Perfetto

Platform-wide performance instrumentation and tracing for Android and Chrome

Tracing Summit 2018 - Edinburgh

primiano@google.com
What is Perfetto about?

1. Record traces
   - Tracing library & daemons

2. Analyze traces
   - Perfetto trace processor

3. Visualize traces
   - Perfetto UI

on-device

offline
What is Perfetto about?

1. An open source (AOSP / Apache2 license) project for recording, processing and visualizing traces.


3. Integration with ftrace, /proc/{stat,vmstat,pid/*} and soon perf_event_open.

4. A SQLite-based codebase for analyzing and processing traces.

5. A UI frontend.

* Some copies / allocations are involved, once every ~4KB.
Perfetto UI
Visualize all the things

Perfetto tracing service
Producer endpoint

Platform probes services
Linux ftrace
/proc/... interfaces

Consumer endpoint

Any app / process

Shared memory

Protobuf-based config

Protobuf-based trace

Perfetto trace processor
Ingest traces and expose as SQLite vtables

Heap profiler
Coming soon!

Perfetto trace processor
Ingest traces and expose as SQLite vtables
Where to find the code?

Android AOSP
source of truth
//external/perfetto

Chromium
//third_party/perfetto

Github
catapult-project/perfetto

Mirrors
What is Perfetto about?

- **Linux ftrace**
- **Linux /proc/ interfaces**
  - /proc/stat, /proc/pid/stat*
- **perf_event_open**
  - Coming soon
- **Perfetto traced_probes**
  - privileged access to kernel interfaces
- **Arbitrary userspace processes**
- **Perfetto traced**
  - userspace tracing daemon
- **Perfetto trace processor**
  - Ingest traces and expose as SQLite vtables
- **Perfetto UI**
  - Visualize all the things
Userspace tracing library
# Key concepts

## Producers
- The thing that writes protobufs into the trace buffers
- Untrusted. Potentially malicious. Everything can be a Perfetto producer.
- On startup advertises its capabilities to the tracing service.
- At some point the tracing service asks it to start collecting data

## Tracing service
- The thing that owns the log buffers (there is one* buffer for the all system / browser)
- Acts as registry and handles handshakes between producers and consumer(s)
- In chrome: a /services service
- In android: a system service (traced)

## Consumer(s)
- The thing that configures all the tracing session and decides who should trace and what.
- Is allowed to configure the tracing service and read back the trace data
- Trusted / privileged
- In chrome: the thing that exposes data to the UI
- In android: shell (for the UI) and Android Metrics services
Tracing Service

- Producer endpoint
- Trace Buffers
- Consumer endpoint
Tracing service

Trace Buffers

Protozero: zero-copy protobuf

Any user-space process

Perf profiling

Heap profiling

I/O tracing

Platform probes services

Data source: linux.ftrace

Data source: ...

Data source: chromium.evt

Data source: ...

Producer 1

IPC channel

Shared memory

Producer 2

IPC channel

Shared memory

Producer endpoint

Tracing service
Shmem buffer format

Per-process shared memory buffer

- page_state (atomic word)
- proto_content_length
- protobuf encoded payload containing trace data

Page
Trace config

Trace session (one per consumer)

Producer 1
"android.systemui"
Data source: kernel ftrace
Data source: I/O tracing

Producer 2
"org.chromium.browser_process"
Data source: Chrome tracing
Data source: Chrome heap profiler
What is a trace?

A trace can be large (10 GB)
Trace Processor
Trace processor

C++11 + SQLite codebase

Ingests traces of various formats (for now our .proto and Chrome's JSON, in future also ftrace text)

Builds an in-memory columnar database from trace contents.

Exposes the storage to SQLite through vtable hooks

Adds some trace-specific constructs on top of conventional SQLite ones.
Trace processor

- .proto trace
- legacy trace formats

**Trace Processor**

Handles:
- Event sorting
- Data massaging
- Clock syncing
- String interning

Processed trace in columnar storage

- **Sched slices**
  - Start
  - Duration
  - Thread ID
  - CPU

- **Userspace slices**
  - Start
  - Duration
  - Thread ID
  - Event name
  - Depth

- **Thread map**
  - Thread ID
  - Thd name

- **Process map**
  - Process ID
  - Proc name

SQLite Virtual Tables

- `$_`
- `trace_processor_shell`
- `ui.perfetto.dev`

Docs on [www.perfetto.dev](http://www.perfetto.dev)
See [/docs/trace-processor.md](http://docs.perfetto.dev)


```
$ out/mac_release/trace_processor_shell ~/Downloads/1gb-trace-truncated.proto
trace_processor_shell.cc Trace loaded: 1048.58 MB (184.9 MB/s)

> select proc_name, cpu, cpu_sec from (select process.name as proc_name, upid, cpu, cpu_sec from (select cpu, utid, sum(dur)/1e9 as cpu_sec from sched group by utid) left join thread using(utid) left join process using(upid)) group by upid, cpu order by cpu_sec desc limit 100

<table>
<thead>
<tr>
<th>proc_name</th>
<th>cpu</th>
<th>cpu_sec</th>
</tr>
</thead>
<tbody>
<tr>
<td>migration/2</td>
<td>2</td>
<td>2532.212882</td>
</tr>
<tr>
<td>migration/3</td>
<td>3</td>
<td>2529.064936</td>
</tr>
<tr>
<td>migration/1</td>
<td>1</td>
<td>2527.338100</td>
</tr>
<tr>
<td>migration/4</td>
<td>4</td>
<td>2526.877703</td>
</tr>
<tr>
<td>migration/5</td>
<td>5</td>
<td>2524.508852</td>
</tr>
<tr>
<td>migration/6</td>
<td>6</td>
<td>2523.372052</td>
</tr>
<tr>
<td>migration/7</td>
<td>7</td>
<td>2522.564051</td>
</tr>
<tr>
<td>/system/bin/surface</td>
<td>3</td>
<td>22.770180</td>
</tr>
<tr>
<td>rcu_preempt</td>
<td>7</td>
<td>16.257903</td>
</tr>
<tr>
<td>irq/760-synapti</td>
<td>4</td>
<td>14.566679</td>
</tr>
<tr>
<td>smem_native_rpm</td>
<td>3</td>
<td>11.273782</td>
</tr>
<tr>
<td>kswapd0</td>
<td>7</td>
<td>10.327598</td>
</tr>
<tr>
<td>ksoftirqd/0</td>
<td>0</td>
<td>10.231438</td>
</tr>
<tr>
<td>kworker/u16:2</td>
<td>7</td>
<td>9.276288</td>
</tr>
<tr>
<td>migration/0</td>
<td>0</td>
<td>8.302623</td>
</tr>
<tr>
<td>/vendor/bin/msm_irqb</td>
<td>3</td>
<td>8.256403</td>
</tr>
<tr>
<td>kworker/u16:4</td>
<td>7</td>
<td>7.876912</td>
</tr>
<tr>
<td>rcuop/0</td>
<td>7</td>
<td>6.730403</td>
</tr>
<tr>
<td>rcuos/0</td>
<td>7</td>
<td>6.469543</td>
</tr>
<tr>
<td>sugov:0</td>
<td>3</td>
<td>6.113958</td>
</tr>
<tr>
<td>/vendor/bin/bc</td>
<td>3</td>
<td>5.919216</td>
</tr>
</tbody>
</table>
```
CREATE VIRTUAL TABLE `bounds` USING window;
UPDATE `bounds` SET window_start=X, window_dur=Y where 1
CREATE VIRTUAL TABLE clipped USING span(`sched`, `bounds`)
New constructs

CREATE VIRTUAL TABLE `bounds` USING window;
UPDATE `bounds` SET quantum=Z where 1
CREATE VIRTUAL TABLE `quantized` USING span(`sched`, `bounds`)
New constructs

CREATE VIRTUAL TABLE quantized USING span(sched, counters, cpu)

Join key
UI
Perfetto UI

Re-written from scratch from the ashes of chrome://tracing
Web-based: TypeScript + WebAssembly running in a worker
All the processing / analysis engine is based on the Trace Processor
Supports ~5 GB traces (limited by browser renderer limit)

URL: [https://ui.perfetto.dev](https://ui.perfetto.dev)
Or just build it from sources and run locally.
<table>
<thead>
<tr>
<th>Cpu Track 1</th>
<th>Cpu Track 2</th>
<th>Cpu Track 3</th>
<th>Cpu Track 4</th>
<th>Cpu Track 5</th>
<th>Cpu Track 6</th>
<th>Cpu Track 7</th>
<th>Cpu Track 8</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image1.png" alt="Cpu Track 1" /></td>
<td><img src="image2.png" alt="Cpu Track 2" /></td>
<td><img src="image3.png" alt="Cpu Track 3" /></td>
<td><img src="image4.png" alt="Cpu Track 4" /></td>
<td><img src="image5.png" alt="Cpu Track 5" /></td>
<td><img src="image6.png" alt="Cpu Track 6" /></td>
<td><img src="image7.png" alt="Cpu Track 7" /></td>
<td><img src="image8.png" alt="Cpu Track 8" /></td>
</tr>
</tbody>
</table>
Thanks for your attention

For docs / links:
www.perfetto.dev

primiano@google.com