[PUBLIC] Stop Loading in Background on Android

Authors: panicker@chromium.org

One-page overview

Summary

Loading tasks in blink scheduler and fetching of resources will be stopped when a renderer has been in the background after grace time of 5 minutes, on Android.

Link to Intent to Implement:

https://groups.google.com/a/chromium.org/forum/#!topic/blink-dev/EQJX2aGVeUU

Platforms

Mobile: Android.

Team

panicker@google.com, fmeawad@google.com

Bug

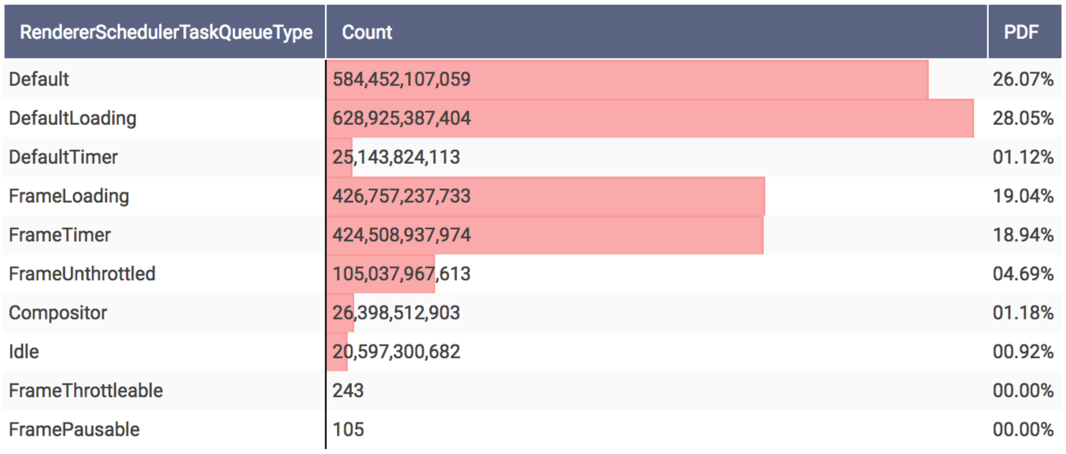
https://bugs.chromium.org/p/chromium/issues/detail?id=775761

Code affected

The condition is triggered in Blink Scheduler and loading task queue is stopped there. Also the signal is sent to ResourceLoadScheduler (Blink) which stops loading of resources.

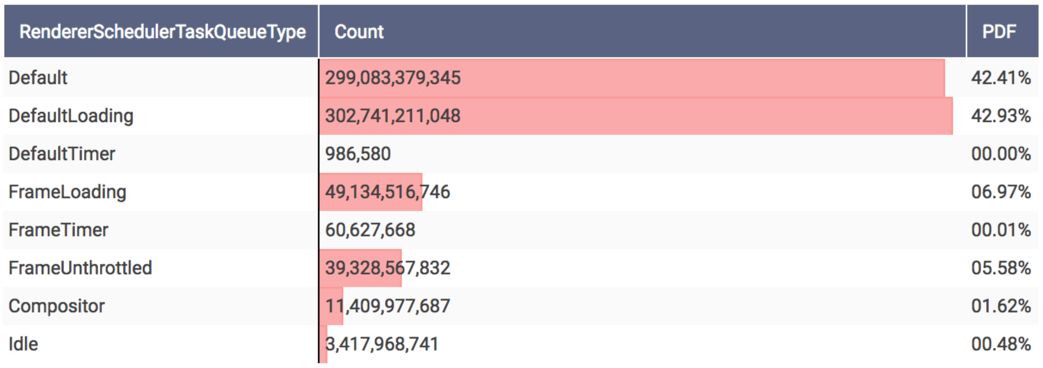
Motivation

Many sites do not stop loading activity, even after their app has been in background for over 5 minutes (a generous grace time), this consumes non-trivial amount of CPU and hurts the responsiveness of foreground app.
Loading task queue is the largest contributor to work in background: [see UMA](see UMA).
See “DefaultLoading” and “FrameLoading” task queues below:



RendererSchedulerTaskQueueType	Count	PDF
Default	584,452,107,059	26.07%
DefaultLoading	628,925,387,404	28.05%
DefaultTimer	25,143,824,113	01.12%
FrameLoading	426,757,237,733	19.04%
FrameTimer	424,508,937,974	18.94%
FrameUnthrottled	105,037,967,613	04.69%
Compositor	26,398,512,903	01.18%
Idle	20,597,300,682	00.92%
FrameThrottleable	243	00.00%
FramePausable	105	00.00%

After 5 minutes in background:



RendererSchedulerTaskQueueType	Count	PDF
Default	299,083,379,345	42.41%
DefaultLoading	302,741,211,048	42.93%
DefaultTimer	986,580	00.00%
FrameLoading	49,134,516,746	06.97%
FrameTimer	60,627,668	00.01%
FrameUnthrottled	39,328,567,832	05.58%
Compositor	11,409,977,687	01.62%
Idle	3,417,968,741	00.48%

Under the umbrella of Web Lifecycle we will stop background work in cases where it would improve the foreground user experience -- starting with mobile (and eventually followed by desktop).
The motivation for this is to:
- improve the responsiveness of the foreground tab
- conserve data (on data connections
- conserve device battery life

Design

Implementation:

- Stop loading task queue in Blink Scheduler:
when threshold (5 mins) have elapsed in background, stop the loading task queue.
https://chromium-review.googlesource.com/c/chromium/src/+/653493

- Stop loading resources:
send the “stopped” signal to ResourceLoadScheduler, which in turn will fully throttle resource loading.
https://chromium-review.googlesource.com/c/chromium/src/+/667449

Metrics

TODO(panicker): Update with results so far:
https://uma.googleplex.com/p/chrome/variations/?sid=4ecb8fa410c363f10c30cbf6cf8cb960

Success metrics

We will watch the following success metrics:

1. Reduced CPU usage in background

NOTE Jun 6: UMA on Stable is looking good.
SUM moved by 14.9%.

NOTE Apr 16 based on this UMA:
Based on RendererScheduler.TaskDurationPerQueueType2.Background.AfterFifthMinute - overall task duration CPU time is reduced, most task queues are doing less work, in particular FrameLoading is statistically significant.
(there is an increase in IPC queue - not statistically significant, but still; we are investigating this)

2. Reduced data usage

NOTE Jun 6 Stable UMA:
DataUse.TrafficSize.User.Downstream.Background.Cellular -5.54%
DataUse.TrafficSize.User.Downstream.Foreground.Cellular - 1.12%
DataUse.TrafficSize.User.Upstream.Background.Cellular +1.78% ?
DataUse.TrafficSize.User.Upstream.Foreground.Cellular -1.38%

NOTE Apr 16 Beta
UMA for cellular data usage
UMA for non-cellular data usage
NOTE: these UMAs are broken down by chrome itself being in foreground / background (and not individual tabs), therefore both foreground and background are relevant.
DataUse.TrafficSize.User.Upstream.Background.Cellular: 3.4% reduction in sum
DataUse.TrafficSize.User.Downstream.Background.Cellular: 3.3% reduction in sum
DataUse.TrafficSize.User.Downstream.Foreground.Cellular: 1.9% reduction in sum
DataUse.TrafficSize.User.Upstream.Foreground.Cellular: 3.5% reduction in sum
DataUse.TrafficSize.User.Upstream.Background.NotCellular: 19.3% reduction in sum and 5.9% reduction in count
PageLoad.Experimental.Bytes.Total is too general to capture background / 5mins case.

"Multi-tab loading" for foreground

NOTE Apr 16 (Beta) based on UMA:

- Much improved FCP & FMP for foreground tab when there are 2+ tabs loading. (unfortunately we don't have an UMA for 2+ tabs existing). eg.
  - FCP improved by 11.5% at median and 9.5% at 95th percentile
  - total count for FCP increased by 18.3%
- Time to switch to background tab has improved when there are 2+ tabs loading:
  - FMP improved by 3.7% at median and 6% at 95th percentile
  - total count for FMP increased by 10.3%
- Increased sample counts indicate that the case of 2+ tabs loading simultaneously is better supported and there is less abandonment.

Scroll Latency and Responsiveness for foreground renderer

Event.Latency.ScrollBegin.Touch.TimeToScrollUpdateSwapBegin2
RendererScheduler.ExpectedTaskQueueingDuration (aka EQT)
NOTE Apr 16: These did not move. Again we don't have UMA for multi-tab case which may have helped.

Regression metrics

If the user were to switch from foreground to a background tab (that underwent this intervention) their time to interaction should not regress much.
This metric is under development, however the following related metrics can be used:
PageLoad.Experimental.PaintTiming.ForegroundToFirstMeaningfulPaint
PageLoad.PaintTiming.ForegroundToFirstContentfulPaint

NOTE Mar 16: These have indeed not regressed:
https://uma.googleplex.com/p/chrome/variations/?sid=78c4de0978708413c916e445ff97f6fd

Experiments

Finch flag CL

Rollout plan

Standard experiment-controlled rollout with flag:
“stop-loading-in-background”

Intervention Guidelines

Predictability:
The mechanism is well defined and predictable, same as stopping timers in background. (The behavior will be eventually spec’d as part of Lifecycle)

Avoidability:
Continuing to be loading in background after 5 minutes, is not a good practice, especially on mobile -- for data and battery (in addition to responsiveness of foreground app).

Transparency:
We are working on firing a JS callback to notify the app when the page transitions to STOPPED state in background. We are looking into firing a console message or notification via Reporting API.

Data-driven:
As mentioned, current data indicates significant work in background renderers due to loading, even after 5 mins.
We will use our success and regression metrics to inform moving forward with this intervention.

Privacy considerations

N/A

Testing plan

Added unit-tests.
Manually tested.

There isn’t much here for local testing, the main thing is getting feedback from the experiment regarding what real world websites it is breaking (this cannot be determined from local testing).

Followup work

TODO.