Jitter-Trace: a low-overhead OS noise tracing tool based on Linux Perf

Nelson Mimura, Alessandro Morari, Fabio Checconi

IBM T. J. Watson Research Center – Yorktown Heights, NY

June 27, 2017

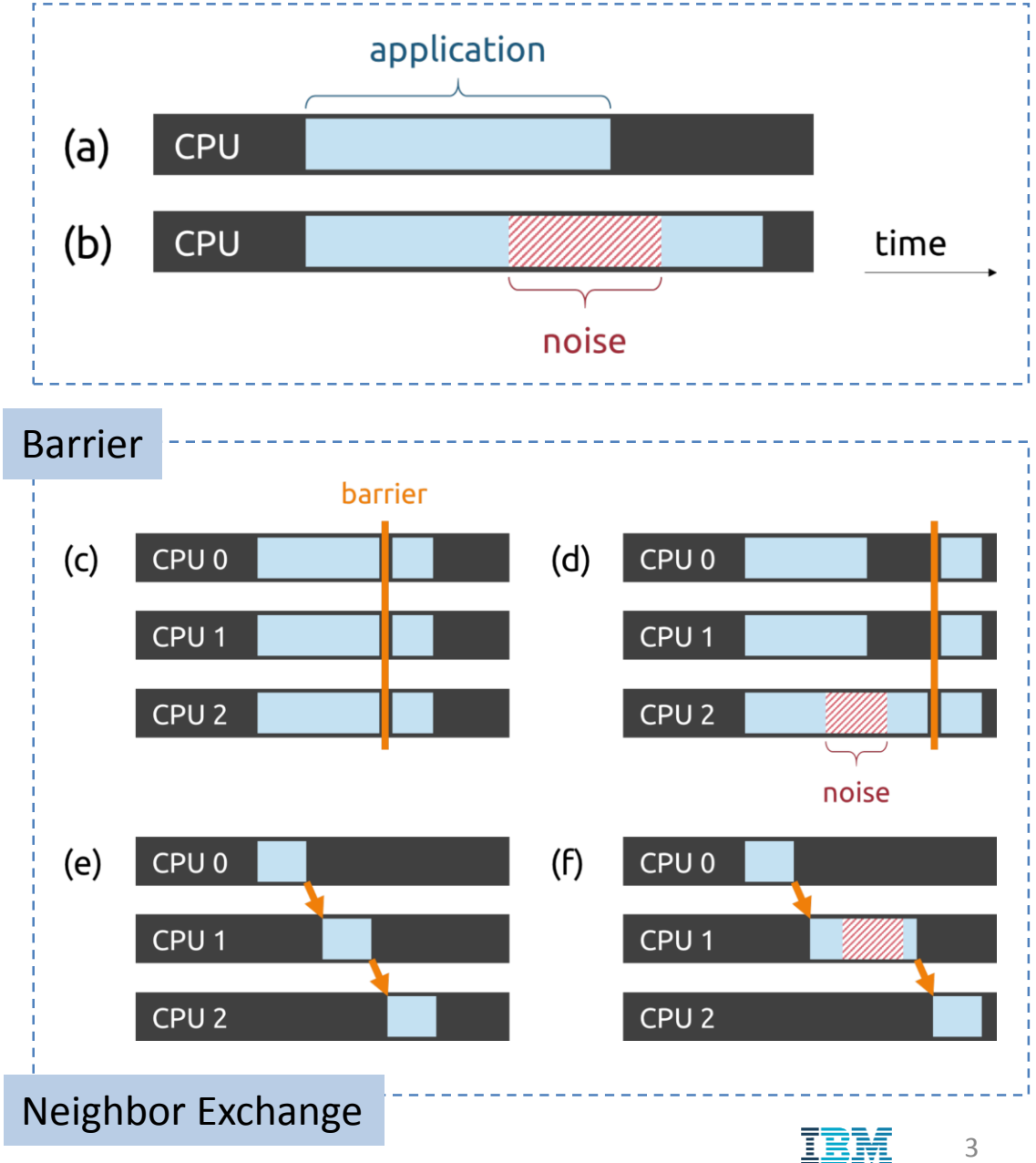
IBM

Outline

- Jitter
- Scheduling and Linux Tracing
- Tracepoints and Events
- Implementation and Experiments
- Overhead
- Conclusions

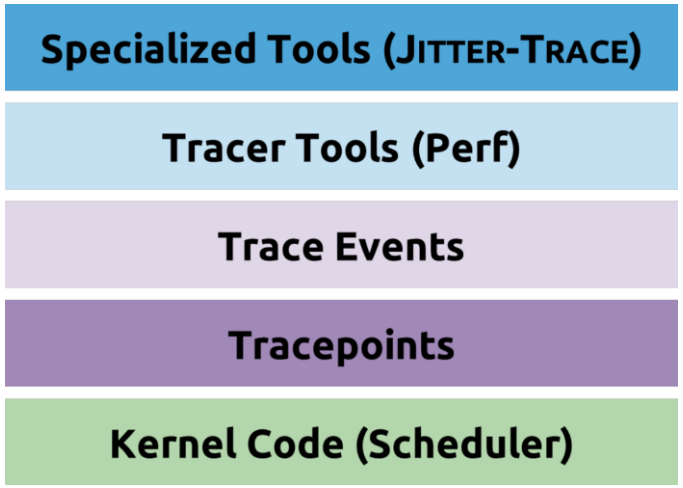
Jitter

- Activities can interfere with a running application by **stealing CPU time** from it
  - Noise can come from other sources (e.g., network, other applications)
  - This work focuses on noise caused by explicitly scheduling another activity on the (V)CPU of a running application
- System activity that stole processor time from application is called **offender**
- **Jitter** (runtime variability) increases amount of time to complete a task
  - Reduces system efficiency and throughput
  - Effects even more pronounced when application relies on **communication collectives** (typical of parallel distributed workloads)



Scheduler and Linux Tracing

- Linux kernel exposes several **Tracepoints** that can be used for debugging purposes
- **Trace events** provide a higher level API on top of tracepoints to simplify their utilization
- **Tracer tools** such as Perf provide a user-friendly interface to these events
  - User space event-driven tool
  - Widely available in many distributions
  - Based on Linux Performance Events Susbsystem
  - Low-overhead measurements (at least an order of magnitude faster than instrumenting profilers)
- **Jitter-Trace** is designed to use the Perf tracing infrastructure to perform a quantitative analysis of OS noise
  - No kernel patching/recompilation required



Tracepoints and Events

Tracepoint	Description
sched_wakeup	Wake up and resume a task
sched_switch	Context switch from one process to another
sched_migrate_task	Migrate task to another CPU
sched_process_free	Free a task (end process)
sched_stat_runtime	Account for process run time on a CPU

Track current status of each process in execution

→ Track how long each process has been running on a (V)CPU

	Process	PID	CPU	Timestamp	Function
1	swapper	0	[080]	451.405645:	sched_wakeup: psnap
2	swapper	0	[080]	451.405648:	sched_switch: swapper R ==> psnap
3	psnap	3	[080]	451.405651:	sched_stat_runtime: runtime=5802 [ns]
4	psnap	3	[080]	451.405651:	sched_switch: psnap D ==> swapper

- Scheduler manages CPU utilization by switching processes in and out
- Example of trace showing **switch in** (line 2, context is switched to a process) and **switch out** (line 4, context is switched to something else)

Example of Noise and Process State

	Process	PID	CPU	Timestamp	Function
1	psnap	2	[080]	368.440339:	sched_stat_runtime: runtime=4220 [ns]
2	psnap	2	[080]	368.440340:	sched_switch: psnap R ==> kworker
3	kworker	9	[080]	368.440355:	sched_stat_runtime: runtime=15946 [ns]
4	kworker	9	[080]	368.440356:	sched_switch: kworker S ==> psnap

states

	Process	PID	CPU	Timestamp	Function
1	psnap	3	[080]	451.572976:	sched_stat_runtime: runtime=24750 [ns]
2	psnap	3	[080]	451.573010:	sched_switch: psnap D ==> kworker
3	kworker	9	[080]	451.573039:	sched_stat_runtime: runtime=29938 [ns]
4	kworker	9	[080]	451.573040:	sched_switch: kworker S ==> swapper
5	swapper	0	[080]	451.573066:	sched_wakeup: psnap
6	swapper	0	[080]	451.573069:	sched_switch: swapper R ==> psnap
7	psnap	3	[080]	451.573072:	sched_stat_runtime: runtime=6172 [ns]
8	psnap	3	[080]	451.573073:	sched_switch: psnap D ==> swapper

- **psnap** is the application and **kworker** the OS process that generates noise
- **kworker** is switched in (line 2) and executes for around 16 us (line 3)

- Not every switch out leads to noise; at line 2, psnap is switched out in sleep state (flag **D**);
- Different from example above where psnap is switched out while in running state (flag **R**)

More Examples (1)

- Example of noise using microbenchmarks

	Process	PID	CPU	Timestamp	Function
1	konstant	2	[028]	407.930085:	sched_switch: konstant R ==> periodik
2	periodik	3	[028]	407.940083:	sched_stat_runtime: runtime=9998058 [ns]
3	periodik	3	[028]	407.950083:	sched_stat_runtime: runtime=9999818 [ns]
4	periodik	3	[028]	407.950085:	sched_switch: periodik R ==> konstant

More Examples (2)

- Composite noise (two sources in sequence: *periodik*, then *kworker*)

	Process	PID	CPU	Timestamp	Function
1	konstant	2	[028]	397.740341:	sched_stat_runtime: runtime=2204 [ns]
2	konstant	2	[028]	397.740341:	sched_switch: konstant R ==> periodik
3	periodik	3	[028]	397.750149:	sched_stat_runtime: runtime=9806684 [ns]
4	periodik	3	[028]	397.750151:	sched_wakeup: kworker
5	periodik	3	[028]	397.750340:	sched_stat_runtime: runtime=192446 [ns]
6	periodik	3	[028]	397.750346:	sched_switch: periodik R ==> kworker
7	kworker	2	[028]	397.750361:	sched_stat_runtime: runtime=16090 [ns]
8	kworker	2	[028]	397.750362:	sched_switch: kworker S ==> konstant

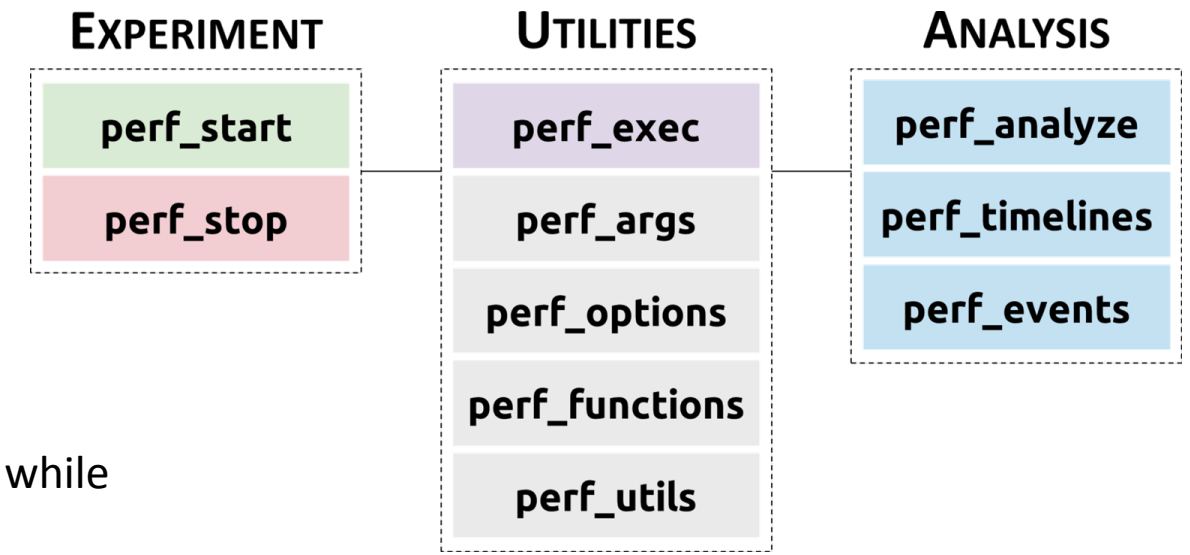
More Examples (3)

- Example of sleep – wake up (*periodik*)

	Process	PID	CPU	Timestamp	Function
1	periodik	3	[028]	398.760328:	sched_switch: periodik S ==> konstant
2	konstant	2	[028]	398.770312:	sched_stat_runtime: runtime=9984846 [ns]
3	(...)				
4	konstant	2	[028]	407.769103:	sched_wakeup: periodik
5	konstant	2	[028]	407.770086:	sched_stat_runtime: runtime=984164 [ns]
6	konstant	2	[028]	407.770088:	sched_stat_runtime: runtime=1502 [ns]
7	konstant	2	[028]	407.770088:	sched_switch: konstant R ==> periodik
8	periodik	3	[028]	407.780087:	sched_stat_runtime: runtime=9999280 [ns]

Implementation

- Python and Bash
- Three main modules to run experiments and perform analysis
  - perf_start and perf_stop
  - perf_analyze
- Analysis process
  1. Perf is used to collect scheduling information while application is run
  2. Perf writes its outputs in binary format
  3. **perf_timelines** uses Perf to convert binary format into textual timelines
  4. **perf_events** converts timelines into lists of events
  5. **perf_analyze** concentrates the required logic to identify sources of jitter and quantify the noise they generate



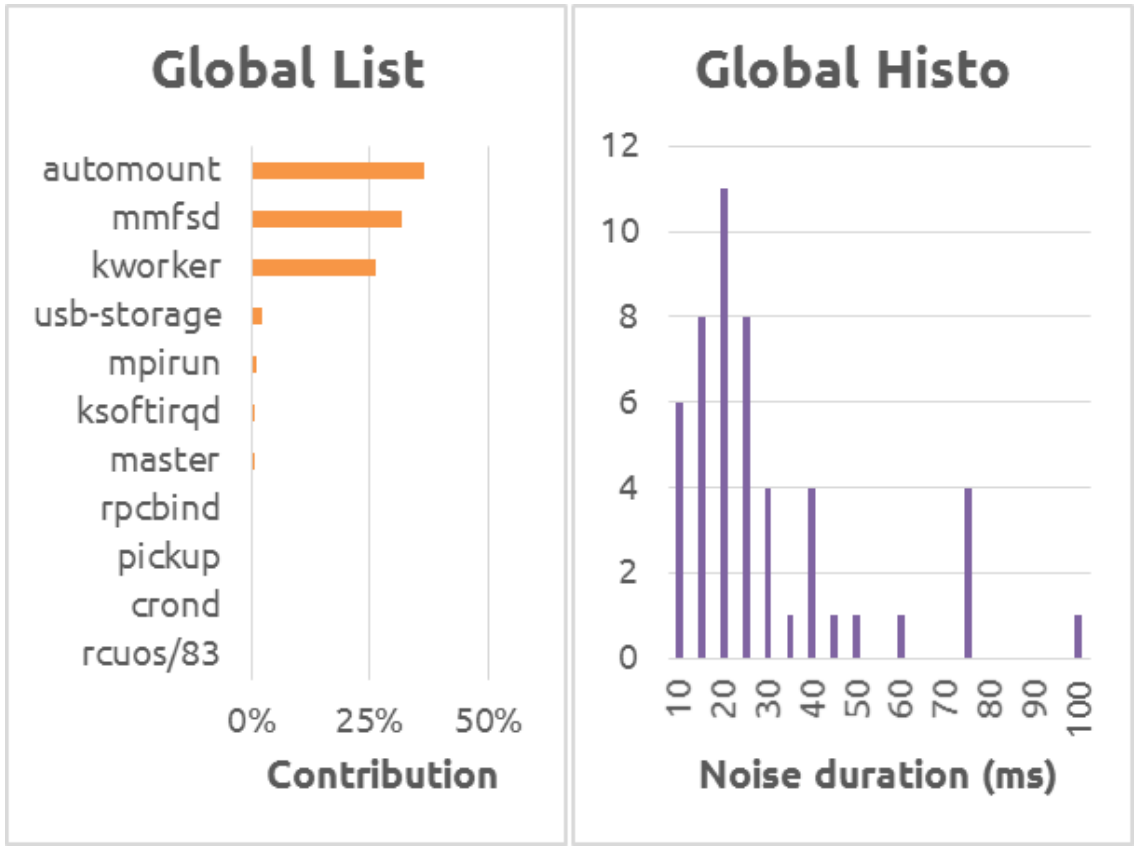
Experimental Platform

Property	Value
OS	Red Hat Enterprise Linux Server 7.3
CPU	POWER8 3.5 GHz
Number of sockets	2
Cores per socket	10
Threads per core	8
Total cores	160 (per node, SMT8)
Memory	512 GiB DDR3 1333 MHz

- IBM Spectrum MPI 10.1
- Job submission/management: IBM Spectrum LSF 10.1

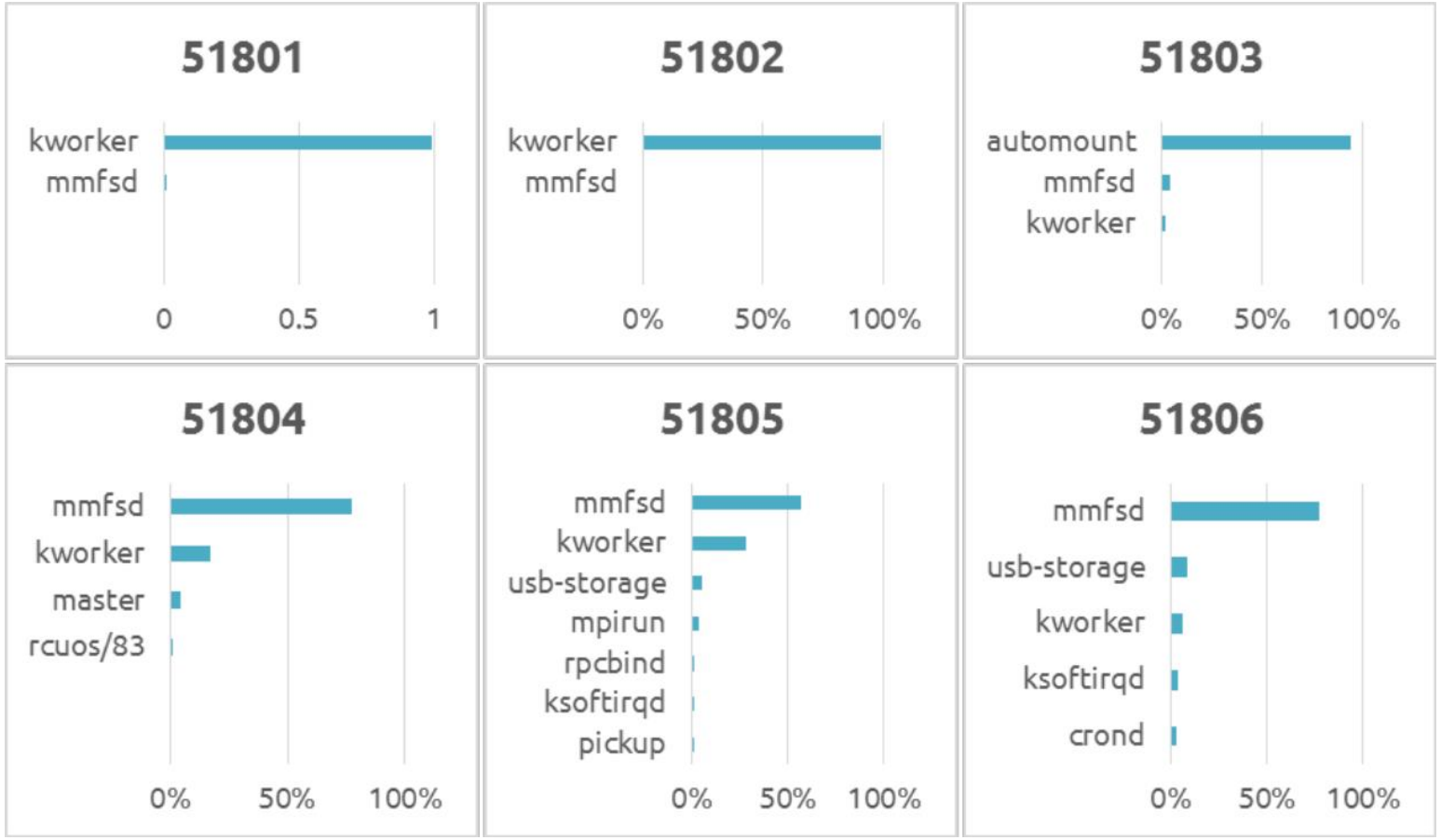
Experiments using PSNAP (1)

- PAL System Noise Activity Program, benchmark typically used to quantify noise from application perspective
- **Global List of Offenders**
- **Global Histogram of Noise Events**



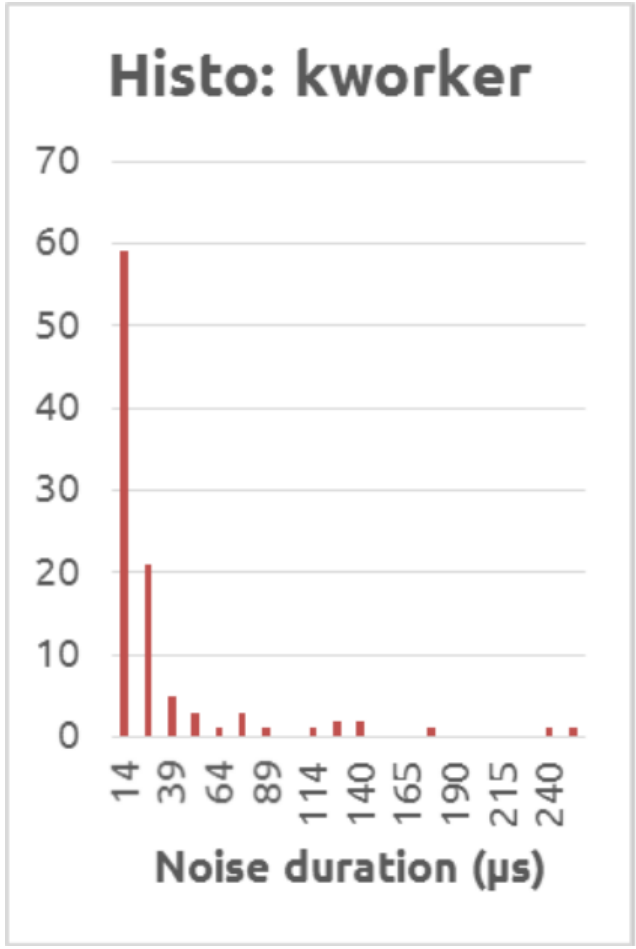
Experiments with PSNAP (2)

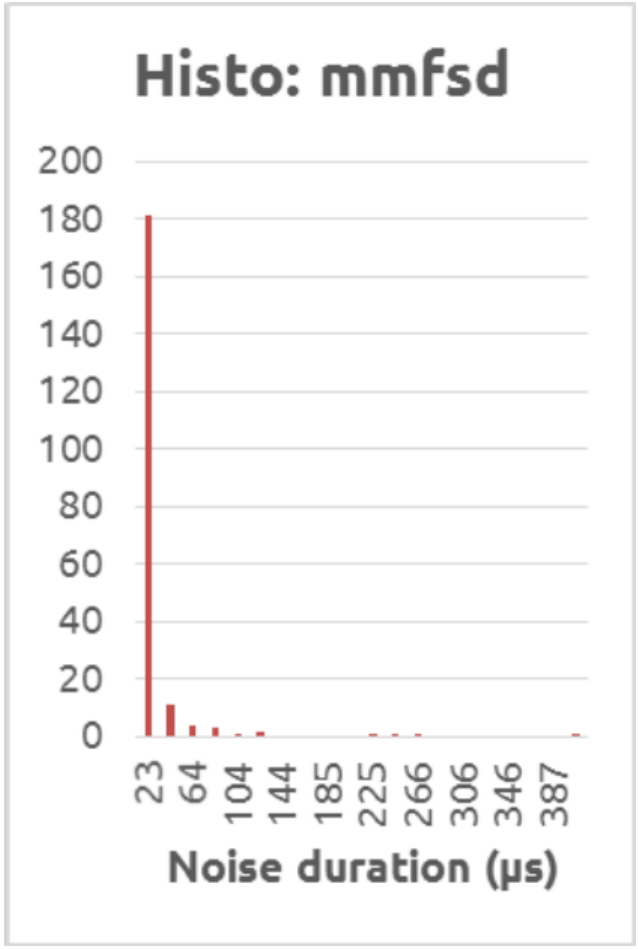
- **Local List of Offenders**
  (list per application PID)



Experiments with PSNAP (3)

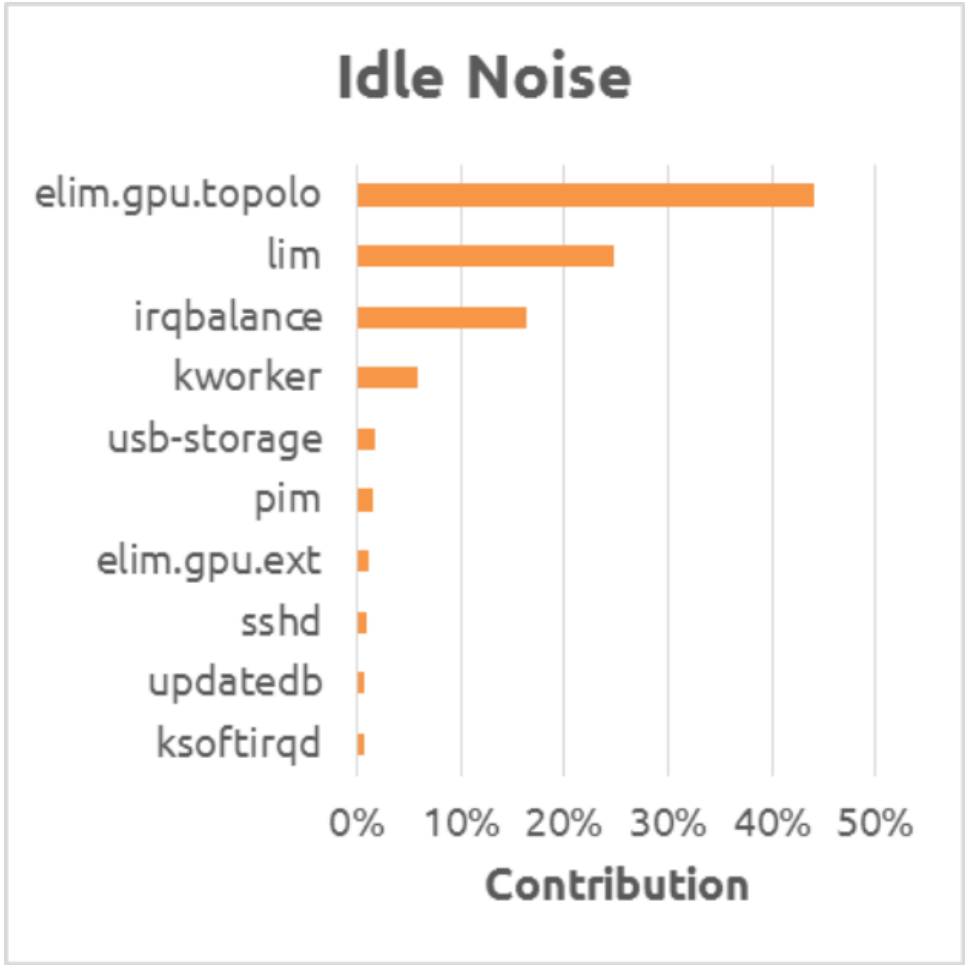
- **Histograms per Offender**





Experiments on Idle System

- **System-wide analysis** (identify all processes running on an idle system)



Jitter-Trace Overhead

- The processing of noise data is done after the execution of the application
  - Only potential source of overhead: **Perf**
- Overhead quantification: execution of **benchmark** (*konstant*)
  - One instance per CPU hardware thread (total of 160 processes)
  - CPU affinity set for Perf (one VCPU)
  - Internal application counters used to evaluate noise
- Total of 3.2 billion iterations (1.6 million less than case without tracing)
  - **Overhead of 0.05%**
  - This is also consistent to the overhead computed using Jitter-Trace itself (analyzing the statistics about Perf)

Conclusions

- **Jitter-Trace**
  - Built on top of Linux Perf
  - Identifies sources of jitter (noise) and quantifies them
  - Analyzes switching activity from kernel scheduler
- **Limitations**
  - Perf requires **administrative privileges** or the configuration of a kernel parameter to allow unprivileged tracing
  - Jitter-Trace does not account for jitter **sources** that are not processes
    - Newer implementation (after submitting this paper) provides support to other events
- **Future Work**
  - Work queue events (**kworker** detailed information) – done
  - **Interrupts** (hardware and software) – in progress
  - **Open source** – in progress

Thank you!