Show navigation

Getting garbage collection for free

Published 07 August 2015 · Tagged with internals memory

JavaScript performance continues to be one of the key aspects of Chrome’s values, especially when it comes to enabling a smooth experience. Starting in Chrome 41, V8 takes advantage of a new technique to increase the responsiveness of web applications by hiding expensive memory management operations inside of small, otherwise unused chunks of idle time. As a result, web developers should expect smoother scrolling and buttery animations with much reduced jank due to garbage collection.

Many modern language engines such as Chrome’s V8 JavaScript engine dynamically manage memory for running applications so that developers don’t need to worry about it themselves. The engine periodically passes over the memory allocated to the application, determines which data is no longer needed, and clears it out to free up room. This process is known as garbage collection.

In Chrome, we strive to deliver a smooth, 60 frames per second (FPS) visual experience. Although V8 already attempts to perform garbage collection in small chunks, larger garbage collection operations can and do occur at unpredictable times — sometimes in the middle of an animation — pausing execution and preventing Chrome from hitting that 60 FPS goal.

Chrome 41 included a task scheduler for the Blink rendering engine which enables prioritization of latency-sensitive tasks to ensure Chrome remains responsive and snappy. As well as being able to prioritize work, this task scheduler has centralized knowledge of how busy the system is, what tasks need to be performed and how urgent each of these tasks are. As such, it can estimate when Chrome is likely to be idle and roughly how long it expects to remain idle.

An example of this occurs when Chrome is showing an animation on a web page. The animation will update the screen at 60 FPS, giving Chrome around 16.6 ms of time to perform the update. As such, Chrome will start work on the current frame as soon as the previous frame has been displayed, performing input, animation and frame rendering tasks for this new frame. If Chrome completes all this work in less than 16.6 ms, then it has nothing else to do for the remaining time until it needs to start rendering the next frame. Chrome’s scheduler enables V8 to take advantage of this *idle time period* by scheduling special *idle tasks* when Chrome would otherwise be idle.

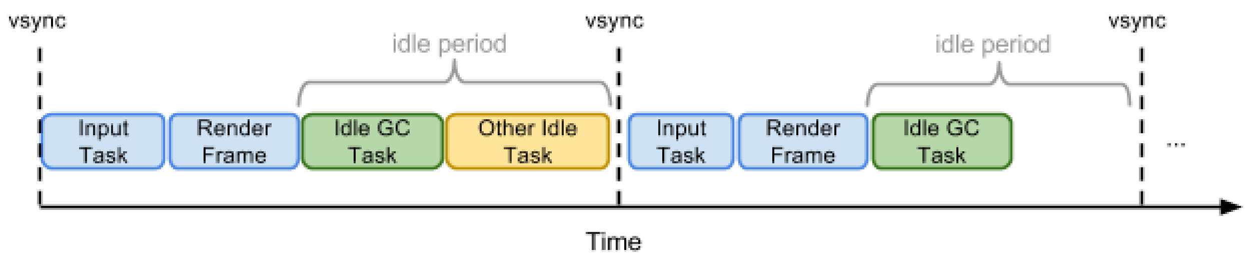


Figure 1: Frame rendering with idle tasks

Idle tasks are special low-priority tasks which are run when the scheduler determines it is in an idle period. Idle tasks are given a deadline which is the scheduler's estimate of how long it expects to remain idle. In the animation example in Figure 1, this would be the time at which the next frame should start being drawn. In other situations (e.g., when no on-screen activity is happening) this could be the time when the next pending task is scheduled to be run, with an upper bound of 50 ms to ensure that Chrome remains responsive to unexpected user input. The deadline is used by the idle task to estimate how much work it can do without causing jank or delays in input response.

Garbage collection done in the idle tasks are hidden from critical, latency-sensitive operations. This means that these garbage collection tasks are done for "free". In order to understand how V8 does this, it is worth reviewing V8's current garbage collection strategy.

Deep dive into V8's garbage collection engine

V8 uses a generational garbage collector with the Javascript heap split into a small young generation for newly allocated objects and a large old generation for long living objects. Since most objects die young, this generational strategy enables the garbage collector to perform regular, short garbage collections in the smaller young generation (known as scavenges), without having to trace objects in the old generation.

The young generation uses a semi-space allocation strategy, where new objects are initially allocated in the young generation's active semi-space. Once that semi-space becomes full, a scavenge operation will move live objects to the other semi-space. Objects which have been moved once already are promoted to the old generation and are considered to be long-living. Once the live objects have been moved, the new semi-space becomes active and any remaining dead objects in the old semi-space are discarded.

The duration of a young generation scavenge therefore depends on the size of live objects in the young generation. A scavenge will be fast (<1 ms) when most of the objects become unreachable in the young generation. However, if most objects survive a scavenge, the duration of the scavenge may be significantly longer.

A major collection of the whole heap is performed when the size of live objects in the old generation grows beyond a heuristically-derived limit. The old generation uses a mark-and-sweep collector with several optimizations to improve latency and memory consumption. Marking latency depends on the number of live objects that have to be marked, with marking of the whole heap potentially taking more than 100 ms for large web applications. In order to avoid pausing the main thread for such long periods, V8 has long had the ability to incrementally mark live objects in many small steps, with the aim to keep each marking steps below 5 ms in duration.

After marking, the free memory is made available again for the application by sweeping the whole old generation memory. This task is performed concurrently by dedicated sweeper threads. Finally, memory compaction is performed to reduce memory fragmentation in the old generation. This task may be very time-consuming and is only performed if memory fragmentation is an issue.

In summary, there are four main garbage collection tasks:

1. Young generation scavenges, which usually are fast
2. Marking steps performed by the incremental marker, which can be arbitrarily long depending on the step size
3. Full garbage collections, which may take a long time
4. Full garbage collections with aggressive memory compaction, which may take a long time, but clean up fragmented memory

In order to perform these operations in idle periods, V8 posts garbage collection idle tasks to the scheduler. When these idle tasks are run they are provided with a deadline by which they should complete. V8's garbage collection idle time handler evaluates which garbage collection tasks should be performed in order to reduce memory consumption, while respecting the deadline to avoid future jank in frame rendering or input latency.

The garbage collector will perform a young generation scavenge during an idle task if the application's measured allocation rate shows that the young generation may be full before the next expected idle period. Additionally, it calculates the average time taken by recent scavenge tasks in order to predict the duration of future scavenges and ensure that it doesn't violate idle task deadlines.

When the size of live objects in the old generation is close to the heap limit, incremental marking is started. Incremental marking steps can be linearly scaled by the number of bytes that should be marked. Based on the average measured marking speed, the garbage collection idle time handler tries to fit as much marking work as possible into a given idle task.

A full garbage collection is scheduled during an idle tasks if the old generation is almost full and if the deadline provided to the task is estimated to be long enough to complete the collection. The collection pause time is predicted based on the marking speed multiplied by the number of allocated objects. Full garbage collections with additional compaction are only performed if the webpage has been idle for a significant amount of time.

Performance evaluation

In order to evaluate the impact of running garbage collection during idle time, we used Chrome's Telemetry performance benchmarking framework to evaluate how smoothly popular websites scroll while they load. We benchmarked the top 25 sites on a Linux workstation as well as typical mobile sites on an Android Nexus 6 smartphone, both of which open popular webpages (including complex webapps such as Gmail, Google Docs and YouTube) and scroll their content for a few seconds. Chrome aims to keep scrolling at 60 FPS for a smooth user experience.

Figure 2 shows the percentage of garbage collection that was scheduled during idle time. The workstation's faster hardware results in more overall idle time compared to the Nexus 6, thereby enabling a greater percentage of garbage collection to be scheduled during this idle time (43% compared to 31% on the Nexus 6) resulting in about 7% improvement on our jank metric.

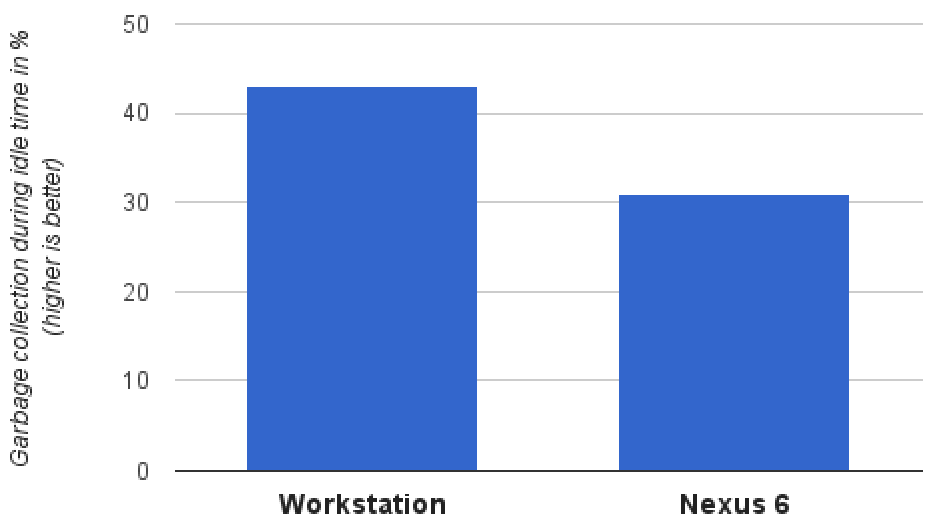


Figure 2: The percentage of garbage collection that occurs during idle time

As well as improving the smoothness of page rendering, these idle periods also provide an opportunity to perform more aggressive garbage collection when the page becomes fully idle. Recent improvements in Chrome 45 take advantage of this to drastically reduce the amount of memory consumed by idle foreground tabs. Figure 3 shows a sneak peek at how memory usage of Gmail's JavaScript heap can be reduced by about 45% when it becomes idle, compared to the same page in Chrome 43.

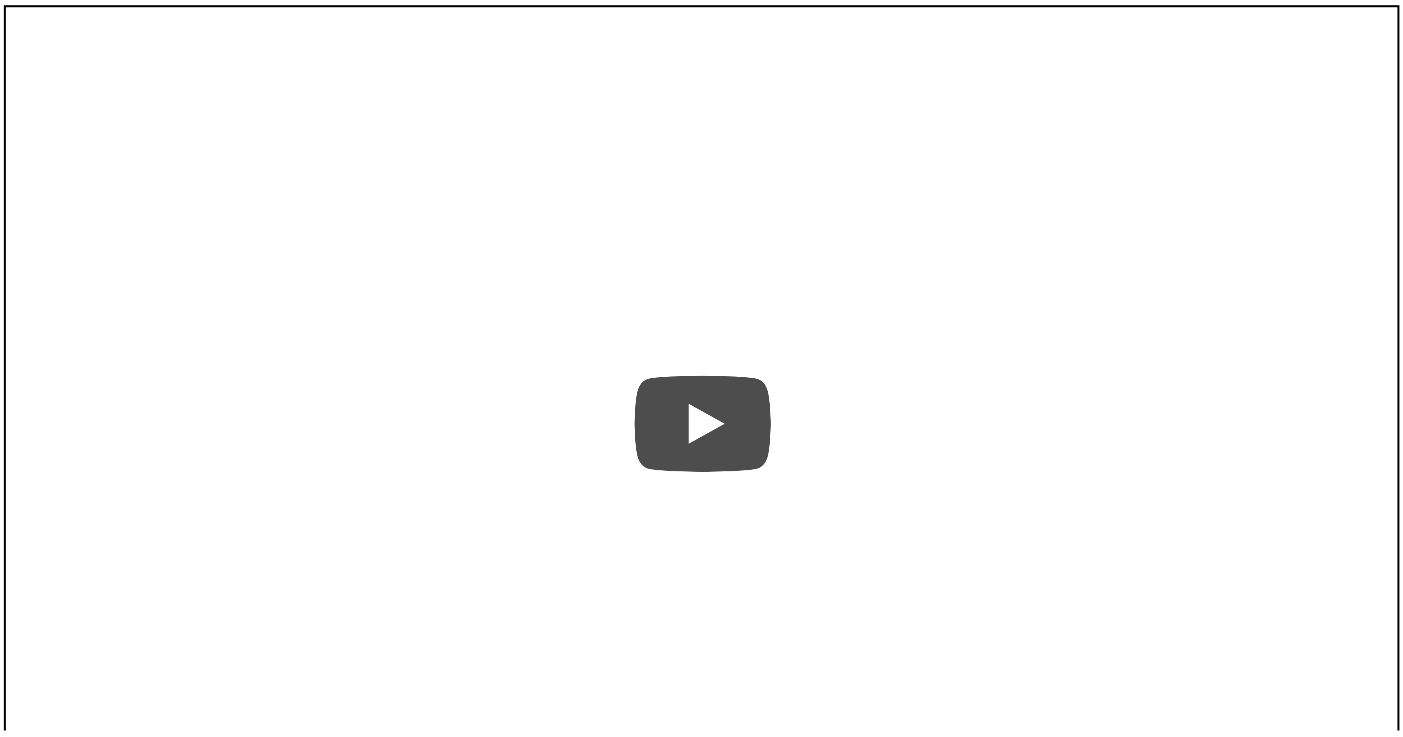
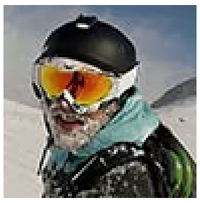


Figure 3: Memory usage for Gmail on latest version of Chrome 45 (left) vs. Chrome 43

These improvements demonstrate that it is possible to hide garbage collection pauses by being smarter about when expensive garbage collection operations are performed. Web developers no longer have to fear the garbage collection pause, even when targeting silky smooth 60 FPS animations. Stay tuned for more improvements as we push the bounds of garbage collection scheduling.



Posted by Hannes Payer and Ross McIlroy, Idle Garbage Collectors.

Branding · Terms · Privacy · Twitter · Edit this page on GitHub Dark Theme