Resource Loading tasks and the Blink Scheduler

Tracking bug: http://crbug.com/391005

Prototype patches:

https://codereview.chromium.org/655603002/ (Resposting resource dispatcher tasks)
https://codereview.chromium.org/663453004/ (at IPC layer)
https://codereview.chromium.org/648403004/ (as an IPC Filter)

Motivating traces:

LA Times, The verge, FTW usa today, Pinterest

Introduction

On resource heavy sites, quite a lot of time on the render thread can be spent processing resources and the current incarnation of the blink scheduler can only reorder tasks that are posted through it. To help improve queuing durations and input latency, when the user is interacting with the device, we would like to post some or all main thread resource dispatcher tasks through the blink scheduler.

Initial profiling suggests there are some significant wins to be had on some sites.

Design

We propose to add a ResourceSchedulingFilter which will dispatch resource messages on a nominated SingleThreadedTaskRunner. Thanks to the existing filtering logic we can make sure this filter only runs for resource messages by overriding GetSupportedMessageClasses and returning ResourceMsgStart.

To ensure correct behavior, this needs to be the last one in the chain. At first blush this seems to be challenging since ResourceDispatcher::AttachThreadedDataReceiver adds yet another filter. Fortunately due to the way MessageFilterRouter::TryFilters is implemented, global filters (without an overridden GetSupportedMessageClasses) are tried first, so the desired ordering is preserved.

A unit test will be added to guard against this changing.

LoadingTaskQueue

I propose to add the concept of a LoadingTaskQueue. When the scheduler is in kNormalPriorityPolicy, tasks posted on the LoadingTaskQueue will be treated as high priority. The reason for this, is we may be able to avoid some unnecessary (and likely expensive) page layouts if we process resources first (some initial profiling suggests we may see an 8%

reduction in page load times on mobile). When the scheduler is in kCompositorPriorityPolicy, however, the LoadingTaskQueue will be marked as kBestEffortPriority which deprioritizes them. I suspect we may eventually add a kLoadingPriorityPolicy which would prioritize them.

Is reordering of the resource dispatcher safe?

This is a difficult question to answer. The resource dispatcher messages are not supposed to have any ordering dependencies with other types of messages.

To try and answer this question I developed a fuzzing scheduler policy (which selects a random queue to run) and routed the IPC tasks through their own queue. Then I ran the layout rests a few times through the bots to see if anything suddenly failed or became flaky. The idea being that this should increase the task reordering and hopefully shake out any issues.

Alternatives considered:

- Reposting all resource dispatcher tasks. I tried this in patch 655603002, however that appears to have introduced a race condition (likely between the task filters, the reposted tasks and the transition from foreground to background loading) which manifested in some of the blink layout tests becoming flaky. I experimented with getting the filter tasks to repost themselves. This appeared to reduce the flakiness somewhat but it made the patch significantly more complex.
- Reposting only the ResourceDispatcher's OnRequestComplete task. This should be simpler and safer than reposting all the resource dispatcher tasks and it's usually the most expensive call (due to JS execution) representing between 40% - 60% of the resource dispatcher runtime. Of course we would not be able to prioritize compositor tasks over the remaining resource dispatcher tasks which limits the gains. This would be particularly noticeable on pages like the LA Times where OnRevievedResponse is more expensive than OnRequestComplete.
- Teach ChannelProxy to spot loading tasks based on the IPC Message's type (using a hashset) and post these on a nominated SingleThreadedTaskRunner rather than on the thread's default one. This guarantees that all resource dispatcher tasks will be executed in order with respect to themselves.

  To support this, ChannelProxy would have two new public methods:

  ```
  // Used to allow the scheduler to prioritize loading IPC tasks with
  // respect to other tasks. Only tasks specified via
  // UseLoadingTaskRunnerForTaskID will be posted on this task
  // runner.
  void SetListenerLoadingTaskRunner(const scoped_refptr<
      base::SingleThreadTaskRunner>& listener_loading_task_runner);
  ```

```
// Specifies that IPC messages with task_id should be dispatched on
// the task runner specified via SetListenerLoadingTaskRunner.
void UseLoadingTaskRunnerForTaskID(uint32 task_id);
```

I prototyped this in patch https://codereview.chromium.org/663453004/. It appears to work, although architecturally it feels wrong to add what is effectively another type of filter.

- Adding a separate SyncChannel to the render thread just for resource dispatcher tasks wired up to the scheduler, and using routing ids to send the resource dispatcher messages to the right place. This would likely result in a larger patch because all the message senders need to set the routing id, as well as having changes inside render_thread_impl.cc & .h. It would be easy to forget to set it in new code, potentially leading to hard to diagnose order of execution bugs.