Panappticon: Event-Based Tracing to Optimize Mobile Application and Platform Performance

Lide Zhang†, David R. Bild†,
Robert P. Dick†, Z. Morley Mao†, and Peter Dinda‡

† Department of Electrical Engineering and Computer Science
University of Michigan

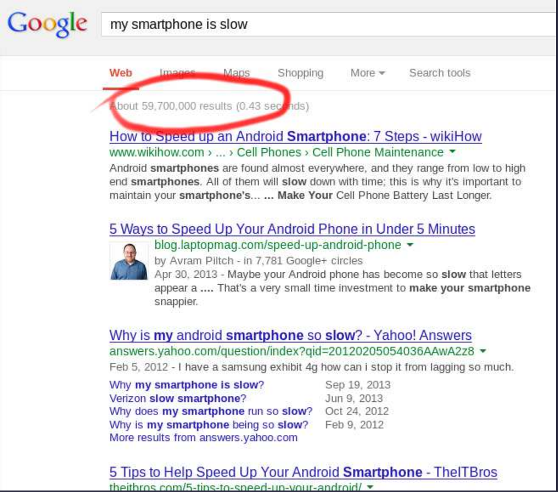
† Department of Electrical Engineering and Computer Science
Northwestern University

2 October 2013

Supported, in part, by the NSF under award CNS-1059372.

Outline

Goal: make smartphones faster



Why not make everything faster?

That could degrade

- cost,
- battery lifespan, or
- satisfaction with user interface.

Instead, make some things faster

What things?

Whenever smartphone users perceive that they are waiting for the machine, we have an opportunity to improve user-perceived performance.

How do we know when a smartphone user perceives that they are waiting?

User-perceived transaction definition

The best definition

A series of operations in the system started by user input and ended by the resulting output to the user.

A definition 1.5 graduate students can implement infrastructure for in a reasonable amount of time

A series of operations in the system started by a screen touch or button press and ended by the resulting display update.

How to monitor and analyze a user-perceived transaction?

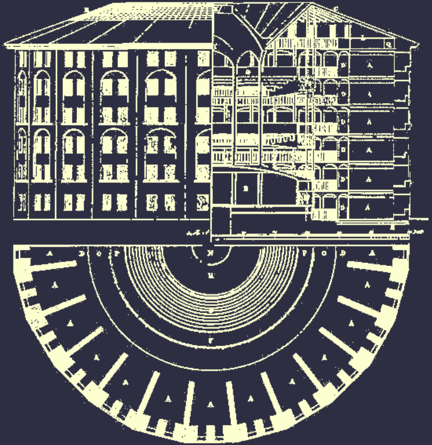
Questions

- When does it start and end?
- What are the causal relationships among events within the transaction?
- What takes time during the transaction?

Answering these questions is hard!

- The operating system and many user-level processes cooperate.
- Processes synchronize and communicate in many ways.
- Simultaneously running applications influence latencies of transactions via resource contention.
- Multiple ways to update the display.

Panopticon



A prison that has been radially arranged to allow a few guards to watch any prisoner at any time.

Panappticon

Smartphone infrastructure that monitors the detailed operations of multiple operating system and application processes to support identification and analysis of user-perceived transactions.

Who is Panappticon for?

Application designers: Optimize application performance.

Operating system designers: Optimize system policies.

Hardware designers: Choose the hardware changes that most improve user-perceived transaction latencies.

Related work

- [Barham'04]: Developer-provided event semantics used for trace analysis on servers.
- [Jovic'11]: Developers identify UI input methods. Unsuitable for multithreaded, asynchronous systems.
- [Ravindranath'12]: Instruments binaries to support tracing. Handles multiple application threads, but not other processes or kernel.

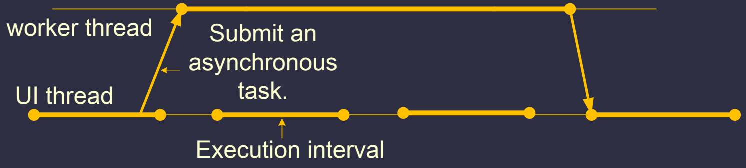
Panappticon handles multiple threads/processes, including kernel threads.

Outline

Algorithm overview

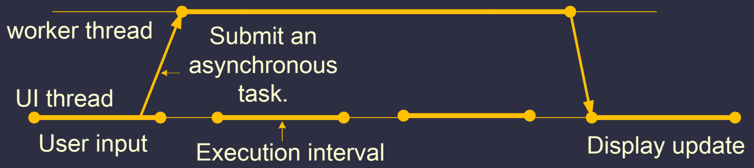
- Identify each execution interval.
- Identify causal relationships between intervals.
- Give intervals semantic labels.
- Do resource accounting along the critical path.

Algorithm overview



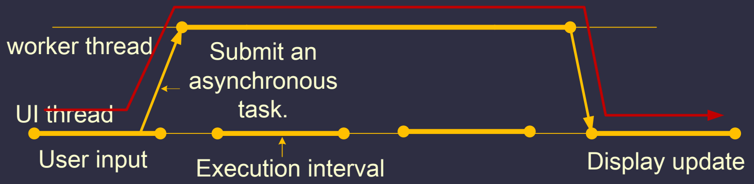
- Identify each execution interval.
- Identify causal relationships between intervals.
- Give intervals semantic labels.
- Do resource accounting along the critical path.

Algorithm overview



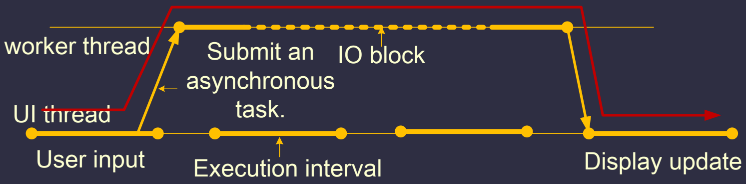
- Identify each execution interval.
- Identify causal relationships between intervals.
- Give intervals semantic labels.
- Do resource accounting along the critical path.

Algorithm overview



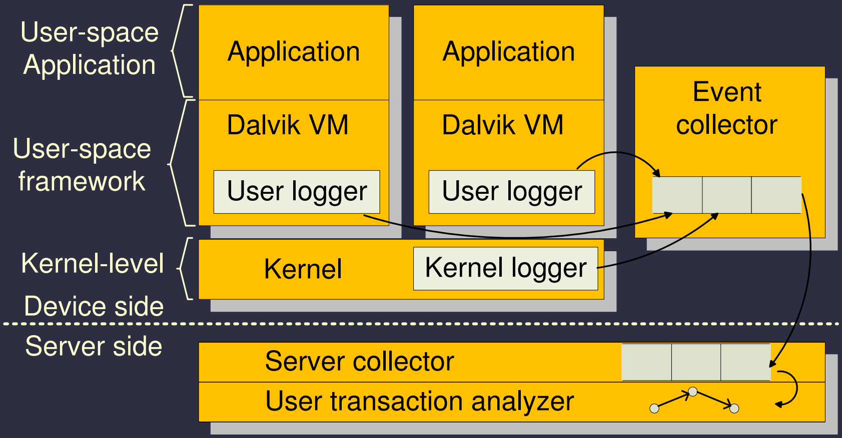
- Identify each execution interval.
- Identify causal relationships between intervals.
- Give intervals semantic labels.
- Do resource accounting along the critical path.

Algorithm overview



- Identify each execution interval.
- Identify causal relationships between intervals.
- Give intervals semantic labels.
- Do resource accounting along the critical path.

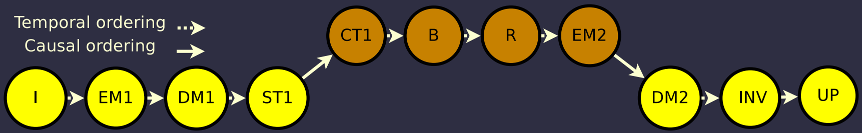
Panappticon architecture



Data captured

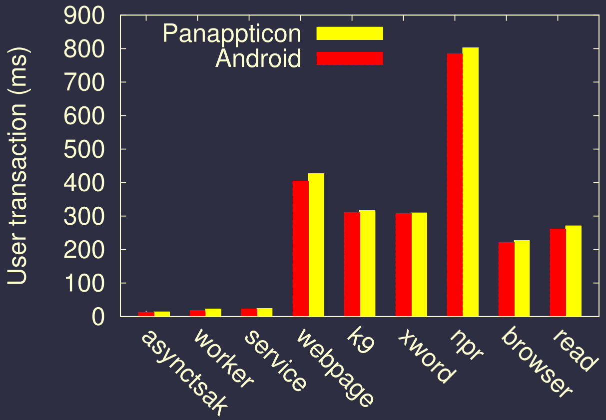
- Input events: screen touch and key press.
- Display update events.
- Causality between execution intervals: asynchronous task, enqueue/dequeue messages, IPC, forking a child thread (and locking primitives).
- Resource accounting events: context switches (and thread state), blocking on IO and network.
- Additional information to understand context: application name, foreground applications.

Relationship graph construction



- User input, enqueues message 1 (callback function for user input).
- Dequeues message 1 and submits asynchronous task 1.
- Consumes asynchronous task 1, blocks on IO, resumes, enqueues message 2.
- Dequeues message 2, triggers UI invalidate, UI display update.

Performance evaluation of Panappticon



Average performance overhead with Panappticon is 6.1%.

Outline

Experimental goals

Identify application performance bugs.

Understand the impact of system policies, e.g., DVFS.

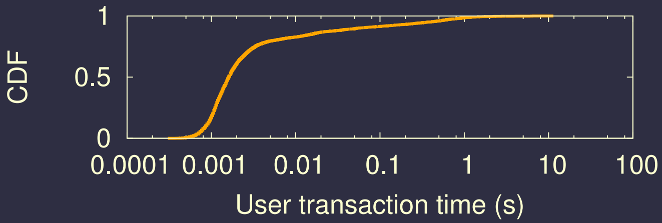
Understand the impact of hardware design decisions, e.g., multi-core versus single core.

Randomly switches between four configurations during deployment.

Study overview

Platform	Galaxy Nexus, Android 4.1.2
Users	14
Analyzed transactions	88,656
Duration	One month

User-perceived transaction durations



Transactions last 38.6 seconds at most. 2% of the transaction lasts longer than 1 second.

Both cores available. DVFS enabled.

Application commonly waits for CPU

Reddit news: a popular news application in Android market with millions of downloads.

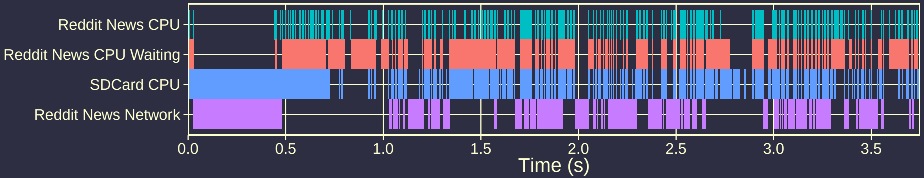
Total latency (s)	Network block (s)	IO block (s)	Waiting for CPU (s)
3.78	0.98	0	1.39
2.35	0.42	0.02	0.93
1.54	0.23	0	0.89
1.27	0.15	0	0.33

Application commonly waits for CPU

Reddit news: a popular news application in Android market with millions of downloads.

Total latency (s)	Network block (s)	IO block (s)	Waiting for CPU (s)
3.78	0.98	0	1.39
2.35	0.42	0.02	0.93
1.54	0.23	0	0.89
1.27	0.15	0	0.33

Cause of application stalls

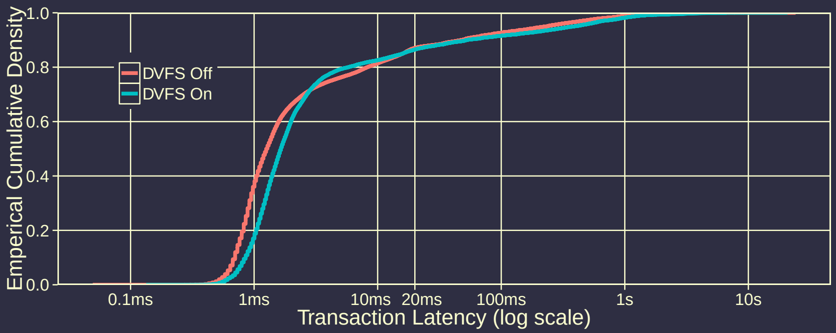


Observations

- System thread responsible for writing to SD card often preempts critical path thread.
- Network downloads temporally correlated with the SD card thread activity.

Possible application-level solution: defer saving until after user transaction.

Transaction latency as function of DVFS policy



- 517 ms additional delay at 98th percentile.
- DVFS governor significantly hurts the performance of long user transactions.

Cause of DVFS policy related latency increase

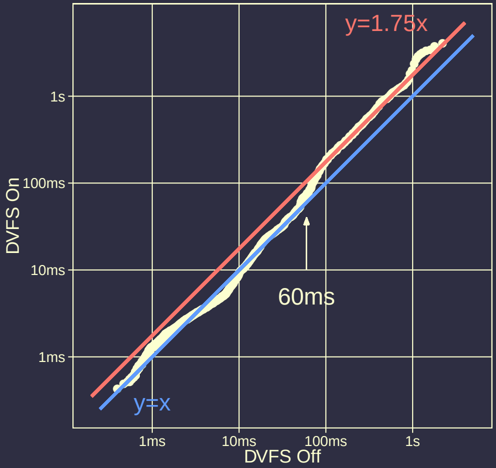
Interactive governor behavior

- Evaluation interval: 20 ms.
- Frequency increase when (1) the utilization in the window is above 85% or (2) on user input.
- Duration to stay at high frequency: 60 ms.

Why does this make long transactions slow?

- For shorter transactions, the frequency is boosted based user interaction.
- The frequency is allowed to drop after 60 ms.

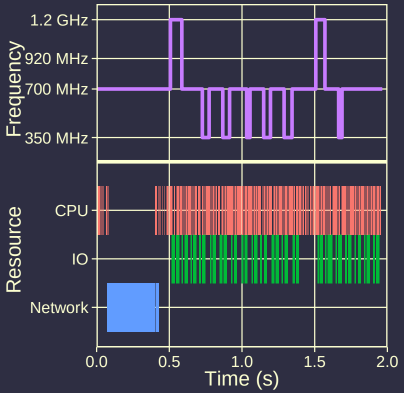
Dependence of latency on transaction duration



DVFS policy doesn't hurt performance for transactions < 60 ms.

$+75\%$ latency for transactions > 60 ms.

Impact of transaction time on DVFS policy and transaction time



Root cause

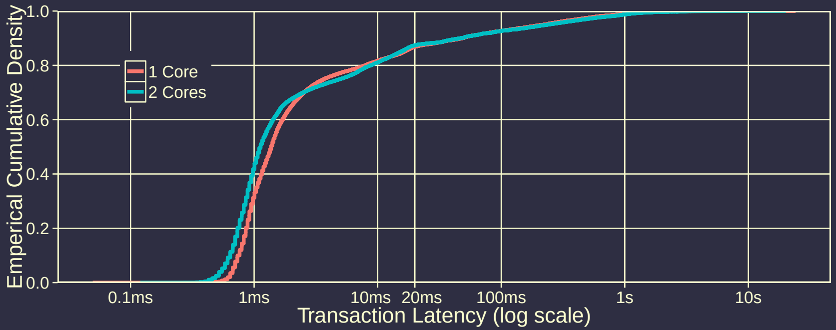
- Disk IO forces CPU frequency low.
- Transaction latency strongly dependent on CPU frequency despite low CPU utilization.

Methods for improving DVFS policy behavior

Extend duration to stay at high frequency (60 ms).

Have DVFS policy treat IO and network blocks as CPU activity.

Comparison of single- and dual-core transaction latencies



- Observation: Additional cores don't influence latencies of long transactions.
- Implication: These applications do not have parallelized CPU-bounded workloads for long transactions.

Suggestions

Parallelize CPU-intensive smartphone applications.

Improve single-core performance.

Panappticon summary

Panappticon traces relevant data to extract perceived user transactions.

We used it briefly to find and understand some interesting application/OS performance problems; you can do better.

Thanks and survey

Thank you for attending!

Try Panappticon: Guide application/OS/hardware improvements based on user-perceived transaction latencies.

http://ziyang.eecs.umich.edu/projects/panappticon.

Informal on-site survey

Who among you plans to use the tool or ideas described in this talk?