Blink Idle Task Scheduling

rmcilroy@ et al

(PUBLIC)

Introduction

The Blink Scheduler provides the ability to decide which tasks get to execute on the main thread at any given time. This enables prioritization of latency sensitive tasks (e.g., input events or compositor updates), but it also provides the opportunity to schedule best-effort work which need only be performed *when time allows*, without impacting higher priority work. This document discusses the design of idle task scheduling in the Blink scheduler, with specific focus on implementing V8's idle notification callbacks using the proposed design.

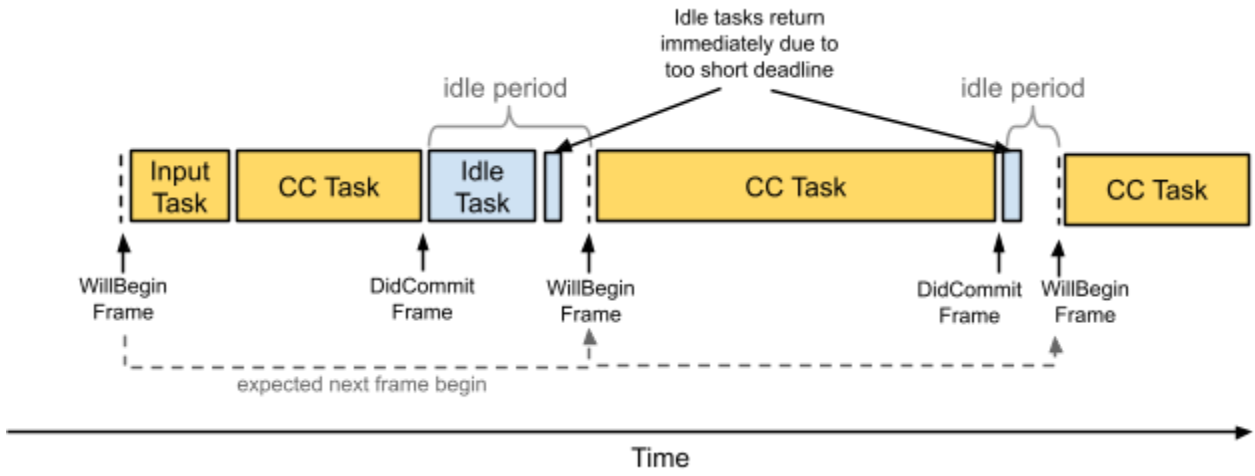
Proposed solution

The Blink Scheduler enables tasks to be posted to different *task types (e.g.,compositor tasks, generic tasks, etc),* which enables the scheduler to decide which type of task should be executed at a particular time. Idle tasks will be a separate task task type with some specific properties:

- These tasks are explicitly best-effort, in that they may take an arbitrarily long time before being executed if the system is otherwise busy.
- The tasks can be re-ordered with respect to other task types posted to the scheduler.
- The tasks are passed a deadline, provided by the scheduler, as an argument to their run method. The task is expected to finish before this deadline.
- The tasks may not be able to do any real work before the deadline expires, in which case it must return immediately and may optionally repost itself to try again on the next idle period.

The scheduler maintains the queue of pending idle task and will schedule these tasks during idle periods of execution. What constitutes an idle period of execution is something which the scheduler is free to decide. Initially the scheduler will use notifications from the compositor about frame begin and commit events, as well as it's knowledge of other higher priority tasks currently pending, to schedule idle events at times when they will not cause jank or increase input latency. Future work will enable longer idle periods when no frames are being committed by the compositor.

During an idle period, the scheduler will take the oldest task from the pending idle queue, and schedule its execution with a deadline which is less than or equal to the remaining idle period time. If the task completes before this deadline then the scheduler may continue execution of idle tasks in FIFO order until the deadline.



Example execution schedule

Detailed Design

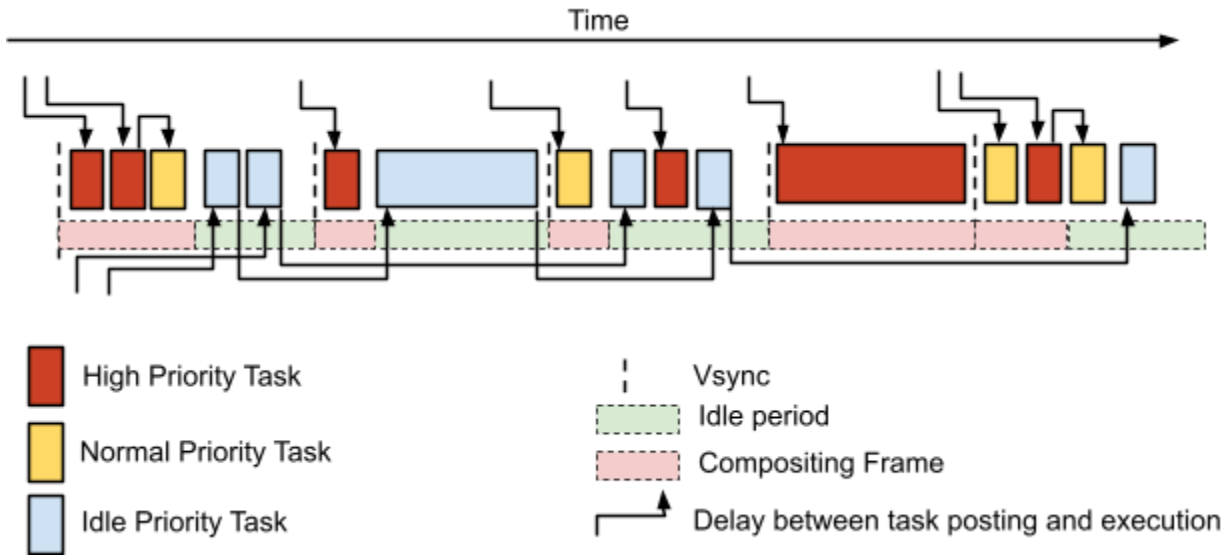
Blink components can post idle task to the scheduler for later execution. These tasks are one-shot tasks, which are have the type WTF::Function<void(double deadlineSeconds)> - i.e., they are function continuations which have one unbound double argument, which will be bound at task execution time by the scheduler to provide the task with a fixed deadline. An idle task will only be executed once, and when executed the task is expected to decide whether it can do any useful work in the time allowed before the deadline expires. If no useful work can be done, the task should return immediately, reposting itself to the idle queue if it wishes to retry. The majority of idle tasks will be executed between frames, therefore the deadline will usually be less than 10ms. A client should not post an idle task unless they expect to be able to do meaningful work in this timeframe, while being adaptable enough to make use of longer deadlines if available.

Posted idle tasks will enter the scheduler and be appended to an incoming idle task queue. At the beginning of a new idle period, these incoming tasks will be flushed to the pending idle task queue, where the scheduler will execute them in a FIFO manner. Having a separate incoming queue which is only flushed to the pending queue at the start of an idle period means that idle tasks can safely re-post themselves during their own execution, even if they could do no real work before the deadline expired. Without this separation, an idle task could cause a flood of

reposting towards the end of an idle period, where it would decide it couldn't do useful work in the current deadline, repost itself, then get immediately rescheduled in the same idle period.

The scheduler will use various signals to decide when idle periods should begin and end. The initial implementation will use the input from the compositor to ensure that idle tasks are only scheduled between the time when a frame has been committed, and the time when the next frame is expected to begin. This limits idle periods to inter-frame times and means that no idle periods will occur when the compositor is not active (due to frames not being drawn). Therefore, a future extension will involve the scheduler posting a delayed task which will trigger an idle period if no frames have been drawn for a period of time. During these non-compositor initiated idle periods we may provide longer idle deadlines, however this will need to be balanced with a mechanism to ensure that a long idle task cannot unduly delay any critical work which comes in during this period. This may involve either splitting longer idle periods up into shorter idle periods with shorter delays, or exposing the scheduler's shouldYieldForHighPriorityWork() to these tasks and expecting idle tasks to call this function regularly and yield if appropriate.

During idle periods the scheduler is free to schedule higher priority tasks (e.g., cc or input tasks) in preference to idle tasks, however it is only free to schedule idle tasks during the idle periods.



Scheduling of idle tasks within frame compositing times

Integration with V8's Idle Notifications

The initial use-case for idle tasks is the V8 idle notification API, which triggers incremental garbage collection (can be linearly configured from 0ms-XXms), scavenges (usually ~5-10ms) and long full garbage collections (30-XXXms) in V8. By doing this work during idle time, V8 can avoid GC jank during critical path JavaScript execution.

The idle notification call takes an argument specifying the number of milliseconds for which it can run, and aims to always return before this time. Currently the compositor issues idle notification calls at frame commit time (see Compositor Driven Garbage Collection). While the compositor has knowledge of when frame are scheduled, it does not have global knowledge of all tasks that need to be executed, nor can it decide to schedule a call to an idle notification when the compositor is not active (e.g., during long idle periods when no frames are being drawn), which would be the ideal time to do long-running GC work. The Blink Scheduler has more knowledge about global task execution which means it should be able to do a better job of scheduling idle notifications, both to avoid delaying critical work, and to provide more opportunities for scheduling the work during real idle times.

The integration of the V8 idle tasks will involve blink posting an initial V8 idle task to the blink scheduler when it initializes V8. This idle task will call the idleNotification API, passing an appropriate idle millisecond value based on the task's deadline. The task will then repost itself back on the idle queue so that it is executed on the next idle period.

In the future, we will use the return value of the idle notification call to decide whether there is any further idle work that V8 can do, and only repost the idle task if necessary. This will require a mechanism to enable us to post the idle task when V8 does any additional JS execution to enable the resumption of idle notifications when there may be more work which can be done.

Implementation

The following CLs implement this proposal and hook it up to the V8 idle notification task scheduling:

- Scaffolding for idle tasks in the scheduler: https://codereview.chromium.org/410143002/ (landed)
- Scheduling of idle tasks: https://codereview.chromium.org/595023002/ (landed)
- Support reposting of idle tasks: https://codereview.chromium.org/640053003/
- Enable V8 idle tasks: https://codereview.chromium.org/639773007/

Benchmarks

We should track the the root mean square of time spent in GC that was not scheduled - aiming to increase the proportion of GC work which was has been moved off of the critical path into the scheduler's idle period. We should also track mean_frame_time, jank and queueing_delay to ensure do not regress (and hopefully improve) these metrics.

The following benchmarks should be tracked:

- Compositor + ACDCGC workload [to be added to dashboard]
- Smoothness benchmarks
- top-10-slowest-tasks [to be added to dashboard]

References

- Blink Scheduler Design Doc
- Compositor Driven Garbage Collection