Understanding how a web browser works
or tracing your way out of (performance) problems

Alexander Timin (altimin@chromium.org)
FOSDEM 2024
(self-link)
What is this talk actually about?

● Hi, I’m Alex
  ○ Software engineer in Google's Web-on-Android Performance team for the last 8+ years
● Problem solving in complex systems with illustrations
  ○ Chromium (chromium.org)
  ○ Perfetto (perfetto.dev)
● Not a how-to guide, but hopefully source of inspiration
  ○ For actually practically useful stuff, see this “intro to Chrome tracing” article
  ○ Would love to hear and chat more about similar problems
So, you want to improve performance

- Knowing what to improve is often most of the effort
- Performance problems can be anywhere in the code
- Modern web is complex (API surface / browser implementation / various sites)

⇒ ... then you’ll be spending considerable effort understanding new code on a recurring basis
How can do it?

- **Read the code**
  - Good luck!
- **fprintf**
  - console.log, (V)LOG, etc.
- **debugger**
  - gdb, lldebug, rr, Chrome DevTools

**These approaches don’t scale effectively to complex environments**
- Especially when multiple threads/processes are involved
- Indeterminism (flaky tests)
- Typically focusing on low-level details, not insights into high-level architecture
Enter tracing

Structured logging with visualisation:

- Turning this:

```java
void ResourceManagerImpl::RequestResourceFromJava(AndroidResourceType res_type,
   int res_id) {
    TRACE_EVENT2("ui", "ResourceManagerImpl::RequestResourceFromJava",
       "resource_type", res_type,
       "resource_id", res_id);
    Java_ResourceManager_resourceRequested(base::android::AttachCurrentThread(),
       java_obj_, res_type, res_id);
}
```

- Into this:
Enter tracing

Visualisation of what multiple threads / processes do in parallel
If you want to try it yourself

- ui.perfetto.dev + “Open Chrome example”
But how to make it useful?

- **Starting point: instrumenting the code you are working on**
  - Flexible and powerful, but not most convenient
  - Folks want to solve the problem, not add instrumentation
    - a single `fprintf` is more convenient
    - debuggers are guaranteed to have all information
- **Unrealistic to have all functions instrumented**
  - Too much data and overhead: slow to record and analyse
- **Finding opportunities for scaling the usefulness**
  - Few instrumentation points which give multiple insights
  - Usually infra / foundational pieces
Chromium task scheduler

- Event loop model:
  - Thread schedulers for “named” threads
  - Thread pool for “background” work
- Various places in the codebase post tasks:

```cpp
callback_task_runner->PostTask(
    FROM_HERE,
    base::BindOnce(std::move(callback), success, std::move(keys_entries)));
```

- Great chokepoint for tracing instrumentation
  - A couple of paths ~all work in Chromium is going through
  - Can get basic info which part of the codebase a given task is coming from
Chromium task scheduler: a single task

A single task (RunTask): FROM_HERE provides basic info about the task
Overview of all thread activity
RunTask trace events: cross-task dependencies are very powerful
Beyond task scheduler

- **FROM_HERE** might be useful
  - And might be not
- **Other “chokepoints”**
  - IPC system (mojo): cross-process communication
  - console.log & (D)(V)LOG
  - blink bindings (JS => C++ boundary): which JS calls are being made
  - JNI: Java => C++ boundary
  - GPU scheduler
  - Blink dispatched events
  - locks and other //base primitives

<table>
<thead>
<tr>
<th>Arguments</th>
</tr>
</thead>
<tbody>
<tr>
<td>task.posted_from</td>
</tr>
<tr>
<td>file_name</td>
</tr>
<tr>
<td>function_name</td>
</tr>
<tr>
<td>line_number</td>
</tr>
</tbody>
</table>
What’s next?

Status quo:

- **Good**: we have visibility into ~everything Chromium is doing
- **Bad**: it’s mostly low-level details and slow to work with
- **Ugly**: expertise-intensive

Aspiration:

- One can open a trace and learn something about how Chromium works
- (instead of requiring MS in tracing and PhD in Chromium architecture)
Inspiration

Architecture diagram from a Life of a Navigation talk from Chromium University
... and the status quo

The information is there, but the same insights will take a bit longer to get. (trace)
Existing examples

- EventLatency: breakdown of processing an input event and generating a frame
- Currently requires plumbing all of the data to a single location
  - Plumbing is very expensive in a large project (e.g. layering concerns, serialisation cost)
  - Difficult to scale
Enter Perfetto

- From chrome://tracing to perfetto.dev
- New UI, new more efficient format
- SQL data mode and query engine
  - Running custom queries from the UI
  - Running trace processor + SQLite in the browser via WASM
- Allows separation of “recording” and “analysis"
Perfetto powers

Query:

```
select thread_name, process_name, dur / 1e6 as dur_ms, printf('%s:%s', extract_arg(arg_set_id, 'task.posted_from.file_name'), extract_arg(arg_set_id, 'task.posted_from.file_name')) as posted_from from thread_slice where name = 'ThreadControllerImpl::RunTask' ORDER BY dur desc limit 100
```

Enter ":" into the search box to enter the SQL mode
Next steps

- Trying to build navigation instrumentation in Chromium as PoC
  - Focusing on the higher-level concepts
  - Links to the lower-level implementation details (e.g. specific functions being called)
  - Inline documentation in the UI and explaining the concepts

- Challenge: complexity and # of corner cases
  - ~50+ of various cases which affect the breakdown
  - Automatic testing is a prerequisite

Current status of the prototype:
Bonus: Chrome DevTools
and the importance of presenting the right information
Bonus: Chrome DevTools
and the importance of presenting the right information

It's just Chrome traces with post-processing in DevTools frontend

You can open the same trace in chrome://tracing / Perfetto, but it will be less useful