# V8 Performance Mode

## (a.k.a. RAIL Scheduling for V8)

skyostil@, hpayer@
April 28th, 2016
Tracking bugs: crbug.com/613518 (V8), crbug.com/613520 (Blink scheduler)

V8 is currently carefully optimized for the whole performance trade-off space to provide high throughput, low latency, and low memory consumption in the general case. However, optimizing for all dimensions with the same priority is not possible. Hence, we have special implicit performance modes in V8 where V8 tries to guess a given user scenario like page load and configures itself to favor that scenario slightly more.

Guessing the modes is complicated and depends usually (in the context of the GC) on complicated heuristics that may be tricked easily. Getting an explicit mode from the trusted embedder may be a cleaner signal for V8.

# RAIL Mode API

If Chrome could tell us what type of V8 performance is expected, we could potentially do a way better job in optimizing for the given user expectation. Chrome could tell us the desired performance mode through a PerformanceMode API.

```
enum RAILMode {
  DEFAULT,
  RESPONSE,
  ANIMATION,
  IDLE,
  LOAD
}

void SetRAILMode(RAILMode rail_mode);
```

We have the RAIL states and DEFAULT which would mean V8 ToT default configuration.

**Open question: Would it make sense to distinguish R and A?**
RESPONSE and ANIMATION both care about latency. RESPONSE could be seen as a less restrictive latency mode. For example, we could configure the garbage collector to perform more compaction than in A mode.

**Design decision MEMORY mode**
Use a separate API for the memory coordinator to make RAIL and memory decisions independent of each other.

# RAIL Mode in V8

**NONE**

- Optimize for the general case. V8 ToT configuration

**IDLE**

- This signal would mean that the tab is truly idle (not on a frame granularity). That would be a strong signal to register a memory reducing garbage collection callback.

**LOAD**

- Right now we try to delay the first full garbage collection as long as possible when V8 thinks that it is in page load mode. There are some complicated semi-broken heuristics in the heap growing strategy that also depend on context disposals to optimize for that scenario.
- Don't perform GC latency optimizations in that phase: turn off incremental marking, allow large heap growing factor, etc.
- Don't perform memory optimizations in that phase: turn off or limit compaction
- Use a special compiler configuration: Make tiering up unlikely and use the interpreter as much as possible to finish page load fast. Problem: what if we would need optimized code to start up fast?

**ANIMATE and RESPONSE**

- This is very similar to ToT mode right now since we try to optimize for jank in the common case.
- Avoid large heap growing factors.
- Avoid compaction.

**DEFAULT**

- Regular ToT configuration of V8. Optimizing for the general case. It may be the case that DEFAULT is similar to ANIMATE right now. We can still make this distinction internally.

# V8 Garbage Collector

1. **Incremental Marking Steps**
   Currently the incremental marking steps are optimized for latency, i.e. they are small to make sure to avoid long pause times. Hence, incremental marking may take long to finish, which is fine from a latency perspective. However, incremental marking does not come for free. It comes with a heavy write barrier and incremental marking steps have to performed every n allocations which degrades throughput. From a throughput perspective, it would be ideal to not perform incremental marking or perform it in just a few big steps.
   Conclusion: Perform small incremental marking steps when low latency is important, perform big incremental marking steps when high throughput is required.
2. **Heap Shrinking**
   We currently start out with a huge old generation limit (maximum old generation size/2) and shrink this limit after every young generation GC depending on the

survival rate. With that we try to avoid expensive full GCs during page load time. As soon as the first full GC happens we are in regular V8 mode. When a major context disposal happened, we turn on the shrinking strategy again.
Conclusion: Knowing that we are performing page load would be an explicit signal and we do not have to approximate that mode with a heap shrinking strategy.

3. **Idle Signal**
   V8 think that it is idle if the allocation rate is low and JS invocation rate is low. However, that is not necessarily a good signal, e.g. looking at inbox or gmail kept open in the foreground tab will have a high JS invocation rate for a long time and it will not transition into idle state for a long time.
   Conclusion: Having a better external signal for idleness would allow us to shrink the heap faster.
4. **Compaction**
   Currently we perform a little compaction on regular GCs to optimize for latency and heavy compaction on memory reducing GCs to optimize for memory consumption.
   Conclusion: Depending on how critical latency is (in A, R, L) we could scale the compaction rate accordingly and use I as an indicator that we should compact heavy.

## Others

The compiler pipeline also may have some benefits. But currently there seem to be many open questions and issues.
I am not sure about the runtime system. What could we fundamentally different when we are in L or A?

# Chrome using the V8 RAIL Mode API

The Blink scheduler should know about the RAIL state we are currently in. It would tell V8 the given RAIL state over the SetRAILMode API.

RAIL only cares about the foreground tab. The RAIL mode may be impacted by a concurrently running memory coordinator which may set us into a memory mode which would tank RAIL performance. The memory coordinator has higher priority than the RAIL signal.

### RAIL use case detection

- **Response (LATENCY)**
  - Already detected by the Blink scheduler.
- **Animation (LATENCY)**
  - Essentially active rendering while there's no user input (!BeginFrameNotExpectedSoon && !recent_user_input)
- **Idle (IDLE)**
  - Already detected by the Blink scheduler.

    - Should restrict this to long idle periods only.
    - Problematic example case: http://imgur.com/gallery/lWuLjD5. This site is continually animating but looks visually static. Is there a way to classify this as idle without hurting other use cases such as animation or video playback?
- **Load (THROUGHPUT)**
    - Strawman idea: until first contentful paint we are definitely loading. After that, if the main thread is not responsive and we haven't passed a timeout since the last navigation, we're still loading.
    - Should check what Speed Infra / loading-dev are doing here
    - Problematic example case: Animation while loading. For example Gmail shows a progress bar while loading the app. We do not want to optimize for jank while the progress bar is present.