Blink Spatial Scheduling DRAFT

{alexclarke,skyostil}@chromium.org   Tracking bug: http://crbug.com/510398

(PUBLIC)

Goal

Reduce Jank, first gesture scroll update latency, and queueing durations by allowing the scheduler to defer task execution of tasks that relate to offscreen-content.

Introduction

While a lot of work has already been done* to schedule tasks on the Blink Main thread, there are still sites where important work gets bogged down behind work for things that are offscreen. Currently the blink scheduler has no idea about where on the page tasks come from. If we could tell it which frame tasks where associated with, then it would be able to de-prioritize tasks associated with offscreen frames.

* Previous work: Initial Blink Scheduler; Resource Loading Optimizations; Scheduling JS Timer Execution

Design

We shall introduce the concept of a `WebFrameScheduler`, which associates loading and timer `WebTaskRunners` with a blink `LocalFrame`. The `WebFrameScheduler` API will let blink tell the scheduler if the frame is on a background tab, or if it's offscreen.   The `WebURLLoader` will be given an api to set the `WebTaskRunner` its tasks run on. Various Blink classes will be changed to the new API, including:

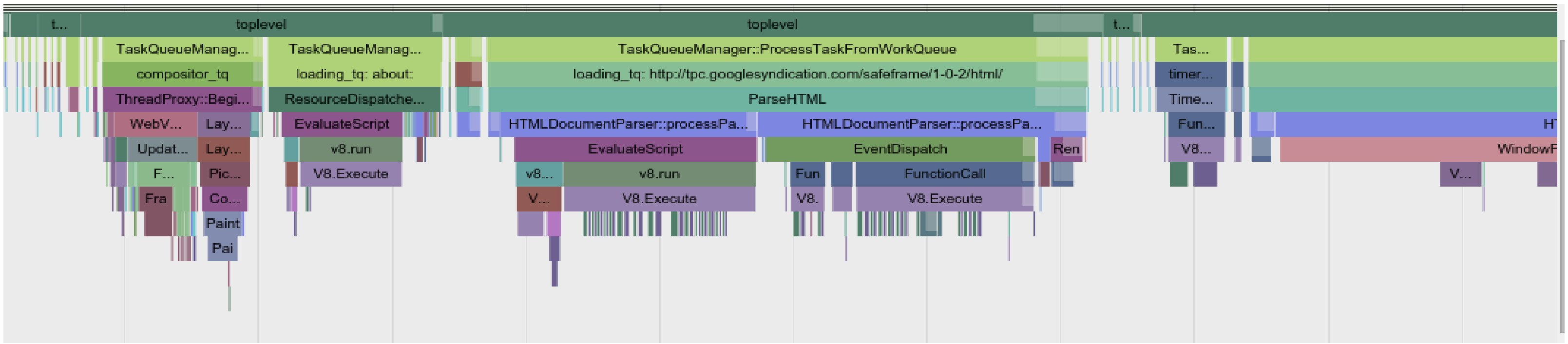
1. TimerBase
2. DOMTimer
3. LocalDOMWindow
4. ScriptLoader & ScriptRunner
5. V8 ScriptStreamer
6. ResourceLoader & ResourceFetcher
7. FrameLoader & FrameFetchContext

Most of the changes required will be pretty mechanical, just plumbing through the new WebTaskRunner, however we envisage doing something more complex for ScriptRunner.

Currently ScriptRunner acts as a mini-scheduler with it's own queue of scripts that are ready to execute. We propose to make the following changes, which will simplify it a lot:

- Change ScriptRunner::notifyScriptReady to post a loading task for ready async scripts on the corresponding frame's loading task queue.
- Change ScriptRunner::notifyScriptReady to post a loading task on the default loading task queue for all inorder scripts that are ready.
- Remove ScriptRunner::executeScripts.

In general, the more tasks we move off the default task queues the better. Not only will this enable the scheduler to make better decisions, it will also improve traces because it will now be possible to see which origin tasks came from.



API changes:

```
class BLINK_PLATFORM_EXPORT WebTaskRunner {
public:
    virtual ~WebTaskRunner() {}

    // Schedule a task to be run on the the associated WebThread.
    // Takes ownership of |WebThread::Task|.
    // Can be called from any thread.
    virtual void postTask(
        const WebTraceLocation&, WebThread::Task*) {}
    // Schedule a task to be run after |delayMs| on the the
    // associated WebThread. Takes ownership of |WebThread::Task|.
    // Can be called from any thread.
    virtual void postDelayedTask(
       const WebTraceLocation&, WebThread::Task*,
       double delaySecs) {}

#ifdef INSIDE_BLINK
    // Helpers for posting bound functions as tasks.
    typedef Function<void()> Task;

    void postTask(const WebTraceLocation&, PassOwnPtr<Task>);
    void postDelayedTask(
        const WebTraceLocation&, PassOwnPtr<Task>,
        double delaySecs);
#endif
};

class BLINK_PLATFORM_EXPORT WebFrameScheduler {
public:
    virtual ~WebFrameScheduler() {}

    // Used to name the loading and timer task queues.
    virtual void setOrigin(const char* origin) { }

    // If the page associated with the frame is in the background
    // then it’s task queues will be throttled.
    virtual void setPageInBackground(bool) { }

    // If a cross-origin frame is not currently visible then it’s
    // task queues will be throttled.
    virtual void setFrameVisible(bool) { }

    // If a frame is never visible (for example it has zero size)
    // then it won’t be throttled due to to frame visibility.
    virtual void setFrameNeverVisible(bool) { }

    // Tells the scheduler if this is an same-origin or cross-origin
    // iframe.  Cross origin iframes may be throttled due to
    // frame visibility.
    virtual void setIsCrossOrigin(bool) { }

    // Returns the WebTaskRunner for loading tasks.
    // NOTE the WebFrameScheduler owns the returned WebTaskRunner.
    virtual WebTaskRunner* loadingTaskRunner() { return nullptr; }

    // Returns the WebTaskRunner for timer tasks.
    // NOTE the WebFrameScheduler owns the returned WebTaskRunner.
    virtual WebTaskRunner* timerTaskRunner() { return nullptr; }
};

class BLINK_PLATFORM_EXPORT WebScheduler {
public:
    ...

    // Returns the default WebFrameScheduler.  Owned by WebScheduler.
    virtual WebFrameScheduler*
    defaultWebFrameScheduler() { return nullptr; }

    // Returns a new WebFrameScheduler.  Owned by the caller.
    virtual WebFrameScheduler*
    createWebFrameScheduler() { return nullptr; }

    ...
};


class WebURLLoader {
public:
    ...

    // Sets the WebTaskRunner to be used when fulfilling requests.
    virtual void setLoadingTaskRunner(WebTaskRunner*) {}
    ...
};
```

Timer Queue Throttling

Previously Blink throttled timers for background tabs via the DOMTimerCoordinator which aligned all the registered timers to the nearest second. Now that DOMTimers are posted to chromium, where task cancellation isn't supported, this is no longer a good method of throttling (because we can end up with a large proportion of timers on the queue being cancelled).   Instead we propose to implement it on the chromium side. Idle task queues are manually pumped, which lets the IdleHelper control when idle tasks are run. We shall use the same approach in a new ThrottlingHelper which will mark throttled queues as manually pumped, and it will run a task on the control queue once a second to pump all throttled queues. This should be considerably more efficient than the current method, and should work well for timers, but it's not so well.

Loading Queue Throttling

The loading queue has quite different workload characteristics to the timer task queue, because rather than a timer posting another timer, we get large numbers of tasks posted in response to IPCs. The throttling scheme above, which works well for timers, doesn't work well here since throttling execution doesn't slow down the rate at which the loading tasks get posted.

Instead we propose to mark loading queues for hidden frames as best effort priority, and to turn them on only during idle periods. These tasks will still run, but at a much reduced rate unless the system is quiescent. If the tasks are short then they won't induce jank.

Throttling off-screen frames

We would like to throttle loading and timer queues for cross-origin iframes that are not currently visible, but could be if the page was scrolled.   I.e. zero sized iframes won't be throttled, but an ad (for example) that's not visible would be.

Open questions:

- Is it always safe to throttle cross-origin iframes if they have a non-zero size?
- Do we need to treat iframes that are only just off screen differently from ones that are well off-screen? E.g. should they be processed faster than 1Hz?

Detecting frame visibility

skyostil@ is adding a `ViewportVisibilityObserver` to `LocalFrame` we shall add an observer to `LocalFrame` which will proxy the visibility signal to the `WebFrameScheduler` associated with the `LocalFrame`.   See his prototype here: https://codereview.chromium.org/1246173002

A more likely option is to build the internal infrastructure needed for PositionObserver and reuse that for visibility detection here.

Scaling the Blink Scheduler to hundreds of tasks queues

Several algorithms in the scheduler had linear complexity with respect to the number of task queues. These have now been changed, TaskQueueSelector::SelectQueueToService has O(log max sum of queues with a given priority) complexity, and TaskQueueManager::UpdateWorkQueues is O(sum of queues with a non-empty incoming queue) which is typically only a couple of queues.

Compatibility Risk

We want to make sure we don't break things! There are various risks associated with throttling frames that we are aware of and wish to avoid.

Audio playback

We don't want to throttle background tabs with audio playback. I don't believe we need to do anything special there for background tabs, since the code that calls Page::setVisibilityState should already take care of that. However it's possible we may need to plumb something in so we can know if a frame has audio playback so we can avoid throttling that when it's offscreen.

Same-origin iframes

GMail and various other sites do processing in invisible same-origin iframes. We likely can't safely throttle these.

Spatial Scheduling Prototype

We have built a working prototype here: Chromium Patch, Blink Patch

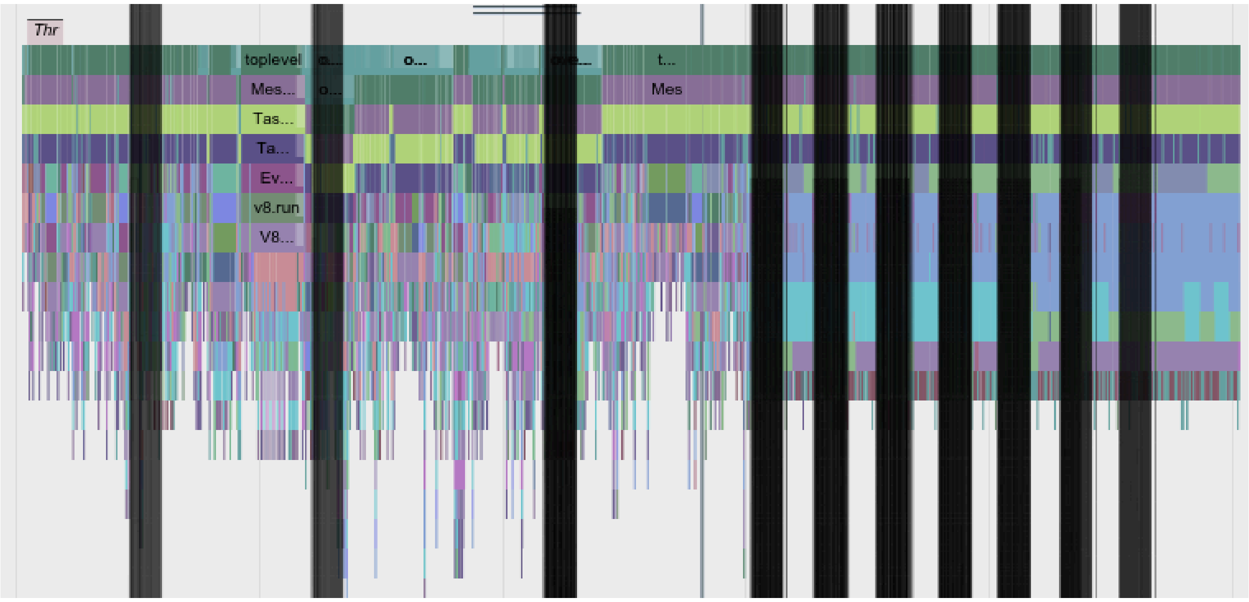
Metrics and performance measurement

Our understanding of the causes of main thread jank has evolved over the past year and most of the time the only thing that matter matters is how responsive the main thread is when you want to start scrolling. Our existing telemetry pagesets have two issues that makes them less useful for measuring the impact of this work:

1. They wait till the DOMContentLoaded event fires before scrolling fires. This is unfortunate because the worst user visible jank occurs while loading.
2. They only scroll once which is unrealistic.

To remedy this skyostil@ developed the smoothness.scrolling_tough_ad_cases metric which aims to scroll the page 10x in short succession. The initial version of this had a problem illustrated by the picture below. The black lines are flow events and show where it's trying to scroll. As you can see they are not evenly distributed, this was for two reasons:

1. The DevToolsAgent IPCs to evaluate JS where run a normal priority on the main thread and getting stuck behind other work.
2. Each mini scroll is a separate ScrollAction which was doing the setup every time resulting in unnecessary roundtrips through the main thread.



Fixing these issues results in a fairer benchmark, with more evenly distributed scrolls.

TODO add picture.

Breakdown of render thread task location and running times for various websites

In each case the site's homepage was traced while loading, and between 10 and 12s of data was recorded. Note in this test there was (mostly) no scrolling and I had disabled the throttling of offscreen frames. The green rows denote cross origin iframes that we could potentially throttle.   It's quite apparent that some sites will be helped more by spatial scheduling than others, androidpolice.com and forbes.com in particular look ike they'll be helped (this does appear to be the case in pratice too).

androidpolice.com trace

Task Queue	#Tasks	Time MS
loading_tq: about:	151	2902
loading_tq: http://www.androidpolice.com/	567	1229
loading_tq: https://googleads.g.doubleclick.net/pagead/	88	1151
timer_tq: http://www.androidpolice.com/	13	1044
compositor_tq	108	567
timer_tq: about:	130	278
loading_tq: data:text/	63	258
timer_tq: default	274	241
loading_tq:	32	102
loading_tq: https://c.betrad.com/	12	69
loading_tq: https://s1.2mdn.net/ads/richmedia/studio/pv2/37110835/20150626072720943/	5	36

loading_tq: http://googleads.g.doubleclick.net/pagead/html/r20150729/r20150730/	8	35
default_tq	167	30
timer_tq: https://googleads.g.doubleclick.net/pagead/	55	29
default_loading_tq	542	28
idle_tq	7	8

forbes.com trace

Task Queue	**#Tasks**	**Time MS**
loading_tq: http://www.forbes.com/	2095	5515
compositor_tq	275	1722
loading_tq: http://www.wikinvest.com/partner/Forbes/oauth/api/	177	1026
loading_tq: http://c.brightcove.com/services/viewer/	159	859
timer_tq: http://www.forbes.com/	681	568
timer_tq: http://c.brightcove.com/services/viewer/	16	380
timer_tq: default	691	334
loading_tq: default	2	316
loading_tq: http://admin.brightcove.com/viewer/us20150514.1307//html/	31	177
default_loading_tq	1190	88
loading_tq: http://loadus.exelator.com/load//	37	63
default_tq	155	34
timer_tq: http://www.wikinvest.com/partner/Forbes/oauth/api/	12	32
timer_tq: http://admin.brightcove.com/viewer/us20150514.1307//html/	3	31

loading_tq: http://tpc.googlesyndication.com/safeframe/1-0-2/html/	7	24
idle_tq	32	22

theverge.com trace

Task Queue	#Tasks	Time MS
loading_tq: http://www.theverge.com/	2112	5033
compositor_tq	507	2956
timer_tq: http://www.theverge.com/	404	1692
loading_tq: http://tpc.googlesyndication.com/safeframe/1-0-2/html/	86	544
loading_tq: about:	10	178
timer_tq: http://tpc.googlesyndication.com/safeframe/1-0-2/html/	102	109
loading_tq:	93	10
timer_tq: default	230	67
default_tq	194	45
loading_tq: default	1	39
default_loading_tq	548	30
idle_tq	13	22
timer_tq: about:	32	14

tmz.com trace

Task Queue	#Tasks	Time MS
loading_tq: http://www.tmz.com/	1642	4014
compositor_tq	137	740

timer_tq: http://www.tmz.com/	420	385
loading_tq: https://www.youtube.com/embed/	66	320
timer_tq: default	287	128
default_loading_tq	1226	65
loading_tq: http://tpc.googlesyndication.com/safeframe/1-0-2/html/	7	46
loading_tq: http://platform.tumblr.com/v1/	14	41
default_tq	136	23
loading_tq: about:	6	22
idle_tq	62	19

latimes.com trace

Task Queue	**#Tasks**	**Time MS**
compositor_tq	751	4558
timer_tq: http://www.latimes.com/	106	3918
loading_tq: http://www.latimes.com/	1228	2336
loading_tq: default	2	543
loading_tq: about:	37	493
timer_tq: default	107	141
loading_tq: http://tpc.googlesyndication.com/safeframe/1-0-2/html/	7	48
default_tq	156	30
default_loading_tq	438	26
idle_tq	7	16
timer_tq: about:	4	6
loading_tq:	3	2

mashable.com trace

Task Queue	#Tasks	Time MS
loading_tq: http://mashable.com/	849	6383
compositor_tq	346	1099
timer_tq: http://mashable.com/	130	704
loading_tq: http://meerkatapp.co/btn/	64	571
loading_tq: https://platform.tumblr.com/v2/	42	310
loading_tq:	48	232
loading_tq: https://accounts.google.com/o/oauth2/	25	131
loading_tq: about:	6	52
default_loading_tq	799	47
idle_tq	23	35
default_tq	194	32

theguardian.com trace

Task Queue	#Tasks	Time MS
loading_tq: http://www.theguardian.com/	2116	3809
compositor_tq	225	2404
loading_tq: about:	83	1097
timer_tq: http://www.theguardian.com/	136	450
timer_tq: default	136	148
timer_tq: about:	54	52
loading_tq: http://tpc.googlesyndication.com/safeframe/1-0-2/html/	7	49

default_tq	205	37
default_loading_tq	594	31
loading_tq: http://ib.adnxs.com/	17	28
idle_tq	21	3
control_tq	8	1

telegraph.co.uk trace

Task Queue	**#Tasks**	**Time MS**
loading_tq: http://www.telegraph.co.uk/	1316	5911
compositor_tq	159	594
timer_tq: http://www.telegraph.co.uk/	107	260
loading_tq: default	4	135
loading_tq: about:	26	131
loading_tq: http://tpc.googlesyndication.com/safeframe/1-0-2/html/	7	52
default_loading_tq	981	52
loading_tq: http://s.effectivemeasure.net/html/	11	48
idle_tq	20	44
timer_tq: default	203	33
timer_tq: http://s.effectivemeasure.net/html/	1	16
default_tq	83	11