

Google Automotive Partner Bootcamp

# Performance Analysis / Tuning 101

Google Automotive Partner Bootcamp



### **CONFIDENTIALITY REMINDER**

### Everything shared in this presentation is under NDA

Google Automotive Partner Bootcamp

### Ethan Lee

Software Engineer



# Agenda



- •2 Getting Started with Perfetto
- Anatomy of a Trace
- **04** Perfetto Pitfalls



## •5 Trace AnalysisWalkthrough

### **o6** SQL Queries

## Making DebuggingEasier

### **08** Performance Tuning

# A Scientific Approach to Performance Analysis





### What is Performance Analysis?

- Performance issues require a systematic process to uncover their root cause.
- The right tools need to be identified to gather insights into critical parts of complex systems.
- There are a number of techniques which engineers can use to delve deeper into the execution of a system.



### What is Performance Analysis?

There are two techniques that are widely used for performance analysis: Tracing and Profiling



#### **Tracing**

- Tracing involves collecting highly detailed data about system execution.
- Traces contain enough detail to build a timeline of events.
- Traces give us insight into what a program does over time (e.g. which functions are being run) and context about execution (e.g. function call parameters).



#### **Profiling**

- The most common types are memory profiling and CPU profiling.
- Memory profiling surfaces information about heap memory allocation.
- CPU profiling gathers information about the call stack running on a CPU over time.

Profiling involves sampling some usage of a resource by a program.

### Why Choose Perfetto?

Profiling and tracing have different use cases:

#### Why use profiling over tracing?

- Traces, while detailed, are impractical for capturing high-frequency events like every function call due to the sheer volume of data involved.
- Profilers address this limitation through sampling, selectively recording data points to drastically reduce storage requirements.

#### Why use tracing over profiling?

- Profilers offer valuable insights into where resources are consumed within a program's call stack, but they lack the ability to explain the underlying reasons behind those resource allocations.
- For instance, a profiler might reveal that function foo() called malloc numerous times and allocated X bytes, but it cannot tell us why foo() was making those calls.
- Traces fill this gap by combining application and kernel events, providing in-depth context to understand the root cause of resource consumption.

#### Perfetto addresses this by supporting the collection, analysis and visualization of both tracing and profiling.

### How to Use Perfetto Effectively

How does Perfetto and our performance analysis flow fit into our goals?



This approach allows one to easily compare the delta of a potential regression. To achieve this, one should have an established baseline to compare against.

Perfetto will enable one to gather insights beyond just surface level observations. It is imperative that we can translate user-perceptible signals into measurable metrics that can be tested.



Using Perfetto in this approach means that multiple iterations will need to be collected. In order to establish reliable metrics, it is necessary to gather information on a large enough population to capture reproducible issues.

### **How to Use Perfetto Effectively**

How does Perfetto and our performance analysis flow fit into our goals?



### **Perfetto: A Feature Rich Tool**

# 

#### Ease of Use

Perfetto provides an end-to-end solution to capture Android system traces quickly to identify issues in critical user flows.



#### Flexibility

Via Perfetto trace configs, users are able to modify tracing behavior via buffers or data sources. For example, one can easily change data sources to capture various ftrace events or atrace events.



#### **Trace Analysis**

Perfetto provides a comprehensive trace viewer web UI that empowers one to inspect, visualize, and analyze the collected data.



#### **Data Mining**

One can leverage SQL-like syntax to query the trace data, making complex analysis easier.

### How does Perfetto Work?



#### https://perfetto.dev/docs

### How does Perfetto Work?

#### System Wide Tracing for Android and Linux

- Kernel tracing is enabled via Linux ftrace, which allows kernel events such as scheduling events and syscalls to be recorded.
- /proc pollers allow the sampling of process-wide cpu and memory counters over a time period.
- Heap profilers also enable capturing information for the Native and Java heap.



<u>https://perfetto.dev/docs</u>  $\rightarrow$ 



### How does Perfetto Work?

#### **Trace Analysis**

- The Trace Processor is a C++ library that takes in raw trace data and surfaces it through an SQL interface for straight-forward querying.
- Trace importers allow simple ingestion of multiple formats
- Trace-based metrics creates pre-formatted and extensible queries that provide trace summaries. (e.g. CPU usage at different frequency states).

<u>https://perfetto.dev/docs</u>  $\rightarrow$ 



### How does **Perfetto Work?**

#### **Trace Visualization**

- A trace visualizer is instrumental for analysis and is powered by WebAssembly.
- The Perfetto UI works fully offline after initial opening.  $\bullet$



<u>https://perfetto.dev/docs</u>  $\rightarrow$ 

**Google** Automotive Partner Bootcamp

# **Getting Started** with Perfetto

Google Automotive Partner Bootcamp



### **Quick Start: Collecting a Perfetto Trace**

#### After defining an appropriate trace configuration, one can run the trace collection.

- Download the recording script using the below command: 1.
  - \$ curl -0 https://raw.githubusercontent.com/google/perfetto/master/tools/record\_android\_trace
  - \$ chmod u+x record\_android\_trace
- Start tracing using: 2.
  - \$ ./record\_android\_trace -o <trace-name>.trace -c <previous trace file>
- Run the desired CUJ or experiment 3.
- End the trace using Ctrl+C for the command run in Step 2 4.
- The trace will be automatically be opened in the browser after the collection has completed 5.

### Quick Start: Viewing a Trace

If one wants to open an existing trace file, navigate to **ui.perfetto.dev** to open and access a trace:



Q Search or type '>' for commands or ':' for SQL mode

### Quick Start: Viewing a Trace

Once the trace is generated, one can also generate a permalink to the existing trace that can be shared:

🎧 Perfetto 🛛 😑					Q Search or type '>'	for commands
Navigation ^			1 2000400.00:00	4000000:00:00	1 6000400:00:00	" Lucoodo
🗁 Open trace file	UTC 1970-01-01	19755d06:52:0	0	19755d06:52:02	19755d06:52:04	19755d06:52:06
Open with legacy UI	≎ <i>≡</i>	0.00,0.00,0.000,		a form of the second of the second of the		
O) Record new trace	Cpu 0 (little)					
	Cpu 1 (little)					
Current Trace	Cpu 2 (little)					
hoottrace perfetto-trace (244 MB)	Cpu 3 (little)					
Show timeline	Cpu 4 (mid)					
and and an	Cpu 5 (mid)					
≪ Share	Cpu 6 (mid)					
🛃 Download	Cpu 7 (big)					
曼 Query (SQL)	Cpu 0 Frequency	2.5 GHz				
🖾 Viz	Cpu 1 Frequency	2.5 GHz				
Metrics	Cpu 2 Frequency	2.5 GHz				
<ol> <li>Info and stats</li> </ol>	Cpu 3 Frequency	2.5 GHz				
<b>•</b>	Cpu 4 Frequency	2.5 GHz				dia fi I
	Cpu 5 Frequency	2.5 GHz	11 P.M			dist: 1 1
Convert trace 🔷 🗠	Cpu 6 Frequency	2.5 GHz	1 Y 4,14			the first state
	Cpu 7 Frequency	2.5 GHz	11 17 UT1 0 1	Children with the set	A CONTRACTOR OF	ALL AND A
Switch to legacy UI	Android App Startups		android.ca	com.renault.car.launch	her	
🛃 Convert to .json	Android logs					·
▲ Convert to .systrace	GPU Memory 📈	200M 7				

### **Demo Video**

An example video of a trace being collected from beginning to end.

## **Special Case: Collect Boot Time Tracing**

In Android TM+, the trace can be collected as seen previously. However the following setting must be enabled before the device is restarted:

adb shell setprop persist.debug.perfetto.boottrace 1

In Android SC-, the following steps are required to setup the device:

- 1. Boot tracing in Android SC- requires selinux to be set to permissive.
- The following .rc file on the right must be created. 2.
- adb root && adb remount must be run to remount the device. 3.
- Use the following commands to push the .rc and config file to the device: 4.

adb push perfetto\_boot.rc /etc/init/perfetto\_boot.rc adb push perfetto\_trace\_config.textproto /data/misc/perfetto-traces/boottrace.pbtxt

class late\_start disabled user shell group nobody oneshot stdio\_to\_kmsg

EOF

```
cat >> perfetto_boot.rc << 'EOF'</pre>
service perfetto_boot /system/bin/perfetto --txt -c
/data/misc/perfetto-traces/boottrace.pbtxt -o
/data/misc/perfetto-traces/boottrace.perfetto-trace
```

```
seclabel u:object_r:perfetto_exec:s0
capabilities DAC READ SEARCH
```

```
on property:persist.perfetto.boottrace=1
   rm /data/misc/perfetto-traces/boottrace.perfetto-trace
   start perfetto_boot
```

### **Special Case: Collect Boot Time Tracing**

The following steps are required to collect the trace:

- 1. Reboot the device using adb reboot
- 2. Stop perfetto and pull the trace:

adb shell pkill perfetto

adb pull /data/misc/perfetto-traces/boottrace.perfetto-trace



# Trace Anatomy

Google Automotive Partner Bootcamp



## **Trace Config Setup**

### Selecting the right trace config will allow one to collect the necessary data from the system.

- Perfetto provides granular control over data collection. Unlike always-on logging systems (e.g., Linux's rsyslog, Android's logcat), its tracing data sources start in an idle state.
- The TraceConfig is a protobuf message that controls your Perfetto tracing session. It outlines:
- System-wide Settings:
  - Maximum trace duration.
  - Number and size of memory buffers.
  - Maximum output file size.
- Data Source Specifications:
  - For kernel tracing, which ftrace events to enable.
  - For the heap profiler, the target process name and sampling rate.
- Data Routing: Specifies which buffer each data source should write into

#### Note: a sample config can be found at perfetto.dev/docs/concepts/config



#### How the Tracing Service Uses the TraceConfig

- The tracing service (traced) is your config manager. When you start a tracing session, the service:
  - **Reads System Settings:** It determines its behavior based on the TraceConfig's outer section (duration, buffers, etc.).
  - Activates Data Sources: It finds Producers that match the data sources listed in the config. Then, it starts each Producer and provides the relevant DataSourceConfig settings.

Tra	ace config
	Duration: 10s
	Buffers: #0/: 4MB #1/: 16MB
	<pre>data source: "linux.ftrace Ftrace_config {    Ftrace_events: "sched_switch    Ftrace_events: "sched_wakeup } data source: "android.heap heapprofd_config {    sampling_interval_bytes: 1    process_cmdline: "adbd"    Continuous_dump_config {      dump_phase_ms: 10000      Dump_interval_ms: 10000    } }</pre>



#### **Defining Buffers:**

- This section defines the number, size and policy of in-memory buffers owned by the tracing service.
- Fill Policy:
  - A RING\_BUFFER (default) fill policy will wrap over when full and replace the oldest trace data in the buffer.
  - A DISCARD fill policy will stop accepting data once full.

#### **Dynamic Buffer Mapping:**

• The target\_buffer field can be specified to indicate different buffers for data sources.

# Defining several buffers buffers: { size\_kb: 4096 fill\_policy: RING\_BUFFER buffers { size\_kb: 4096 fill\_policy: RING\_BUFFER }

#### **Defining Data Sources (Logcat)**

This data source will enable Android logcat messages to be shown:

```
data_sources: {
   config {
      name: "android.log"
      android_log_config {
      }
   }
}
```

-19576d02:16:14 981 609 617			
≎ <i>≡</i>			
Cpu 0 (little)			
Cpu 1 (little)			
Cpu 2 (little)			
Cpu 3 (little)			
Cpu 4 (mid)			
Cpu 5 (mid)			
Cpu 6 (mid)			
Cpu 7 (big)			
Cpu 0 Frequency			
Cpu 1 Frequency			
Cpu 2 Frequency			
Cpu 3 Frequency			
Cpu 4 Frequency			
Cpu 5 Frequency			
Cou 6 Frequency			
Current Selection	Ar	ndroid	Log
Android Logs [1	115,	129]	/ 85
19576d16:46:33.6	D		1
19576d16:46:33.6	D		
19576d16:46:33.6	D		
19576d16:46:33.6	W		1
19576d16:46:33.6	W		
19576d16:46:33.6	W		
19576016:46:33.6	D		
19576416:46:33.6	V		
19576d16:46:33.6	D		
19576d16:46:33.6	V		
19576d16:46:33.6	I		and dots
19576d16:46:33 6	D		

		Q Search		
00:00:00	2000d00:00:00	1   1   1   4000d00:00:00	1   1 6000d00:00:00	1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
		19576d16:46:33	19576d16:4	6:33
			700 000 000	
C COM			ç	
C C	Z	ссс	com	С
	tt	S	C Sima	t
		e		P P
			Š	S R R
dt S	C d	C C C C W W W W	Ca Wa W	
C C C C C C C C	carw c c Watc W W	C t C C W s W s	C	C C W
		C CO	Z	Z
2.5 GHz			1	
2.5 GHz				
2.5 GHz			L.	
2.5 GHz		anisana ali ana seconde hinning a saha		e di secondo di secondario di secolo
S				
52				
CompatibilityChangeReporter	system_serve	۱۲	Compat change	e id reported: 1357729
CompatibilityChangeReporter	system_serve	ir	Compat change	e id reported: 1357549
Zygote	zygote64		Forked child	process 2770
WindowMagnificationMgr	system_serve	I <b>F</b> Semanta de la fonda de la comunicación	requestConnec	ction duplicated reque
ActivityManager	system_serve		Slow operation	on: 10/ms so far, now
CompatibilityChangeReporter	system serve		Compat change	id reported, 2610721
SystemServerTiming	system_serve	r	ManagedServi	ces.unbindOtherUserSer
SystemServerTimina	system_serve	۱۲	ManagedServio	ces.unbindOtherUserSer
SystemServerTiming	system_serve	r	ManagedServio	ces.unbindOtherUserSer
NotificationListeners	system_serve	۱r	disabling not	tification listener fo
oro	com.google.a	ndroid.carassistant:intera	ctor SslGuard com	oleted installation.
CompatibilityChangeReporter	system_serve	۱۲	Compat change	e id reported: 161145

#### **Defining Data Sources (CPU Frequency)**

Various CPU frequency stats can be collected with the following data sources:

- Enabling the power/cpu\_frequency ftrace event Ο
- Setting cpufreq\_period\_ms > 0 (Note: only works on 0 Android SC-V2 and above)

Cpu 0 Frequency	2.5 GHz
	2.5 GHz
Cpu 2 Frequency	2.5 GHZ
Cpu 3 Frequency	2.5 GHz
Cpu 4 Frequency	2.5 GHz
Cpu 5 Frequency	2.5 GHz
Cpu 6 Frequency	2.5 GHz
Cou 7 Frequency	2.5 GHz

data\_sources: {

}

}

}

```
config {
   name: "linux.sys_stats"
   target_buffer: 1
   sys_stats_config {
        cpufreq_period_ms: 500
   }
data_sources: {
   config {
        name: "linux.ftrace"
        ftrace_config {
            ftrace_events: "power/cpu_frequency"
```

ftrace\_events: "power/cpu\_idle"

ftrace\_events: "power/suspend\_resume"

#### **Defining Data Sources (Jankiness)**

Jankiness can be examined with the frame timeline data source.

}

64.2 s +	1.9 s		+17.3 ms	+37.3 ms	+57.3 ms		+77.3 ms	6.2	+97.3 ms		+117.3 ms	+137.3 ms
Expected Timeline	+	4793	4794	4797	4801	4804	4806	4809	4812		4818	
Expected findine	*									4815		4821
10/13 (10:057) (10:00 (	50.00		4793	4797	4801	4804	4806	480	9 4812		4818	
Actual Timeline	*		4794							4815		4821
										_		

```
data_sources: {
  config {
    name: "android.surfaceflinger.frametimeline"
   target_buffer: 2
```



#### **Defining Data Sources (linux.process\_stats)**

- The linux.process\_stats data source gathers per-process statistics from the /proc/<pid>/status and /proc/<pid>/oom\_score\_adj files on Linux systems
  - Process memory usage (RSS, VMSize, etc.)
  - Open file descriptors Ο
  - Out-of-memory (OOM) score (indicates how likely the 0 kernel is to terminate the process when memory is low)



<ul> <li>system_server 1352</li> </ul>	
mem.virt	10 6
mem.rss	0.5 G
mem.rss.anon	75 M
mem.rss.file	0.25 G
mem.rss.shmem	2.5 M
mem.swap	50 M
mem.locked	100 M
mem.rss.watermark	0.5 G
oom_score_adj	1K

#### Defining Data Sources (linux.sys\_stats)

- The linux.sys stats data source gathers a range of system-level statistics from Linux. The following stat counters can be collected:
  - Stat Counters (proc/stat):
    - STAT CPU TIMES
      - user: Time spent running in user mode
      - nice: Time spent running niced user processes
      - system: Time spent in system (kernel) mode
      - idle: Time the process was idle
  - Mem Info Counters (proc/meminfo): Ο
    - Provides information such as free memory, anonymous memory.
  - VM Stat Counters (proc/vmstat):
    - Provides information on virtual memory such as page faults, pages in and out, etc.
  - Note: cpufreq\_period\_ms is only available above SC-V2. 0
    - The following error will be encountered otherwise:
    - No field named "cpufreq\_period\_ms" in proto SysStatsConfig.

data\_sources: { config { name: "linux.sys\_stats" target buffer: 1 sys\_stats\_config { stat\_period\_ms: 500 stat counters: STAT CPU TIMES

> meminfo period ms: 1000 meminfo\_counters: MEMINFO\_ACTIVE\_ANON meminfo\_counters: MEMINFO\_ACTIVE\_FILE meminfo\_counters: MEMINFO\_INACTIVE\_ANON meminfo\_counters: MEMINFO\_INACTIVE\_FILE meminfo counters: MEMINFO KERNEL STACK meminfo counters: MEMINFO MLOCKED meminfo counters: MEMINFO SHMEM meminfo counters: MEMINFO SLAB meminfo counters: MEMINFO SLAB UNRECLAIMABLE meminfo\_counters: MEMINFO\_VMALLOC\_USED meminfo\_counters: MEMINFO\_MEM\_FREE meminfo\_counters: MEMINFO\_SWAP\_FREE

vmstat\_period\_ms: 1000 vmstat\_counters: VMSTAT\_PGFAULT vmstat counters: VMSTAT PGMAJFAULT vmstat counters: VMSTAT PGFREE vmstat counters: VMSTAT PGPGIN vmstat\_counters: VMSTAT\_PGPGOUT vmstat\_counters: VMSTAT\_PSWPIN vmstat\_counters: VMSTAT\_PSWPOUT vmstat\_counters: VMSTAT\_PGSCAN\_DIRECT vmstat\_counters: VMSTAT\_PGSTEAL\_DIRECT vmstat counters: VMSTAT PGSCAN KSWAPD vmstat counters: VMSTAT PGSTEAL KSWAPD vmstat counters: VMSTAT WORKINGSET REFAULT

# Below field not available on < Android SC-V2 releases. cpufreq\_period\_ms: 500

}

#### Defining Data Sources (linux.sys\_stats)

<ul> <li>Misc Global Tracks</li> </ul>		
ION allocations (heap: all) metric	~	0.75 G
mem.ion	~	0.75 G
MemAvailable	~	5 G
MemFree	~	5G
MemTotal	~	7.5 G
nr_active_anon	~	0.5 M
nr_active_file	~	50 K
nr_file_pages	~	0.5 M
nr_free_pages	~	2.5 M
nr_inactive_anon	~	5 K
nr_inactive_file	~	0.5 M
nr_mapped	~	0.5 M
nr_zspages	~	5
num_forks	~	5 K
pgpgin	~	2.5 M
pgpgout	~	25 K
pswpin	~	0
pswpout	~	0

#### **Defining Data Sources (ftrace)**

Capturing ftrace events allows developers insights into kernel code. They are useful for analyzing latency or performance issues outside of userspace.

- Memory Events
- Low Memory Killer Events
- Sched Events

```
data_sources: {
  config {
    name: "linux.ftrace"
    target buffer: 2
    ftrace_config {
      # Memory events
      ftrace_events: "power/suspend_resume"
      ftrace_events: "mm_event/mm_event_record"
      ftrace_events: "kmem/rss_stat"
      ftrace_events: "ion/ion_stat"
      ftrace_events: "dmabuf_heap/dma_heap_stat"
      ftrace_events: "kmem/ion_heap_grow"
      ftrace_events: "kmem/ion_heap_shrink"
      # LMKD events
      ftrace events: "lowmemorykiller/lowmemory kill"
      ftrace_events: "oom/oom_score_adj_update"
      ftrace_events: "oom/mark_victim"
      # sched events
      ftrace_events: "sched/sched_process_exit"
      ftrace_events: "sched/sched_process_free"
      ftrace_events: "sched/sched_switch"
      ftrace_events: "sched/sched_wakeup"
      ftrace_events: "sched/sched_wakeup_new"
      ftrace_events: "sched/sched_waking"
   }
```

#### **Defining Data Sources (ftrace)**

In order to capture CPU scheduling events, ftrace\_events: "sched/sched\_switch" needs to be added to the linux.ftrace data source.

With this enabled the following can be captured:

- Threads scheduled per CPU
- Why a thread got de-scheduled (pre-emption, blocked by a mutex)
- When a thread becomes runnable

Cpu 0 (little)	C T	com Thr	com.goog	gle.andro Thread	id.compani -2 [2107]	iondevi	•	С Т	. coi Th	<b>n se</b> re se		l		<b>S</b> S		SA	S	system Activity
Cpu 1 (little)	syste andro	sys and	tr		syst	tem_ser	ver [1593] ] [1653]			<b>CO</b> bi	sy	stem_ser android.b	ver [1593 g [1653]	3]				S)
Cpu 2 (little)	trac stack	ed_perf [ -unwinding	<b>2075]</b> [2076]	l con Lit.	n C C C B T B	syster Activ	n_server	[ <b>15</b> · [	idma bind	p <b>2d [21</b> er:2122				com.	google.	androic	I.compa Thread-2	aniono 2 [2107]
Cpu 3 (little)	com sod	com.g	oogle.andro a_process [31	oid.ca 1944]	sys Act	C C S		an bi	C b	C C V S	syste Act	m_serve	<b>r [159</b> Jer [1	idma bind	a <b>p2d [2.</b> . ier:212			
Cpu 4 (mid)	С В	S 9	<b>sy</b> Ac															
Cpu 5 (mid)	sy Ac	<b>c tr</b> B tr	t t															
Cpu 6 (mid)	com.a car.c	com	.andro s. r.carla b.	com.a	android.car carlauncher	<mark>S</mark> b	com.an car.ca	S C b c	<b>S</b> b		com.ar	1	<b>со</b> са	1	com s	C	C	car.ca



#### **Atrace Categories:**

Predefined groups of trace events that make it easier to enable tracing for specific areas of the system.

#### **Fine-grained Process Tracing:**

The atrace\_apps functionality in Perfetto enables selective tracing of specific applications on Android. It allows you to capture trace data only from the processes of interest.

<ul> <li>com.google.android.gms.per</li> <li>502</li> </ul>	rsistent 3								
.gms.persistent 3502	Uni R		Run. Ru.,		Runnable	Uninter. Runna. Ru.	Runnable (Preempted) R	u. Run. R R	Runnable
.gms.persistent 3502	Activity binde	ThreadMain r transaction	ResourcesM	anager#applyCon	figurationToResourc	binder /system/l	ramework/org.apache.http	lega /syst	/system
			bi bind	ler transaction	binder transac	Open GetB IsUs Stat Open	createClassloaderNamesp	0pen G 1 S 0 O	OpenDex GetBes IsUse Stat Open o diopen:
						The second			

```
data_sources: {
  config {
    name: "linux.ftrace"
    target buffer: 2
    ftrace config {
      # Memory events
      atrace_categories: "aidl"
      atrace categories: "am"
      atrace categories: "dalvik"
      atrace_categories: "binder_lock"
      atrace_categories: "binder_driver"
      atrace_categories: "disk"
      atrace categories: "freq"
      atrace categories: "idle"
      atrace_categories: "gfx"
      atrace categories: "hal"
      atrace categories: "pm"
      atrace categories: "power"
      atrace_categories: "rro"
      # atrace apps
      atrace_apps: "lmkd"
      atrace_apps: "system_server"
      atrace apps: "com.android.systemui"
      atrace_apps: "com.google.android.gms"
      atrace_apps: "com.google.android.gms.persistent"
      atrace apps: "android:ui"
      atrace_apps: "com.google.android.apps.maps"
```

#### Writing to a Trace Output File:

If not recording time limit is specified, one will have to manually terminate the tracing session.

If duration\_ms is specified then, the trace will terminate automatically.

If write\_into\_file is true, then Perfetto will periodically stream results into a trace file.

Flush\_period\_ms defines the default drain period. A shorter period means a smaller userspace buffer is required. However, this will increase the performance intrusiveness of tracing.

Max\_file\_size\_bytes is used to cap the size of a trace file.

Flush\_period\_ms is used to periodically issue a Flush() to all data sources, forcing them to commit their data into the tracing service.

# No recording time limit (press CTRL+C to stop recording). # Alternatively: uncomment the line below to set a time limit. #duration ms: 1800000 write\_into\_file: true file\_write\_period\_ms: 5000 max\_file\_size\_bytes: 10000000000 flush period ms: 5000
### **Anatomy of a Trace: Binder Transactions**

There are two types of binder transactions:

- 1. Unidirectional: Using the **oneway** keyword in the AIDL language, these transactions do not wait for a reply after sending a parcel.
- Bidirectional: The transmitting end is blocked until it receives a reply. 2.

Note: This is only available in UDC onwards.





### **Anatomy of a Trace: Bidirectional Transactions**

#### **Bidirectional Transaction:**

Identified by a corresponding binder transaction and binder reply pair.

pool-4-thread-1 17764

Binder:17769\_3 17788

Binder:17769\_3 17788



### **Anatomy of a Trace: Unidirectional Transactions**

#### **Unidirectional Transaction:**

Indicated by an arrow in a Perfetto trace.

DefaultDispatch 1779
Current Selection
Current Selection
Current Selection Slice Details Name
Current Selection Slice Details Name Category
Current Selection Slice Details Name Category Start time
Current Selection Slice Details Name Category Start time Duration

	Runnable (Preempted)	Running
BindingContext\$idle\$1	\$1:invokeSuspend	Bi
A ResultCaliback\$Stud	oProxy.onResult	

binder transaction async binder 3s 377ms 694us Os Ox11 his is a one-way call: async, no return; allow replies with file descript

## Common Pitfalls

Google Automotive Partner Bootcamp



### **Common Pitfalls**

#### **External Process Interference**

One of the reasons that a trace may be empty is that another process is using ftrace.Run either of the following to set the current\_tracer to nop:Run the following command to determine if the current\_tracer is nop:> adb shell echo "nop > /sys/kernel/tracing/current\_tracer"> adb shell cat /sys/kernel/tracing/current\_tracer> adb shell echo "0 > /sys/kernel/tracing/tracing\_on"

> adb shell cat /sys/kernel/debug/tracing/current\_tracer # older kernel may still use debugfs

	00:00:00		2000400:00:00	4000800:00:00	F 6000400:00-00	8000900:00-0	,	10000400-00-00
-19747d09-07-03 373 976 421		19747d09:07:12 000 000 000	19747d09:07:14 000 006 000	19747d09:07:16 000 000 000	19747d09:07:18 000 000 000	19747d09:07:20 000 000 000	19747d09:07:22 000 000 000	19747d09:07:24 000 000 000
x								
Android logs			en farre a restar a	vona div			1000 J 17 1	
∧ Chrome Scroll Jank								
Chrome Scrolls								
Chrome Scroll Janks								
Chrome Scroll Input Latencies								
✓ Misc Global Tracks								
✓ Kernel threads								
✓ surfaceflinger 945								
✓ Process 1494								
✓ Process 4026								
✓ Process 2130								
✓ Process 4160								

	12000400-00-00	14000400-00-0	0 1	6000400-00-00
	19747d09:07:26 000 006 000	19747d09:07:28 000 000 000	19747d09:07:30 000 000 000	19747d09:07:32 000 000 000
_				
			With Cold and	

### **Common Pitfalls**

#### **Insufficient Buffer Size**

Another common pitfall is insufficient data buffer size. Increasing buffer size may alleviate scenarios in which key CUJ data is dropped.

#### **Excessive event collection**

If too many events are being collected, there are some that can be dropped to avoid trampling the output trace file. Including sys\_enter and sys\_exit will lead to all system calls being logged. The below trace demonstrates this, where the tracks do not terminate. data\_sources: {
 config {
 name: "linux.ftrace"
 target\_buffer: 2
 ftrace\_config {
 # Do not include the below:
 ftrace\_events: "raw\_syscalls/sys\_enter"
 ftrace\_events: "raw\_syscalls/sys\_exit"

<ul> <li>system_server 1156</li> </ul>								
system_server 1156								
system_server 1156	S	S	S		S		S	sys_rt
RenderThread 2798								
RenderThread 2798					sys_epoll_pwait			sys. sys.
RenderThread 2839					Ex	it (Dead)		
RenderThread 2839				sys_nanosleep	S	ys_exit		
Signal Catcher 1174								
Signal Catcher 1174								sys
perfetto_hprof_ 1175								1
perfetto_hprof_ 1175								sys_rt sys_re
ADB-JDWP Connec 1176								1
ADB-JDWP Connec 1176								sys_rt sys_p

# Trace Analysis

Google Automotive Partner Bootcamp



### **Trace Analysis Key Steps**

#### **Summary:**

- 1. Narrow the search space: one can achieve this by determining the beginning and ending points via Android system logs or atrace logs.
- 2. Inspect CPU, memory tracks, etc: This will help identify symptoms of a regression so that the analysis window can be tightened.
- 3. Understand context: After capturing a smaller window, it is possible to understand what actions are being performed. (Ex. What is occurring during at this point in the user switch lifecycle?)
- 4. Identify Culprit Process: Given context, it is possible to visualize offending processes in the trace. Adding more logging via atrace will also allow one to trace points in a codepath.
- 5. Analyze thread-level interactions: Looking at markers such as thread state and binder transactions during the window will allow one to make informed hypotheses.



### **Trace Analysis Summary**

#### Sample flow illustrating:

- 1. CUJ profiling
- 2. Trace recording
- 3. CUJ instrumenting
- 4. Performance Fixes



Looking at a trace can be overwhelming. There are several key steps to help narrow down a problem area to root cause a performance issue. A trace analysis walkthrough will help guide investigations. Initially a trace was collected that captured a user switch from user 10 to user 11.

Narrow the search space: Use Android logs to identify key starting and stopping points. In this case flag UserController.startUser-11-fg-start-mode-1 and 1. onCompletedEventUser 11 act as the stop and start points.

	UTC 1970-02-18		19490d21:13:06 000 000 000	19490d21:13:08 000 000 000	19490d21:13:10 000 000 000	19490d21:13:12 000 000 000	19490d21:13:14 000 000 000
Cpu 0 (iii)       Image: Second	\$ <i>=</i>		User 11 Start				
Cpu 1 (ittis)       Image: Second Seco	Cpu 0 (little)						
Cpu 2 (nink)       Cpu 4 (nink) <td< td=""><td>Cpu 1 (little)</td><td></td><td></td><td></td><td></td><td></td><td></td></td<>	Cpu 1 (little)						
Cpu 3 (fitte)       Image: Species of	Cpu 2 (little)						
Cpu 4 (mid)       Image: Spin 4 (mid)	Cpu 3 (little)						
Cpu 5 (mid)       Image: Spin 2 (mid)	Cpu 4 (mid)						
Cpu 6 (mid)       26 Ma	Cpu 5 (mid)						
Cpu7 (bj)       25 BB	Cpu 6 (mid)						
Cpu 0 Frequency       2.5 GHz       4.5 G	Cpu 7 (big)						
Cpu 1 Frequency       25 GHz       25 GHz       24 G	Cpu 0 Frequency	2.5 GHz ar undur there a			ALIMATIL PANE A CAL C		
Cpu 2 Frequency       25 BH2       25 B	Cpu 1 Frequency	2.5 GHz			CHARLING TO THE T		
Cpu 3 Frequency       2.5 Hz       1000 000000000000000000000000000000000	Cpu 2 Frequency	2.5 GHz			CONCRUMENT FOR FOR	1 1 11	
Cpu 4 Frequency 2.56Hz   Cpu 5 Frequency 2.56Hz   Cpu 5 Frequency 2.56Hz   Cpu 7 Frequency 2.56Hz   Cpu 7 Frequency 2.56Hz   Cpu 7 Frequency 2.56Hz   Cpu 7 Frequency 2.56Hz   Current Selection Android Lusz <b>Android Lusz Timestamp</b> Level   Taimestanp SystemServer   SystemServer SystemServer   SystemServer SystemServer   SystemServer SystemServer	Cpu 3 Frequency	2.5 GHz			RUMARA ANA ANA ANA ANA ANA ANA ANA ANA ANA	1 1 11	
Cpu 5 Frequency       2.5 GHz       1000 000000000000000000000000000000000	Cpu 4 Frequency	2.5 GHz	TIMPER, C. N., L. J. LUM. PH. PROVIV.M.		Lui in T	1 1 1	
Cpu 6 Frequency 2.5 GHz   Cpu 7 Frequency 2.5 GHz   Cpu 7 Frequency 2.5 GHz   Other Current Selection Android Logs	Cpu 5 Frequency	2.5 GHz	LINER, C. L. LINE BURNEL DURING T		L HIL MILT IT	1 1 1	
Cpu 7 Frequency 2.5 GHz   Cpu Manoni 0.25 G   Current Selection Android Logs     Android Logs     Android Logs     Level Tag     Process name Message     194990d21:13:06.2   0   SystemServerTiming   system_server   UserController.startUser-11-fg-start-mode-1   19490d21:13:06.5   V   SystemServerTiming     system_server   UserController.startUser-11-fg-start-mode-1	Cpu 6 Frequency	2.5 GHz	I TINE MARKEN MER IN A TINA AND AND A TINA AND A T		LUIL MUTTE THE T	-1 I II II	
Current Selection       Android Logs         Current Selection       Android Logs         Android Logs [0, 0] / 2       Log Level       Verbose ~ Filter by tag       User-11-fg-start         Timestamp       Level       Tag       Process name       Message         19499d21:13:06.2       D       SystemServerTiming       system_server       UserController.startUser-11-fg-start-mode-1         19499d21:13:06.5       V       SystemServerTiming       system_server       UserController.startUser-11-fg-start-mode-1	Cpu 7 Frequency	2.5 GHz	<b>יידער איינעראר אראראר אראראר אראראר</b>				
Android Logs [v] / 2       Log Level Verbose ~ Filter by tag       User-11-fg-state         Timestamp       Level       Tag       Process name       Message         19499d21:13:06.2       V       SystemServerTiming       System_server       UserController.startUser-11-fg-start-mode-1       V         19499d21:13:06.2       V       SystemServerTiming       System_server       UserController.startUser-11-fg-start-mode-1       V	Current Selection Android Log	0 25 G	<b>N</b>				
TimestampLevelTagProcess nameMessage19490d21:13:06.2DSystemServerTimingsystem_serverUserController.startUser-11-fg-start-mode-119490d21:13:06.5VSystemServerTimingsystem_serverUserController.startUser-11-fg-start-mode-1 took to complete: 367ms	Android Logs [0, 0] / 2					Log Level Verbose 🗸 Filter b	y tag User-11-fg-sta
19490d21:13:06.2DSystemServerTimingsystem_serverUserController.startUser-11-fg-start-mode-119490d21:13:06.5VSystemServerTimingsystem_serverUserController.startUser-11-fg-start-mode-1took to complete: 367ms	Timestamp Level	Tag	Process name	Message			
19490d21:13:06.5 V SystemServerTiming system_server UserController.startUser-11-fg-start-mode-1 took to complete: 367ms	19490d21:13:06.2 D	SystemServerTiming	system_server	UserController.star	rtUser-11-fg-start-mode-1		
	19490d21:13:06.5 V	SystemServerTiming	system_server	UserController.star	rtUser-11-fg-start-mode-1 took	to complete: 367ms	



2. Inspect CPU and memory tracks: In this case, it is apparent that there are big and gold cores being underutilized during the user switch.

UTC 1970-02-18		19490d21:13:06 000 000 000	19490d21:13:08 000 000 000	19490d21:13:10 000 000 000	19490d21:13:12 000 000 000
\$ <i>=</i>		User 11 Start	Big and Go	d Cores Idle	
Cpu 0 (little)					
Cpu 1 (little)					
Cpu 2 (little)					
Cpu 3 (little)					
Cpu 4 (mid)					
Cpu 5 (mid)					
Cpu 6 (mid)					
Cpu 7 (big)					
Cpu 0 Frequency	2.5 GHz and undur more				I AL
Cpu 1 Frequency	2.5 GHz			NUMBER OF THE TOTAL OF T	1 1
Cpu 2 Frequency	2.5 GHz				1 10
Cpu 3 Frequency	2.5 GHz	A CONTRACTOR OF			1
Cpu 4 Frequency	215 GHz      [      [      [				
Cpu 5 Frequency	2.5 GHz	LINER, M. L. HANDLER, MARKET, JUNER, A.			1 1
Cpu 6 Frequency	2.5 GHz	I JADARAMAN MARANA A UTATALA AN MARANA DI DAMANA A			I II - 1
Cpu 7 Frequency	2.5 GHz	100000 1000L0070.0070.0070			1 1
ODU Mamany	0.25 G				
Current Selection Android Lo	gs				
Android Logs [0, 0] / 2				Log	g Level Verbose 🗸 Filter
Timestamp Level	Тад	Process name	Message		
19490d21:13:06.2 D	SystemServerTiming	system_server	UserController.sta	rtUser-11-fg-start-mode-1	
19490d21:13:06.5 V	SystemServerTiming	system_server	UserController.sta	rtUser-11-fg-start-mode-1 took to	complete: 367ms



After identifying the area of interest, it is necessary to zoom in on events occurring during this underutilization. This can be achieved by zeroing in on log events or processes that have significant activity during that period.

UTC 1970-02-18		19490d21:13:06 000 000 000	19490d21:13:08 000 000 000	19490d21:13:10 000 000 000	19490d21:13:12 000 000 000
\$ <i>=</i>		User 11 Start			
Cpu 0 (little)					
Cpu 1 (little)					
Cpu 2 (little)					
Cpu 3 (little)					
Cpu 4 (mid)					
Cpu 5 (mid)					
Cpu 6 (mid)					
Cpu 7 (big)					
Cpu 0 Frequency	2.5 GHz nr unruur uurne uu				r II
Cpu 1 Frequency	2.5 GHz				I IU
Cpu 2 Frequency	2.5 GHz				I IU
Cpu 3 Frequency	2.5 GHz				1 10
Cpu 4 Frequency	2.5 GHz				
Cpu 5 Frequency	2.5 GHz	LING MARINE AND A MANAGEMENT AND A MARINE AND A		UNIL UN UNIL MOUTON CONTRACTOR IN THE	
Cpu 6 Frequency	2.5 GHz	. JANG MANA JANG ALA ALA ALA ALA ALA ALA ALA ALA ALA AL		LATT WATER AND A REAL PARTY AND A REAL P	n n n
Cpu 7 Frequency	2.5 GHz				1 1
COULMamon/	0.25 G				
Current Selection Android Logs		Area of Inte	erest		
Android Logs [0, 0] / 2				Lo	g Level Verbose 🗸 Filter by
Timestamp Level Tag		Process name	Message		
<u>19490d21:13:06.2</u> D Syst	emServerTiming	system_server	UserController.	startUser-11-fg-start-mode-1	
19490d21:13:06.5 V Syst	emServerTiming	system_server	UserController.	startUser-11-fg-start-mode-1 took to	complete: 367ms



**3. Understand Context:** Here, only 4 out of 8 cores are being utilized, which undoubtedly contribute to a prolonged user switch. This idleness appears early on in the user switch when user 11 is being started. SystemServiceManager is responsible for starting system services during user initialization. SystemServiceManager will wait until all services are created. It is clear that com.android.role.RoleService is the last service to be initialized and also requires the most time.

JTC 1970-02-18	[	19490d21:13:05 000 000 000	19490d21:13:06 000 000 000		19490d21:13:07 000 000 000	19490d21: 000 000 00	:13:08 0	19490d21:13:09 000 000 000	19490d21:13:10 000 000 000	19490d21:13:11 000 000 000	1	9490d21:13:12 00 000 000	19490d21:13:13 000 000 000	19490d21 000 000 00	:13:14 0
\$ <i>=</i>			F	User 1	1 Start		RoleServi	ice Done							
Cpu 5 (mid)															
Cpu 6 (mid)															
Cpu 7 (big)															
Cpu 0 Frequency	1	2.5 GHz					-				U	■ 1.U		LEAST	1 1
Cpu 1 Frequency		2.5 GHz				.84				m	U				1 1
Cpu 2 Frequency		2.5 GHz - ຄະຫຼາກບານແຜນກາກ. ມາ		win			-			m	U	<b>1</b> U U		ามสามาก	1 1
Cpu 3 Frequency	1	2.5 GHz - พมานการขณะแกรม.แก								ini	U.	<b>T</b> UU			1 1
Cpu 4 Frequency	1	2:5 GHz   ]]][] ] [] [] [] [] [] [] [] [] [] [] [					TUMELLEN			m, in in	L	T UT		Tun_nr	1 1
Cpu 5 Frequency		2.5 GHz					tim_u_n			n i	U	1 11			1 1
Cpu 6 Frequency		2.5 GHz					um u n			11.		1 11			1 1
Cpu 7 Frequency	1	2.5 GHz		mum					1.J. J	п	1	1 11			1 1
GPU Memory	~	0.25 G	-												
Cpu 0 cpu-clock	~	25 G													
Cpu 1 cpu-clock	~	25 G													
Cpu 2 cpu-clock	~	25 G													
Cpu 3 cpu-clock	~	25 G												_	
Cpu 4 cpu-clock	~	25 G													
Current Selection Android Logs															<u></u> ↑ ~
Android Logs [0, 0] / 15								Log Lev	el Verbose 🗸	SystemServiceMar	nager ×	Filter by tag	Search	logs	×
Timestamp Level Ta	ag		Process n	ame			Message								
19490d21:13:06.5 I Sy	stemServi	iceManager	system_serv	ver			Calling d	onStartUser 11							
19490d21:13:07.0 W Sy	stemServi	iceManager	system_serv	ver			Service o	com.android.server.d	m.OverlayManager	rService took 506 m	ns in or	StartUser-11			
19490d21:13:07.2 W Sy	stemServi	iceManager	system_serv	ver			Service o	com.android.server.S	torageManagerSer	rvice\$Lifecycle too	ok 117 m	is in onStartUser	-11		
19490d21:13:07.8 W Sy	stemServi	LceManager	system_serv	ver			Service o	com.android.server.p	olicy.Permission	nPolicyService took	c 644 ms	in onStartUser-	11		
19490d21:13:08.3 W Sy	stemServi	iceManager	system_serv	ver			Service o	com.android.role.Rol	eService took 12	298 ms in onStartUs	ser-11				
19490d21:13:08.4 I Sy	stemServi	iceManager	system_serv	ver			Calling onSwitchUser 11 (from 10)								
19490d21:13:10.2 I Sy	stemServi	iceManager	system_serv	ver			Calling o	onUnlockingUser 11							
19490d21:13:10.4 I Sy	stemServi	LceManager	system_serv	ver			Calling o	onUnlockedUser 11							
9490d21:13:10.6 I Sy	stemServi	iceManager	system_serv	ver			Calling o	onStopUser 10							

#### com.android.role.RoleService onUserStarting method returns

4. Identify Culprit Process: Android logs allow us to identify that the com.google.android.permissioncontroller process is largely responsible for starting the RoleService. Zooming in further, it is apparent that the RoleControllerService thread handles majority of the initialization.

![](_page_49_Figure_2.jpeg)

5. Look for thread level interactions: Inspecting the com.google.android.permissioncontroller process and its threads may reveal further details about thread state. For example, a long period of uninterruptible sleep could indicate heavy I/O usage. In this case there is nothing that indicates anything out of the ordinary.

	com.google.android.permissioncontro ller 24864								
SS	ioncontroller 24864	Running Run	ning 📕	Running	Running 📕 Ru	Running	Runn. R. Runna.	R. R. Runn. R	R R.
SS	ioncontroller 24864								bindApplication
		2		/apex/com.	android.permission/priv-app	/GooglePermi	ssionController@MASTER/	GooglePermissionContro	oller.apk
				OpenDexFilesFromOat(/apex/com.a	ndroid.permission/priv-app/	GooglePermis	ssionController@MASTER/G	looglePermissionContro	olier.apk)
		1		Open dex file /apex/com.andro	id.permission/priv-app/Goo	glePermission	Controller@MASTER/Goog	ePermissionController.	apk
				Verify dex file /apex/com.andro	id.permission/priv-app/Goo	glePermission	Controller@MASTER/Goog	lePermissionController.	apk

![](_page_50_Picture_4.jpeg)

**5.** Analyzing the RoleControllerService thread reveals that there is an excess of costly binder transactions occurring. It is clear that these inter-process communication transactions are the cause of the slow down.

- 1970-02-18		19490d21:13:06 500 000 000	19490d21:13:07 000 000 000	19490d21:13:07 500 000 000		19490d21:13:08 000 000 000		19490d21:13:08 500 000 000	19490d21:13:09 000 000 000	19490d21:13 500 000 000		
≎		User 11 Start				[	Rol	eService Done				
pool-992-thread 24857	I		ssm.or	nStartUser-11_c	om.android.role.RoleS	ervice						
poor 552 thread 24007	т											
ssioncontroller 24864	Ŧ		Verify d									
RoleControllerS 24887	Ŧ											
✓ Kernel threads												
✓ system_server 1303												
<ul> <li>com.google.android.apps.automove.templates.host 25028</li> </ul>	i											
✓ traced_probes 1121			· · · · · · · · · · · · · · · · · · ·		-21 TR 1	8 ° 7 ° 8 7 °		1				
Current Selection Android Logs Flo	w Events									<u></u> ↑ ~		
Slice binder transaction									Contextu	al Options 👻		
Details					Following Flows							
Name binder transaction					~ Flow							
Category binder					Slice binder reply 7							
Start time 19490d21:13:07.9	15442744				Delay Os							
Absolute Time 2023-06-30T21:13		Thread binder:1303_14 3899 (system_server 1303)										
✓ Duration 354us 115ns				A								
Runnable (Preempted) 280us	Runnable (Preempted) 280us 729ns (79.28%)						Arguments					
> Kunning /3us 3	3600S (20.72%)				calling tid ~ 24887							
Process com google android permissioncontroller [24864]					code - 0x08 Java Layer Dependent							
SOLID slice[18934] -	Process com.google.android.permissioncontroller [24864]					292						
					destination name -	binder:1303_14						

## Advanced Topics: SQLQueries

![](_page_52_Picture_2.jpeg)

## **Data Mining Using SQL Queries**

Beyond visually inspecting system issues via the Perfetto UI, it is also possible to gain a deeper understanding through SQL queries.

One can access SQL queries via the below interface:

nerfetto		Q Search or type '>' for commands or ':' for SQL mode
Mandanation	Enter query and press Cmd/Ctrl + Enter	
Navigation	7 slice.id as slice_id, utid,	
🗁 Open trace file	9 FROM slice	
Dpen with legacy UI	J0 JOIN thread_track ON thread_track.id = slice.track_id J1 JOIN thread USING (utid);	
<ul> <li>Record new trace</li> </ul>	Query result (0 rows) - 0.8ms DROP VIEW IF EXISTS slice_with_utid; CREATE VIEW slice_with_utid AS SELECT ts, dur, slice.name as sl	ice_name, slice.id as slice_id, utid, thread.name as thread_name FROM slice JOIN thread_track ON thread_track.id = slice.track_id JOIN thread USING (utid);
	Query history (1 queries)	
Current Trace	DROP VIEW IF EXISTS slice_with_utid; CREATE VIEW slice_with_utid AS	
sample.trace (28 MB)	SELECT	
📰 Show timeline	dur,	
< Share	slice.name as slice_name, slice.id as slice_id, utid,	
🛃 Download	thread.name as thread_name	
S Ottery (SOL)		
🖄 Viz		
Metrics		
<ol> <li>Info and stats</li> </ol>		
Convert trace		
🔲 Switch to legacy UI		
🛃 Convert to .json		
🛃 Convert to .systrace	pe	
Example Traces		
🕒 Open Android exam	mple	
🖹 Open Chrome exam	mple	
Support		
⑦ Keyboard shortcuts	is	
Documentation		
F⊐ Flags		
Report a bug		
O) Record metatrace		

![](_page_53_Picture_5.jpeg)

### **Data Mining Using SQL Queries**

One common example is collecting the CPU Time for slices. The first step is to build a table that links slices with their thread state.

```
DROP VIEW IF EXISTS slice_with_utid;
CREATE VIEW slice_with_utid AS
SELECT
 ts,
  dur,
  slice.name as slice_name,
  slice.id as slice_id, utid,
 thread.name as thread_name
FROM slice
JOIN thread_track ON thread_track.id = slice.track_id
JOIN thread USING (utid);
DROP TABLE IF EXISTS slice_thread_state_breakdown;
CREATE VIRTUAL TABLE slice_thread_state_breakdown
USING SPAN LEFT JOIN(
  slice_with_utid PARTITIONED utid,
 thread_state PARTITIONED utid
);
```

### **Data Mining Using SQL Queries**

From the previous table, the CPU time for each slide in a Running state can be listed.

SELECT slice\_id, slice\_name, SUM(dur) AS cpu\_time
FROM slice\_thread\_state\_breakdown
WHERE state = 'Running'
GROUP BY slice\_id;

v and press Cmd/Ctrl + Enter		
slice_id, slice_name, SUM(dur) AS cpu_time		
state = 'Running'		
BY slice id;		
It (10000 rows) - 686 1ms SELECT slice id slice name SUM/dus) AS cou time EDOM slice thread state	breakdown WHEDE state - 'Dunning' CDOUD BY slice id:	Conviguory Convigent (tev) Close
It (10000 10WS) - 000. This SELECT side_id, side_italite, Som(dui) AS chu_time rRom side_timeau_state		Copy query Copy result (.154) Close
	slice_name	cpu_time
Contending for pthread mutex		28594
sys_epoll_pwait		11146
sys_ioctl		17916
sys_read		4479
sys_ioctl		39323
binder transaction async		0
sys_getuid		1042
	<pre>r and press Cmd/Ctrl + Enter slice_id, slice_name, SUM(dur) A5 cpu_time lice_thread_state_breakdown state = 'Running' BY slice_id; It (10000 rows) - 686.1ms SELECT slice_id, slice_name, SUM(dur) AS cpu_time FROM slice_thread_state Contending for pthread mutex sys_epoll_pwait sys_ioctl sys_read sys_ioctl binder transaction async sys_getuid</pre>	and press Cmd/Ctrl + Enter slice_iname, SUM(dur) AS cpu_time tice_thread_state_breakdown state = 'Running' BY slice_id;  t (10000 rows) - 686.1ms SELECT slice_id, slice_name, SUM(dur) AS cpu_time FROM slice_thread_state_breakdown WHERE state = 'Running' GROUP BY slice_id;  t (10000 rows) - 686.1ms SELECT slice_id, slice_name, SUM(dur) AS cpu_time FROM slice_thread_state_breakdown WHERE state = 'Running' GROUP BY slice_id;  t (10000 rows) - 686.1ms SELECT slice_id, slice_name, SUM(dur) AS cpu_time FROM slice_thread_state_breakdown WHERE state = 'Running' GROUP BY slice_id;  t (10000 rows) - 686.1ms SELECT slice_id, slice_name, SUM(dur) AS cpu_time FROM slice_thread_state_breakdown WHERE state = 'Running' GROUP BY slice_id;  sys_opoll_pwait sys_opoll_pwait sys_icetl sys_icetl sys_icetl binder transaction async sys_getuid

## Making Debugging Easier

![](_page_56_Picture_2.jpeg)

## Make App Debugging Easier

#### **Performance Analysis Flow:**

- 1. CUJ profiling
- 2. Trace recording
- 3. CUJ instrumenting
- 4. Performance Fixes

![](_page_57_Figure_6.jpeg)

## Make App Debugging Easier

A powerful feature that can help with debugging is adding atrace logs that will appear in Perfetto.

Java applications can add trace logs using android.os.Trace.

Native applications can add trace logs using ATrace\_beginSection() / ATrace\_setCounter() defined in <trace.h>

Trace.traceBegin(TRACE\_TAG, "Class#method");

• • •

Trace.traceEnd(TRACE\_TAG);

### Make App Debugging Easier

#### **Before:**

		Loom/_	VehicleStub#set
UserService 2047			Aid/VehicleStub/tget/DisetSync binder transaction
VehicleHal 2028			
binder:606_2 606	*		
binder:606_3 626	*		binder reply

#### After:

![](_page_59_Figure_4.jpeg)

		A.

# Performance Tuning

Google Automotive Partner Bootcamp

![](_page_60_Picture_2.jpeg)

### Performance Tuning After Boot

#### What is Performance Tuning?

With the aid of Perfetto, it is possible to identify performance issues and analyze their root cause. The next step is to implement solutions to solve these issues. One of the ways to achieve this is by iteratively tuning the performance of the system.

#### **Post Boot Tuning**

One of the opportunities for performance tuning is during post boot. After boot complete, there is heavy resource contention as multiple applications attempt to perform initialization. Classically, this is known as the Thundering Herd Problem, where lack of system resources leads to degraded performance. There is opportunity to both improve memory and CPU usage during the critical window after boot complete.

![](_page_61_Picture_6.jpeg)

# Memory Tuning

Google Automotive Partner Bootcamp

![](_page_62_Picture_2.jpeg)

### Kernel kswapd

<ul> <li>kswapd is a Kernel task to manage available free memory.</li> </ul>			
<ul> <li>Kernel uses 3 watermarks per memory zone to track pressure</li> </ul>	100		
<ul> <li>Min, Low, and High</li> </ul>	100		
<ul> <li>When free memory &lt;= Low and &gt; Min</li> </ul>			
<ul> <li>kswapd performs asynchronous/indirect memory reclaim.</li> </ul>	75		
<ul> <li>When free memory &lt;= Min</li> </ul>			
<ul> <li>kswapd performs synchronous/direct memory reclaim.</li> </ul>			
<ul> <li>System becomes unstable</li> </ul>	50		
<ul> <li>When free memory &gt;= High</li> </ul>			
<ul> <li>kswapd temporarily enters sleep state.</li> </ul>	25		
<ul> <li>Periodically checks the memory pressure.</li> </ul>			
Reclaim types			
Indirect memory reclaim	0		
<ul> <li>Increases kswapd CPU usage.</li> </ul>			
<ul> <li>May slow down other processes depending on the CPU &amp; memory pressure.</li> </ul>	Total memory Low		
Direct memory reclaim	High Min		
<ul> <li>All new allocations will be blocked until kswapd frees up memory up to min watermark.</li> </ul>			

#### Kswapd watermark levels

Sleep & Check	
High Watermark - Indirect claim	
 Low Watermark - Aggressive indirect reclaim	
Min Watermark - Direct reclaim	

Watermark levels

### kswapd tuning

Kernel knobs for tuning kswapd behavior:

- **/proc/sys/vm/swappiness** Defines the aggressiveness of swapping out memory pages of inactive processes.
  - Range: 0-100 Default: 60 Ο
  - High values can cause Kernel to swap out processes even when enough Ο memory is available.
  - Low values can cause Kernel to not swap out processes even when the Ο available memory is low.
  - **Recommendation:** Ο
    - Devices with high physical memory use lower swappiness values.
    - Devices with low physical memory use higher swappiness values.
- /proc/sys/vm/watermark\_scale\_factor Used to scale the buffer spaces between memory zone watermarks.
  - Range: 0 1000 Default: 10 Ο
    - 10 means buffer space is 0.1% of available memory.
    - 1000 means buffer space is 10% of available memory.
  - Low values can cause too much direct reclaim or kswapd not freeing up Ο enough memory in a single pass.
  - High values can cause kswapd to free up more memory than needed. Ο

- - Recommended value range: 1% 2% of total system memory. Ο
  - High values can scale up watermark buffer spaces leading to Ο
    - kswapd freeing too much memory than needed.
    - Frequent kswapd invocation causing CPU contention to spike.
  - Low values can cause kswapd to not free up enough memory leading to Ο System slowdown
- - Hangs/crashes

/proc/sys/vm/min\_free\_kbytes - Amount of free memory kept in reserve at all times. Defines min watermark across all memory zones.

Memory fragmentation

#### kswapd tuning example

#### **Before**

- /proc/sys/vm/watermark\_scale\_factor 1
- /proc/sys/vm/min\_free\_kbytes 144 MiB
- /proc/sys/vm/swappiness 60

#### After

- /proc/sys/vm/swappiness 60

Watermark Level Size Relative to Total Physical Memory

![](_page_65_Figure_11.jpeg)

/proc/sys/vm/watermark\_scale\_factor - 109

/proc/sys/vm/min\_free\_kbytes - 60 MiB

#### kswapd tuning example

![](_page_66_Figure_1.jpeg)

Kswapd cpu time millis

#### Working set refault file

![](_page_66_Figure_3.jpeg)

Google Automotive Partner Bootcamp

![](_page_66_Figure_5.jpeg)

![](_page_66_Figure_6.jpeg)

# Dex Optimization

Google Automotive Partner Bootcamp

![](_page_67_Picture_2.jpeg)

### **Dex Optimization**

#### What is Dex Optimization?

By default in Android, apps are executed in interpreted mode. Dex optimization allows for compilation of selected code paths to machine code, which accelerates code execution.

#### Why is this Important?

Background dex optimization already occurs regularly as users interact with their apps. However, after initial installation, boot time performance may be degraded if all apps run in interpreted mode. There is an opportunity to compile key apps ahead of time to greatly reduce boot times.

#### Which Apps should be Dex Pre-Optimized?

When apps run in interpreted mode, they will also undergo JIT compilation. Post-boot, a high level of JIT compilation will lead to a high degree of CPU contention, further exacerbating degraded startup times. Identifying processes with high JIT CPU time will indicate apps that can be dex pre-optimized, forgoing JIT entirely.

![](_page_68_Picture_8.jpeg)

## **Dex Optimization**

#### Heavy JIT Compilation in Perfetto

JIT compilation activity can be visualized via Perfetto.

Within a specific process, one can identify the JIT thread pool tracks. In this example, Car Assistant is displaying a large number of JIT compilation events.

<ul> <li>com.google.android.carassistant:sear ch 3267</li> </ul>				
Jit thread pool 3274	R., R Run., R., R., R R., R	Run., Runn., Runn., Runnab., S	R., A. R. Au, R., Run, Runna, R., Run,	Runnabie Ru. Ru. R.
Jit thread pool 3274	JIT com JIT J. JT. J. J. J. Compili Comp C Com C G C D.	JI JIT J J JIT J JIT J Co Com C C Com C Comp Cor	Fc J JIT compiling t J JIT mpi C Compiling basel C Com	JIT compi J J J J J J JI JI
HeapTaskDaemon 3275	1	1		11
HeapTaskDaemon 3275				

![](_page_69_Picture_6.jpeg)

## **Dex Optimization**

#### Perfetto Query for Top Processes with the Most JIT CPU Time

Using the query will allow one to obtain a table as shown below:

total_duration	instances	prefix_name	
5936808097	12608	JIT	com.google.android.carassistant
4889113085	9984	JIT	com.google.android.apps.map
4597115883	6785	JIT	com.android.vendin
2342370393	3474	JIT	com.google.android.apps.geo.automotive.adas
1440327723	2422	JIT	com.android.vending
1078105791	2169	JIT	com.google.android.tt

DROP VIEW IF EXISTS interesting\_slices\_d0; CREATE VIEW interesting slices dO AS select id as slice\_id, ts, dur, name, track\_id, track\_name, thread\_name, utid, tid, process\_name, upid, pid from \_slice\_with\_thread\_and\_process\_info where depth=0;

USING SPAN LEFT JOIN( thread\_state PARTITIONED utid );

slice\_thread\_state\_breakdown order by total\_duration desc;

INCLUDE PERFETTO MODULE slices.slices;

```
DROP TABLE IF EXISTS slice thread state breakdown;
CREATE VIRTUAL TABLE slice thread state breakdown
interesting_slices_d0 PARTITIONED utid,
```

```
SELECT sum(dur) total duration, count(*) instances, substr(name, 0,
IIF(instr(name, ' ') > 0, instr(name, ' '), IIF(instr(name, ',') > 0, instr(name, ','),
length(name)))) as prefix_name,
substr(process_name, 0, IIF(instr(process_name, ':') > 0, instr(process_name, ':'),
length(process_name))) as process_name_prefix FROM
WHERE state = 'Running' and prefix name = "JIT"
group by prefix_name, process_name_prefix
```

### **Dex Optimization** Configuration

#### How to Configure Dex Pre-Optimization

For more details refer to https://source.android.com/docs/core/runtime/configure#build\_options.

Add packages to the following makefile configuration:

#### PRODUCT\_DEXPREOPT\_SPEED\_APPS += \ MapsCarPrebuilt \
## **Dex Optimization Configuration**

#### How to Verify that an App is Dex Pre-Opted

Run the following ADB command:

\$ adb shell pm art dump com.google.android.apps.maps # Older releases may need to use this command instead: \$ adb shell dumpsys package dexopt | grep -i com.google.android.apps.maps -A 2

The following output indicates Google Maps is executed in interpreted mode:

```
[com.google.android.apps.maps]
path:
/system/product/priv-app/MapsCarPrebuilt/MapsCarPrebuilt.apk
x86_64: [status=verify] [reason=prebuilt]
```

The following output indicates that Google Maps was dex pre-opted:

```
[com.google.android.apps.maps]
path: /product/priv-app/MapsCarPrebuilt/MapsCarPrebuilt.apk
x86_64: [status=speed] [reason=prebuilt] [primary-abi]
```

### **Further Materials / Important Links**

#### Summary of useful information per section:

#### Trace Configuration:

https://perfetto.dev/docs/reference/trace-config-proto

#### How to Collect a Perfetto Trace:

https://perfetto.dev/docs/quickstart/android-tracing

#### Android Boot Tracing:

https://perfetto.dev/docs/case-studies/android-boot-tracing

#### **CPU Tracks:**

https://perfetto.dev/docs/data-sources/cpu-scheduling

#### Memory Tracks:

https://perfetto.dev/docs/data-sources/memory-counters

#### Atrace Logging:

https://perfetto.dev/docs/data-sources/atrace



Google confidential and proprietary | Do not distribute

# Thank you



**Google** Automotive Partner Bootcamp

Google Automotive Partner Bootcamp

