioritize tasks of badly performing iframes
(b) A mechanism to interrupt iframe's long-running task by a top-level frame's task

[Per-iframe scheduler] is a solution for (a). **[Per-iframe scheduler] is anyway needed.** [Multi-processing (TDI)], [Multi-threading] and [Restricted nested message loop] are solutions for (b), and these solutions are exclusive (i.e., if we have [Multi-processing (TDI)], we don't need [Multi-threading] or [Restricted nested message loop]). This means that we have three possible combinations of the strategies.

	[Per-iframe scheduler] + [Multi-processing (TDI)]	[Per-iframe scheduler] + [Multi-threading]	[Per-iframe scheduler] + [Restricted message loop]
Schedulability	OS-dependent	OS-dependent	OS-independent (User-level scheduling will provide a better UX)
Security	Improve	Same as today	Same as today
Memory	Regress	Regress	Same as today
Runtime performance of the main thread	Same as today	Regress (Blink assumes a single-threaded architecture)	Same as today
Implementational	Hard (but we've	Hard	Medium

complexity	already invested a lot of efforts)		
Target devices	Desktops + high-end Android?	-	Low-end Android?

Given the above table, we can observe the following things:

- [Per-iframe scheduler] + [Multi-threading] wouldn't be an option.
- If security is a mandatory requirement, we should choose [Per-iframe scheduler] + [Multi-processing (TDI)].
- If memory is a mandatory requirement, we should choose [Per-iframe scheduler] + [Restricted message loop].
- **A practical solution would be to choose [Per-iframe scheduler] + [Multi-processing (TDI)] for high-end devices and [Per-iframe scheduler] + [Restricted message loop] for low-end devices.** (But do we really want to invest on two strategies?)

Metrics

The goal of this project is to improve responsiveness of the top-level frame. The success will be measured by the following metrics:

- Average / Max execution time of uninterruptable tasks in third-party iframes
- Average / Max waiting time of tasks in the top-level frame
- Average / Max execution time of JavaScript in third-party iframes
    - If the de-prioritization works as incentive for third-parties to improve the contents, long-running JavaScript in third-party iframes will decrease in long term.

Action items

Whereas we need more discussion for how to interrupt iframe's long-running tasks, the idea of the per-iframe scheduler looks promising and non-controversial (the per-iframe scheduler is needed no matter option we choose). Here is a working plan:

1. Implement the per-iframe scheduler
2. Add metrics
3. Discuss more how to interrupt iframe's long-running tasks