Time-based renderer task throttling

altimin@chromium.org
August 19th, 2016
(for context: motivation)

This document outlines proposed infrastructure for renderer task throttling enabling Chromium developers to implement throttling interventions and one example of such interventions — background tab throttling.

Motivation

There's been a history (1, 2, 3, 4, 5) of some websites using up to 15-20% CPU core time when idle in background due to heavy ads and analytics scripts. That hurts performance, battery and user experience and justifies a number of different interventions on user's behalf. Due to variety of possible interventions, our goal should be to build an API which enables Chromium developers to do it easily.

API

Proposed API groups task queues in pools and setting a time budget for a pool. Corresponding object is called TimeBudgetPool and has APIs to change budget and add or remove task queues. (TaskQueue-to-TimeBudgetPool mapping is many-to-one for v1, but can easily be changed to many-to-many when need arises).

```
class ThrottlingHelper {
  TimeBudgetPool* CreateTimeBudgetPool();
}

class TimeBudgetPool {
  void AddQueue(TaskQueue* queue);
  void RemoveQueue(TaskQueue* queue);

  void SetTimeBudgetAsWallTimePercentage(double cpu_percentage);
  void SetMaxThrottlingTime(base::TimeDelta max_throttling_time);

  void Enable();
  void Disable();
}
```

Throttling mechanics

Each TimeBudgetPool has mana points level inside it. When this level is negative, new tasks are forbidden to run and throttled. This level regenerates with time at constant rate `cpu_percentage` seconds per second `(0 <= cpu_percentage <= 1).` This results in one renderer using at most `cpu_percentage` percent of cpu time asymptotically.

However, in order to avoid starving pages for too long (with 0.01 budget and one expensive task of 1s being run, page will be blocked for 100s under rules above), one task is guaranteed to run every `max_throttling_time` seconds. Note that does not replenish mana level — tasks will continue to run on one-per-`max_throttling_time` basis until mana level becomes non-negative.

Things above will apply only to the queues which are marked as throttled by ThrottlingHelper (queues from background and offscreen frames).

Background renderer throttling

This API will be accompanied by a intervention: throttling all throttleable task queues from a renderer with a single TimeBudgetPool. The proposed constants are 1% cpu budget per renderer and one task is being run at least once every 15s.

Intervention guidelines

Predictability: Throttling mechanism is well-defined. High-level description (only pages with significant CPU load are going to be impacted).
Avoidability: Heavy computations on UI thread is not best practice, so only bad-behaved pages are going to be impacted.
Transparency: Console log message is generated when tasks we throttled for non-trivial amount of time (small throttling is not reported because we already align timers in background tabs). Also developer can monitor start time of tasks and detect intervention programmatically.
Data-driven: Main UMA metric that we are monitoring is BackgroundMainThreadLoad. It is percentage of time that renderer spends executing javascript tasks. Given that >90% of pages have low CPU usage, we are expecting changes to high percentiles (95, 99), not average (95 percentile is 10% load, 99 percentile is 90% load).

Future plans and sidenotes

- Task runtime is understood as wall time here, instead of thread time. This skews task runtimes under heavy load, but this results in extra throttling when system is slow and considered a feature.
- Wakeup budget is planned — TimeBudgetPool will also have a budget of wakeups — wakeup is allowed once per N seconds.
- Network fetches can be a sign of a need to allow page to run. IPC queue may or may not be a good source of information for this.
- UserEngagementScore can be used to increase time budget for pages the user is interested in.