Fast by Friday

Why Kernel Superpowers are Essential

Brendan Gregg

What would it take

to solve **any** computer performance issue

in 5 days?

Imagine solving the performance of *anything*

Operating systems, kernels, web browsers, phones, applications, websites, microservices, processors, AI, etc., …

Examples: Linux, Windows, Firefox, Google docs, Minecraft, Amazon.com, Intel GPUs, pytorch, etc., …

Websites should load in the blink of an eye.

Why

Timely performance analysis allows **faster** and more **efficient** software/hardware/tuning options to be adopted

Good for the environment: Less cycles, energy, carbon

Good for innovation: Rewards investment in engineering

Good for companies: Less compute expense

Good for end-users: Lower latency, cheaper products

A vision:

"Fast by Friday":

Any computer performance issue

reported on Monday

should be solved by Friday

(or sooner)

Definitions

"Fast by Friday":
Any computer performance issue
reported on Monday
should be solved by Friday
(or sooner)

Issues: any performance analysis task, especially SW/HW evaluations

Solved by friday: doesn't mean fixed, it means root cause(s) known

"Fast by Friday" is…

A vision

A way of thinking

A call to action

A methodology

A practical deadline

I want to completely understand the performance of everything…in 5 days

The first of three activities

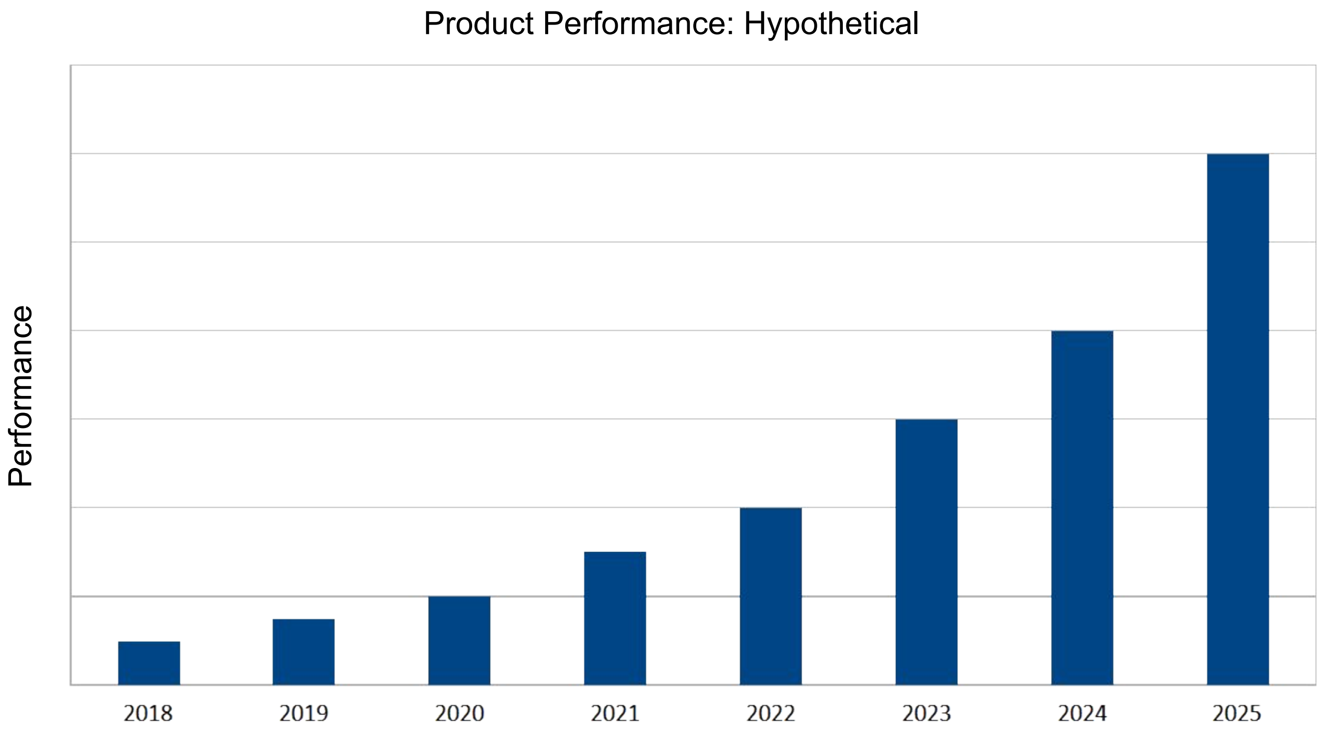
1.	Found	Performance root cause(s) known
2.	Fixed	Fix developed
3.	Deployed	Fixed everywhere

"Fast by Friday" focuses on (1) as it's often the biggest obstacle.

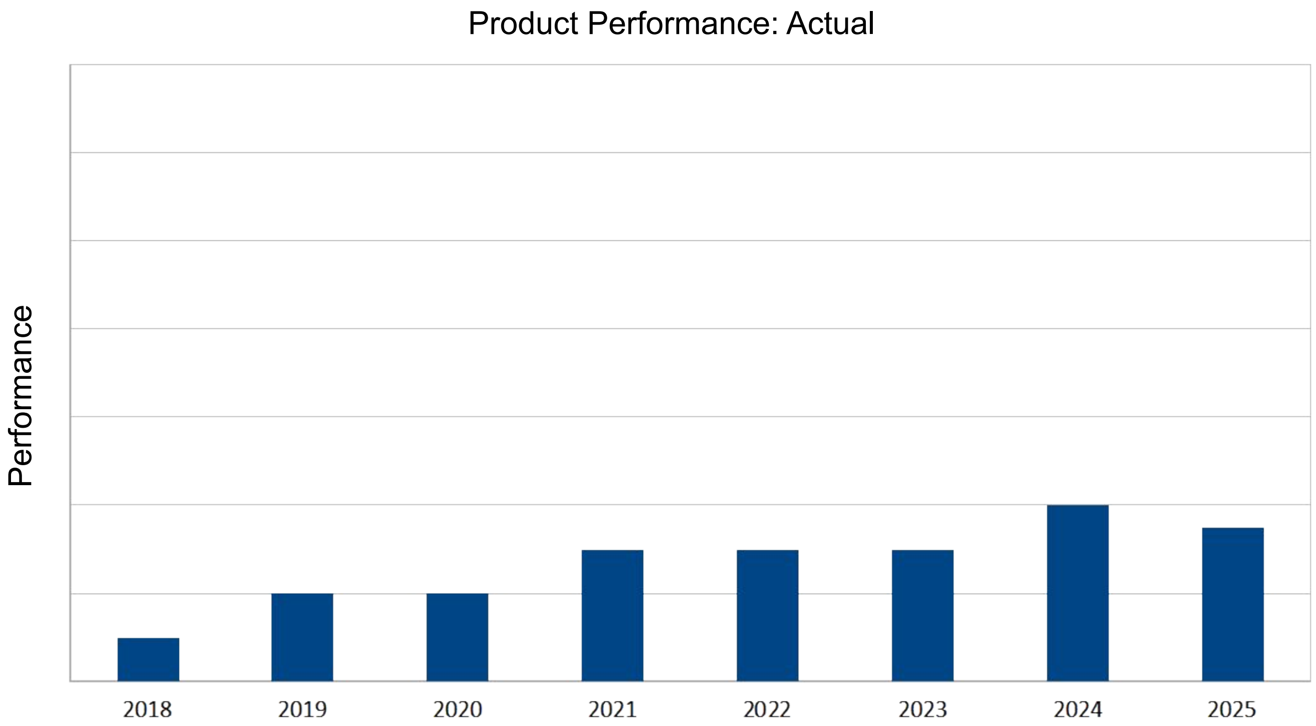
Yes, even for the Linux kernel. Show me a 2x perf fix and I'll show you comparies running it by Friday.
If the wasted cores paper was widely applicable, I'd have a pretty good example.

The Problem

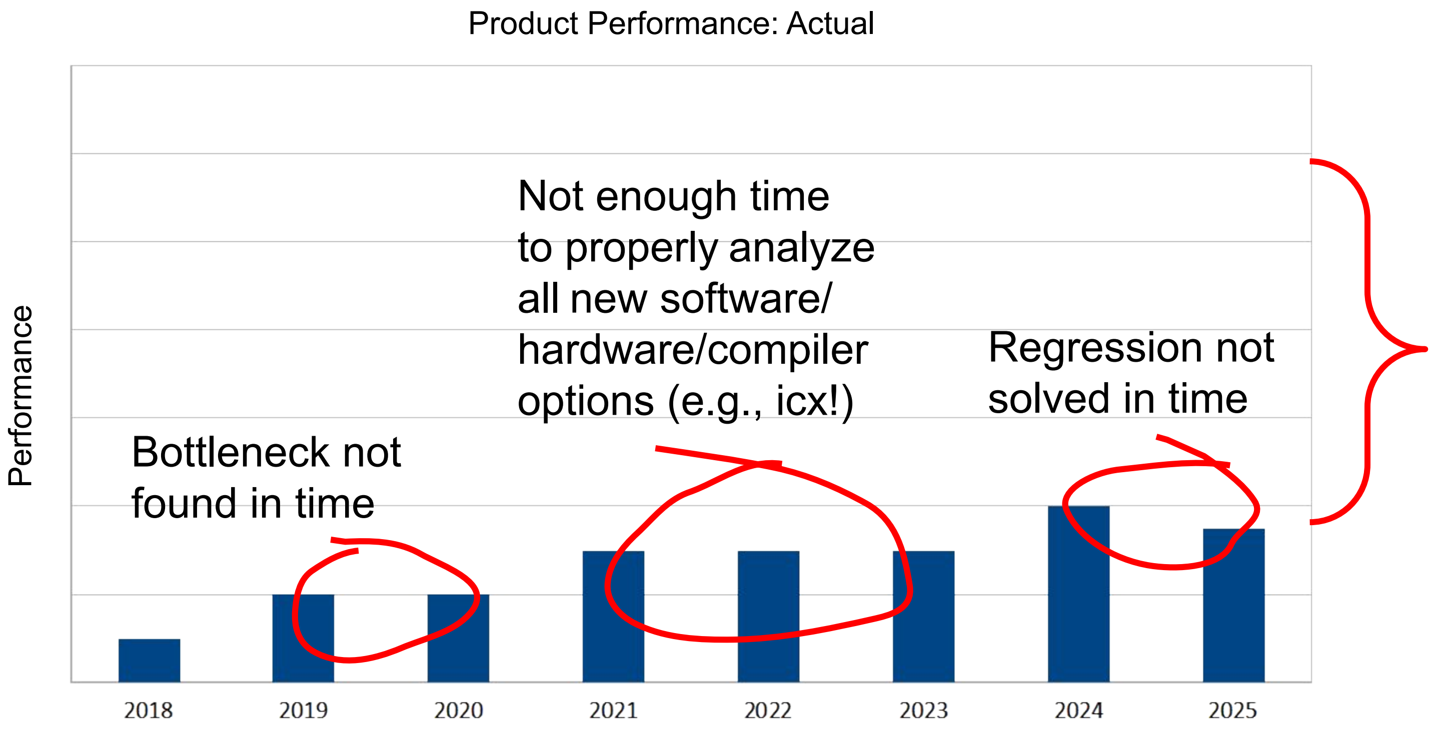
Expected performance improvement for computing products



Example reality



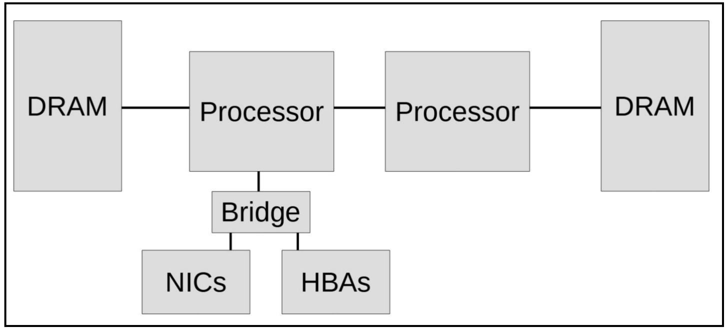
Example reality: 3 issues

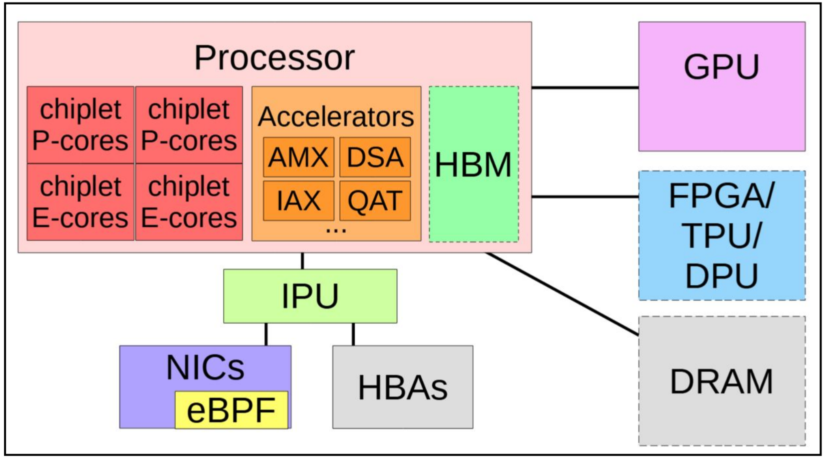


Amount of lost performance

We, engineers, have to fix this!

Problem: Computers are getting increasingly complex





Just one example (computer hardware) of increasing complexity.
Software is worse!

Performance issues can now go **unsolved for weeks, months, years**

Product decisions **miss improvements** as analysis and tuning takes too long

Analogy: Car performance

You build the world's fastest car, but the customer says: "it isn't"

You investigate and discover:

They were sent the wrong car

… with flat tires

… unbalanced wheels

… a minor engine issue

… and older firmware

They also weren't told how to drive it

… and left economy enabled

… and didn't use the turbo button

This may take too long to debug and the customer may leave.

Computers are like this too!

A common scenario at product vendors

Your product is *probably* the fastest

But there's likely some config/tunable error

It's the final week of the customer eval

You have to make it *fast by friday*

How

"Fast by Friday": Proposed Agenda

Prior weeks:	**Preparation**
Monday:	**Quantify, static tuning, load**
Tuesday:	**Checklists, elimination**
Wednesday:	**Profiling**
Thursday:	**Latency, logs, critical path**
Friday:	**Efficiency, algorithms**
Post weeks:	**Case study, retrospective**

Prior weeks: **Preparation**

Everything must work on Monday!

- ❏ Critical analysis tools ("crisis tools") must be preinstalled; E.g., Linux: `procps`, `sysstat`, `linux-tools-common`, `bcc-tools`, `bpftrace`, …
- ❏ **Stack tracing and symbols** should work for the kernel, libraries, and applications
- ❏ Tracing (host & distributed) must work
- ❏ The performance engineers must already have host **SSH root access**
- ❏ A functional diagram of the system must be known
- ❏ Source code should be available

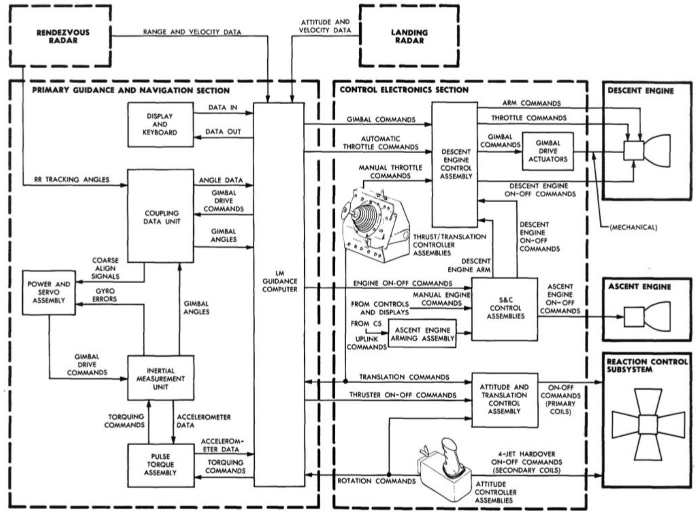


Figure 3-2.4. Primary Guidance Path - Simplified Block Diagram

Example functional diagram
Source: Lunar Module - LM10 Through LM14 Familiarization Manual" (1969):

Current industry status: 1 out of 5

Prior weeks: "Crisis Tools"

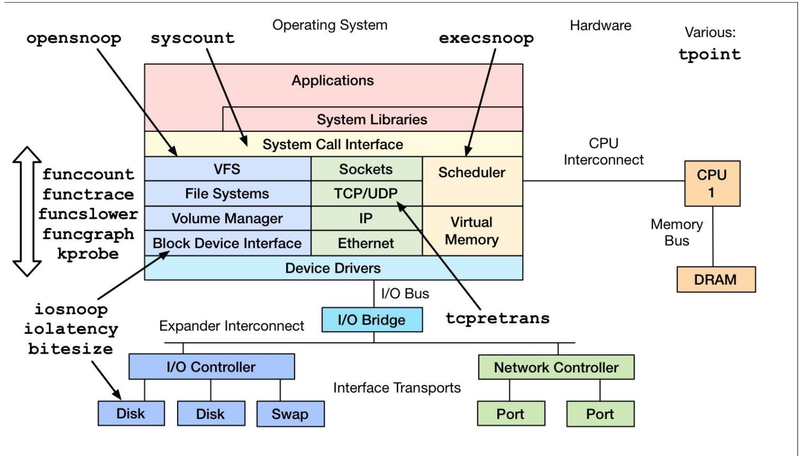
Table 4.1 **Linux crisis tool packages**

Package	Provides
procps	ps(1), vmstat(8), uptime(1), top(1)
util-linux	dmesg(1), lsblk(1), lscpu(1)
sysstat	iostat(1), mpstat(1), pidstat(1), sar(1)
iproute2	ip(8), ss(8), nstat(8), tc(8)
numactl	numastat(8)
linux-tools-common linux-tools-$(uname -r)	perf(1), turbostat(8)
bcc-tools (aka bpfcc-tools)	opensnoop(8), execsnoop(8), runqlat(8), runqlen(8), softirqs(8), hardirqs(8), ext4slower(8), ext4dist(8), biotop(8), biosnoop(8), biolatency(8), tcptop(8), tcplife(8), trace(8), argdist(8), funccount(8), stackcount(8), profile(8), and many more
bpftrace	bpftrace, basic versions of opensnoop(8), execsnoop(8), runqlat(8), runqlen(8), biosnoop(8), biolatency(8), and more
perf-tools-unstable	Ftrace versions of opensnoop(8), execsnoop(8), iolatency(8), iosnoop(8), bitesize(8), funccount(8), kprobe(8)
trace-cmd	trace-cmd(1)
nicstat	nicstat(1)
ethtool	ethtool(8)
tiptop	tiptop(1)
msr-tools	rdmsr(8), wrmsr(8)
github.com/brendangregg/msr-cloud-tools	showboost(8), cpuhot(8), cputemp(8)
github.com/brendangregg/pmc-cloud-tools	pmcarch(8), cpucache(8), icache(8), tlbstat(8), resstalls(8)

Source: Systems Performance 2nd Edition, page 131-132

No time to "`apt-get update; apt-get install...`" during a perf crisis.

Ftrace is great as it's usually there; my Ftrace/perf tools:



https://github.com/brendangregg/perf-tools

Monday: **Quantify, static tuning, load**

1. Quantify the problem
   - Problem statement method
2. Static performance tuning
   - The system without load
   - Check all hardware, software versions, past errors, config
   - Covered in sysperf
3. Load vs implementation
   - Just a problem of load?
   - Usually solved via basic monitoring and line charts

Current industry status: 4 out of 5

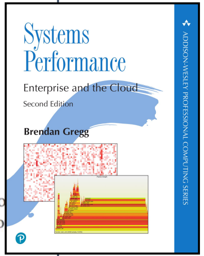
2.5.5 Problem Statement

Defining the problem statement is a routine task for support staff when first responding to issues. It's done by asking the customer the following questions:

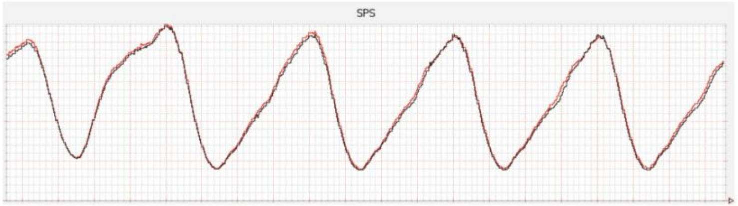
1. What makes you think there is a performance problem?
2. Has this system ever performed well?
3. What changed recently? Software? Hardware? Load?
4. Can the problem be expressed in terms of latency or runtime?
5. Does the problem affect other people or applications (or is it just you)?
6. What is the environment? What software and hardware are used? Versions? Configuration?

Just asking and answering these questions often points to an immediate cause and solutio The problem statement has therefore been included here as its own methodology and sho be the first approach you use when tackling a new issue.

I have solved performance issues over the phone by using the problem statement method alone, and without needing to log in to any server or look at any metrics.



Problem Statement method
Source: Systems Performance 2nd edition, page 44



A familiar pattern of load
Source: https://www.brendangregg.com/Slides/SREcon_2016_perf_checklists

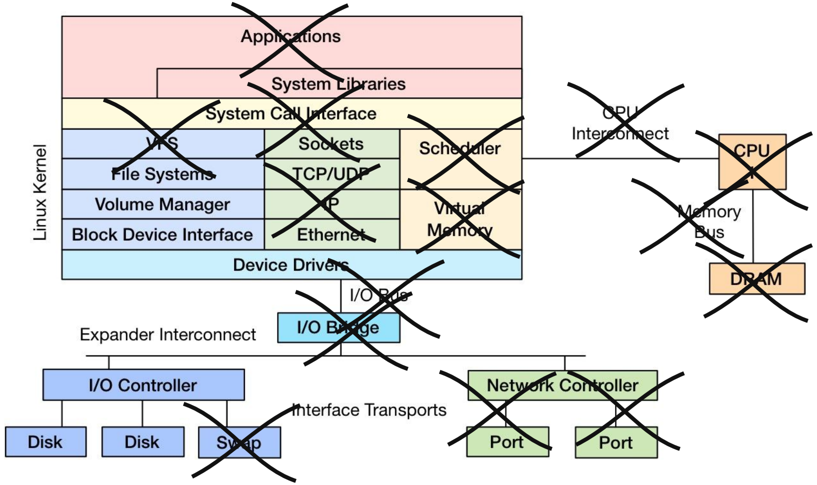
Monday (cont.): **End-of-day Status**

If still unsolved, we now know:

- It’s a real issue, of this magnitude, affecting these systems
- It’s not just config
- It’s not just load

Tuesday: Checklists, elimination

1. Recent issue checklist
   - Often need **new tools** for ad hoc checks
   - Can now be automated by AI auto-tuners (e.g., Intel Granulate)
2. Elimination: Subsystems it isn't
   - It's impossible to deep-dive everything in one week, need to narrow down
   - **New tools to exonerate** components
   - Dashboards of health check traffic lights
   - Include experiments: microbenchmarks



Generic system diagram

Current industry status: 2 out of 5

New observability tools often need kernel superpowers

We need new tools for broad and deep custom performance analysis, ideally that can be developed and run in-situ by Friday. No restarts.

eBPF is a kernel superpower that makes this possible.
(e.g., show me how much workload A queued behind workload B: This is not just queue latency histograms, but needs programmatic filters.)

Ftrace/perf/perf+eBPF also have kernel superpowers in the hands of wizards.

eBPF

Ftrace

perf

Tuesday (cont.): eBPF Tools

Current eBPF tools

***snoop, *top, *stat, *count, *slower, *dist**

Supports later methodologies

Workload characterization, latency analysis, off-CPU analysis, USE method, etc.

Future elimination tools

***health, *diagnosis**

Supports "fast by friday"

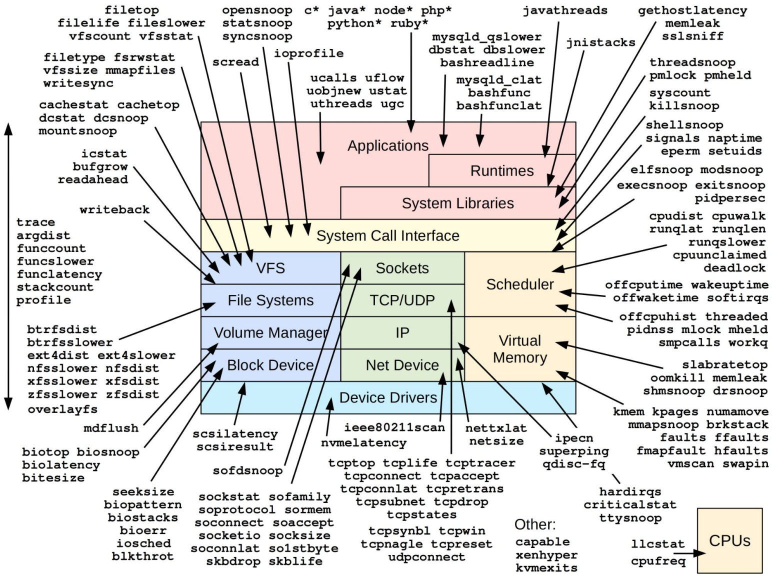
Analyzes existing dynamic workload

Open source & in the target code repo

E.g., Linux subsystem tools should be *in Linux*, like unit tests, accepted by maintainers, and ideally written *by the developers*! E.g., dctcphealth should ideally be written by the dctcp author: Daniel Borkmann!

This ensures they are accurate and maintained.

They should not be in bcc/bpftrace or proprietary.



Current eBPF performance tools

Source: BPF Performance Tools, cover art [Gregg 2019]

Tuesday (cont.): Health Tool Example 1/2

I wrote the ZFS L2ARC (second level cache) so I should write the health check tool, or at least share thoughts for others to follow:

- I designed it to either help or do nothing, so shouldn't be an issue, but... It could burn CPU for scanning, memory for metadata, and disk I/O throughput for caching, and not providing a net win, especially if someone set the record size to very small. Plus there could be outright bugs by new: There was that ARC bug I talked about at the last KR.
- Experimental is easiest: It's a cache, so turn it off! Are things now faster or slower?
- Accurate observability is hard: Measure CPU burn (profiling or eBPF tracing), disk I/O, and impact of L2ARC kernel metadata preventing app WSS from caching, but measuring WSS is hard, and my website is overdue an update www.brendangregg.com/wss.html
- Rough observability: From kernel counters: Is the L2ARC in use? Is the recsize <32k? Is it constantly scanning (CPU)? Is there heavy disk I/O (contention)? Then "maybe".
- I have more thoughts and this should become a bcc tool request ticket. When it's your own code, you know a lot of "however"s!

Tuesday (cont.): Health Tool Example 2/2

I wrote the ZFS L2ARC (second level cache) so I should write the health check tool, or at least share thoughts for others to follow:

- I designed it to either help or do nothing, so shouldn't be an issue, but... It could burn CPU for

In summary, a practical L2ARC health tool could:

1. Use kernel counters to check for possible resource contention versus handpicked thresholds, and report "good" or "maybe issue".
2. If maybe, prompt for an invasive test that disables the L2ARC while monitoring systemic throughput. Report "good" or "bad" and quantify.

If needed can measure contention via kprobe/kfunc tracing and eBPF.

The tool should be in ZFS and its logic and thresholds maintained.

code, you know a lot of "however"s!

Tuesday (cont.): Health Tool Points

A. An ugly half-good tool is better than nothing
B. Sharing thoughts can let others write it (Documentation/*/health.txt)
C. Reporting "maybe" is ok
D. Not an C64 diagnostics cart: Has to analyze exsiting workloads
E. Test hierarchy: safe -> violent, only progress if needed, can prompt
F. Be pragmatic: eBPF, perf, Ftrace, /proc, use anything

Current tools: "Here's data, you figure it out"

Health tools: "I figured it out"

Tuesday (cont.): **End-of-day Status**

If still unsolved, we now know:

- It’s a real issue, of this magnitude, affecting these systems
- It’s not just config
- It’s not just load
- It’s not a recent issue
- It’s caused by these components

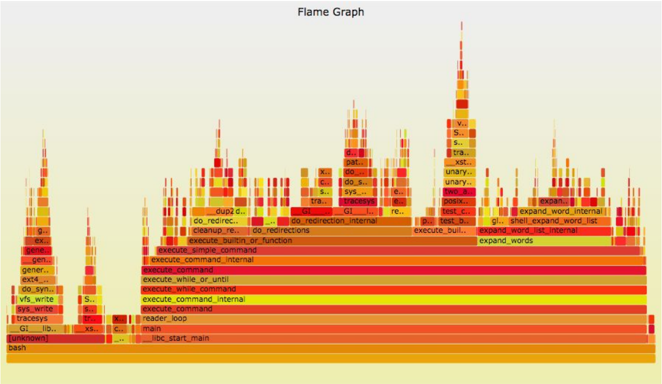
Wednesday: **Profiling**

1. CPU Flame Graphs
   - More efficient with eBPF
   - eBPF runtime stack walkers
2. CPI Flame Graphs
   - Needs PMCs PEBS on Intel for accuracy
3. Off-CPU Flame Graphs
   - Impractical without eBPF

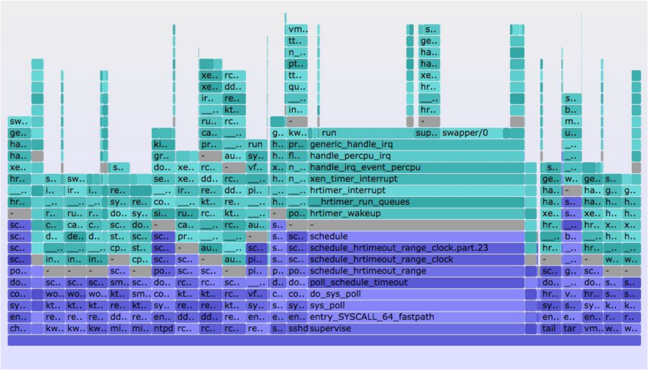
Solves most performance issues

Needs preparation!

Current industry status: 3 out of 5



CPU flame graph



Off-CPU/waker time flame graph

Wednesday (cont.): **End-of-day Status**

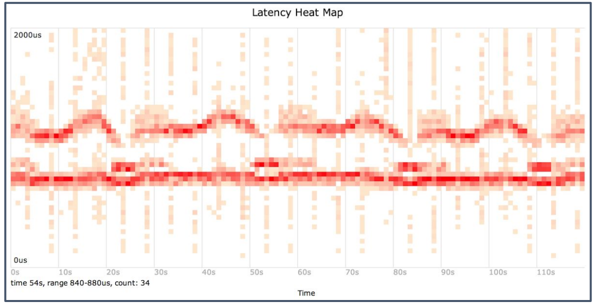
If still unsolved, we now know:

- It’s a real issue, of this magnitude, affecting these systems
- It’s not just config
- It’s not just load
- It’s not a recent issue
- It’s caused by these components
- It’s caused by these codepaths

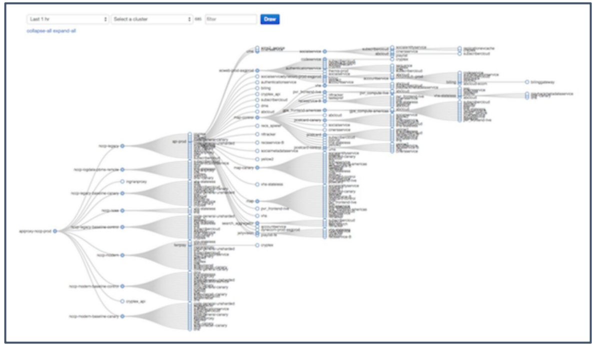
Thursday: **Latency, logs, critical path, HW**

1. Latency drilldowns
    - Latency histograms
    - Latency heat maps
    - Latency outliers
2. Logs, event tracing
    - Custom event logs
3. Critical path analysis
    - Multi-threaded tracing
    - Distributed tracing across a distributed environment
4. Hardware counters

Current industry status: 3 out of 5



Latency heat maps
Source: https://www.brendangregg.com/HeatMaps/latency.html



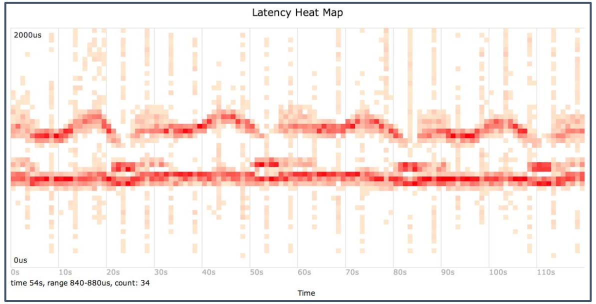
Distributed tracing
Source: https://www.brendangregg.com/Slides/Monitorama2015_NetflixInstanceAnalysis

Thursday: **Latency, logs, critical path, HW**

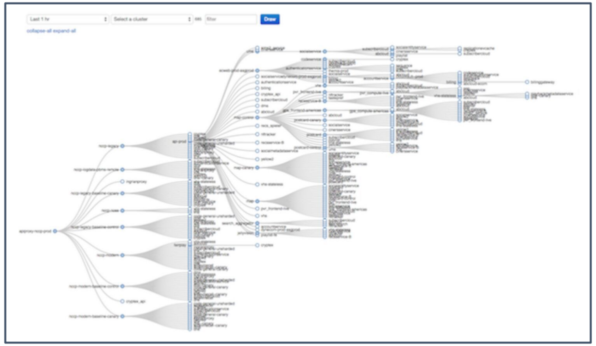
eBPF Tools

1. Latency drilldowns
   - Latency histograms ← ***dist**
   - Latency heat maps
   - Latency outliers ← ***slower**
2. Logs, event tracing
   - Custom event logs ← ***snoop, bpftrace**
3. Critical path analysis
   - Multi-threaded tracing
   - Distributed tracing across a distributed environment ← **"Zero instrumentation"** **(when faster uprobes is done; currently: https://dont-ship.it)**
4. Hardware counters ← **perf & its subcommands**

Current industry status: 3 out of 5



Latency heat maps
Source: https://www.brendangregg.com/HeatMaps/latency.html



Distributed tracing
Source: https://www.brendangregg.com/Slides/Monitorama2015_NetflixInstanceAnalysis

Thursday (cont.): **End-of-day Status**

If still unsolved, we now know:

- It’s a real issue, of this magnitude, affecting these systems
- It’s not just config
- It’s not just load
- It’s not a recent issue
- It’s caused by these components
- It’s caused by these codepaths
- Latency has this distribution, over time, and these outliers
- Latency is coming from this specific component
- It's not a low-level hardware issue

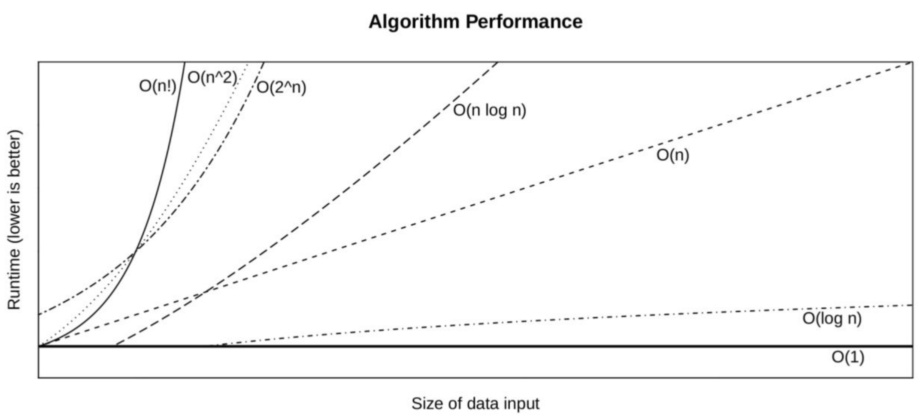
Friday: **Efficiency, algorithms**

1. Is the target *efficient*?
    - A largely unsolved problem
    - Cycles/carbon per request
    - Compare with similar products
    - **New efficiency tools (eBPF?)**
    - System efficiency equals the least efficient component
    - Modeling, theory
2. Use faster algorithms?
    - Big O Notation

Current industry status: 1 out of 5

Protocol	CIFS	iSCSI	FTP	NFSv3	NFSv4
Cycles(k) per 1k read	2241	1843	970	395	485

Example efficiency comparisons (made up)



Source: Systems Performance 2nd Edition, page 175

Friday (cont.): **End-of-day Status**

If still unsolved, we now know:

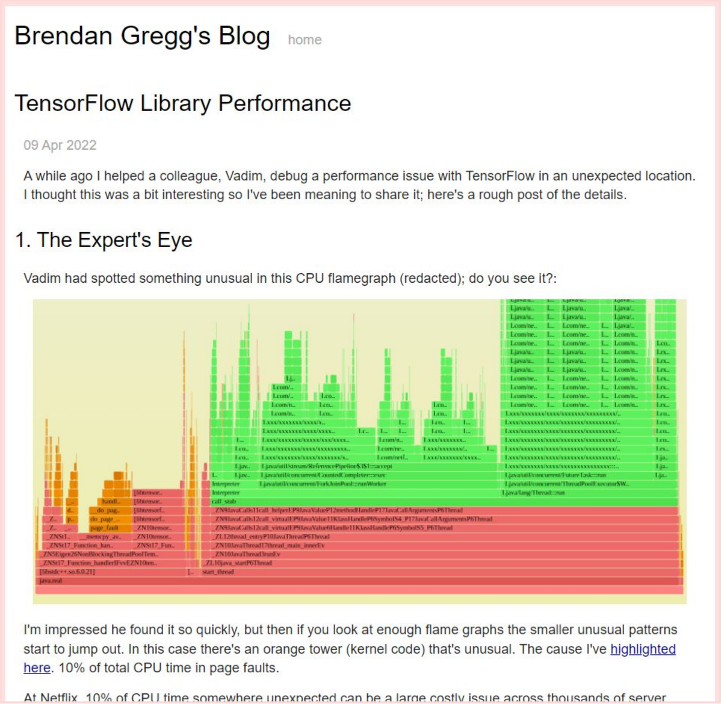
- It’s a real issue, of this magnitude, affecting these systems
- It’s not just config
- It’s not just load
- It’s not a recent issue
- It’s caused by this component
- It’s caused by these codepaths
- Latency has this distribution, over time, and these outliers
- Latency is coming from this specific component
- It's not a low-level hardware issue
- The code is efficient already. There is no “problem”!

Post weeks: **Case study, retrospective**

1. Document as a case study
    - JIRA, wiki, gist
    - External blog/talk

      Including (redacted) flame graphs is great: You may find overlooked perf issues years later from them.
    - Repetition?

      Add to Tuesday's "Recent issue checklist"
2. Retrospective
    - How to debug it faster **by friday**?


Brendan Gregg's Blog home

TensorFlow Library Performance

09 Apr 2022

A while ago I helped a colleague, Vadim, debug a performance issue with TensorFlow in an unexpected location. I thought this was a bit interesting so I've been meaning to share it; here's a rough post of the details.

1. The Expert's Eye

Vadim had spotted something unusual in this CPU flamegraph (redacted); do you see it?:

I'm impressed he found it so quickly, but then if you look at enough flame graphs the smaller unusual patterns start to jump out. In this case there's an orange tower (kernel code) that's unusual. The cause I've highlighted here. 10% of total CPU time in page faults.

At Netflix, 10% of CPU time somewhere unexpected can be a large costly issue across thousands of server

Example blog post: https://www.brendangregg.com/blog

Current industry status: 1 out of 5

"Fast by Friday": My current industry ratings (5 == best)

Prior weeks:	Preparation	1
Monday:	Quantify, static tuning, load	4
Tuesday:	Checklists, elimination	2
Wednesday:	Profiling	3
Thursday:	Latency, logs, critical path	3
Friday:	Efficiency, algorithms	1
Post weeks:	Case study, retrospective	1

We are not currently good at this

"Fast by Friday": Linux Kernel Superpowers

Prior weeks:	Preparation
Monday:	Quantify, static tuning, load
Tuesday:	Checklists, elimination
Wednesday:	Profiling
Thursday:	Latency, logs, critical path
Friday:	Efficiency, algorithms
Post weeks:	Case study, retrospective

eBPF
perf
Ftrace

What Needs to Change

A way of thinking, a call for action

Consider perf wins that took weeks as **room for improvement**

New tracing tools needed: ***diagnose, *health**

Crisis tools should be installed by default in enterprise distros

Stack walking should work by default for everything

Stack walking, frame pointers, and eBPF walking

Frame pointers already enabled at major companies.

- Fedora first distro to offer it?

Can't we be smarter if needed?

- NOP/__fentry__ style rewrites (Rostedt)? Options with LD/ELF.

eBPF custom runtime stack walkers (Java, etc.)

- Yes, multiple people are doing this. They should ship as open source with the runtime code.

Reasons FPs were disabled in 2004:

- i386
- gdb doesn't need them
- gcc vs icc

[PATCH] Omit frame pointer and fix %ebp by default on x86 (take 3)

```
The following patch is the latest revision of a patch to enable
-fomit-frame-pointer by default on x86.  The GDB and GCC's debugging
folks have done an impressive job supporting debugging without a
frame pointer, and it would be a shame if 3.5 didn't benefit from
those efforts.  As recently as a few hours ago, one of GCC's
benchmarking gurus reported new performance figures of GCC vs icc
without using "-fomit-frame-pointer" reflecting the need to get
better optimization with GCC's default flags.
```

https://gcc.gnu.org/legacy-ml/gcc-patches/2004-08/msg01033.html

Summary

"Fast by Friday" Summary

Prior weeks: **Preparation**

Day 1: **Quantify, static tuning, load**
Day 2: **Checklists, elimination**
Day 3: **Profiling**
Day 4: **Latency, logs, critical path**
Day 5: **Efficiency, algorithms**

Post weeks: **Case study, retrospective**

Fast by Friday:
Any computer performance issue reported on Monday should be solved by Friday (or sooner)

"*Fixed* by Friday" (a different talk) sample

Performance Mantras:

1. **Don't do it**
2. **Do it, but don't do it again**
3. **Do it less**
4. **Do it later**
5. **Do it when they're not looking**
6. **Do it concurrently**
7. **Do it cheaper**

Fixed by Friday:
Any known performance bug reported on Monday should have a fix by Friday (or sooner)

AFAIK these mantras are from Craig Hanson and Pat Crain (I'm still looking for a reference)

Take Aways

"Fast by Friday": Any computer performance issue reported on Monday should be solved by Friday (or sooner)

Kernel superpowers, especially eBPF, are essential for such fast in-situ production analysis

It will take all of us many years: OS changes, kernel support, new tools, methodologies. How can you help? One step at a time!

Q&A

Thanks

Jesper Dangaard Brouer

eBPF: Alexei Starovoitov (Meta), Daniel Borkmann (Isovalent), David S. Miller (Red Hat), Jakub Kicinski (Meta), Yonghong Song (Meta), Andrii Nakryiko (Meta), Thomas Graf (Isovalent), Martin KaFai Lau (Meta), John Fastabend (Isovalent), Quentin Monnet (Isovalent), Jesper Dangaard Brouer (Red Hat), Andrey Ignatov (Meta), Stanislav Fomichev (Google), Joe Stringer (Isolavent), KP Singh (Google), Dave Thaler (Microsoft), Liz Rice (Isovalent), Chris Wright (Red Hat), Linus Torvalds, and many more in the BPF community

Ftrace: Steven Rostedt (Google) and the Ftrace community

Perf: Arnaldo Carvalho de Melo (Red Hat) and the perf community

Kernel Recipes 10th edition!