CMP325

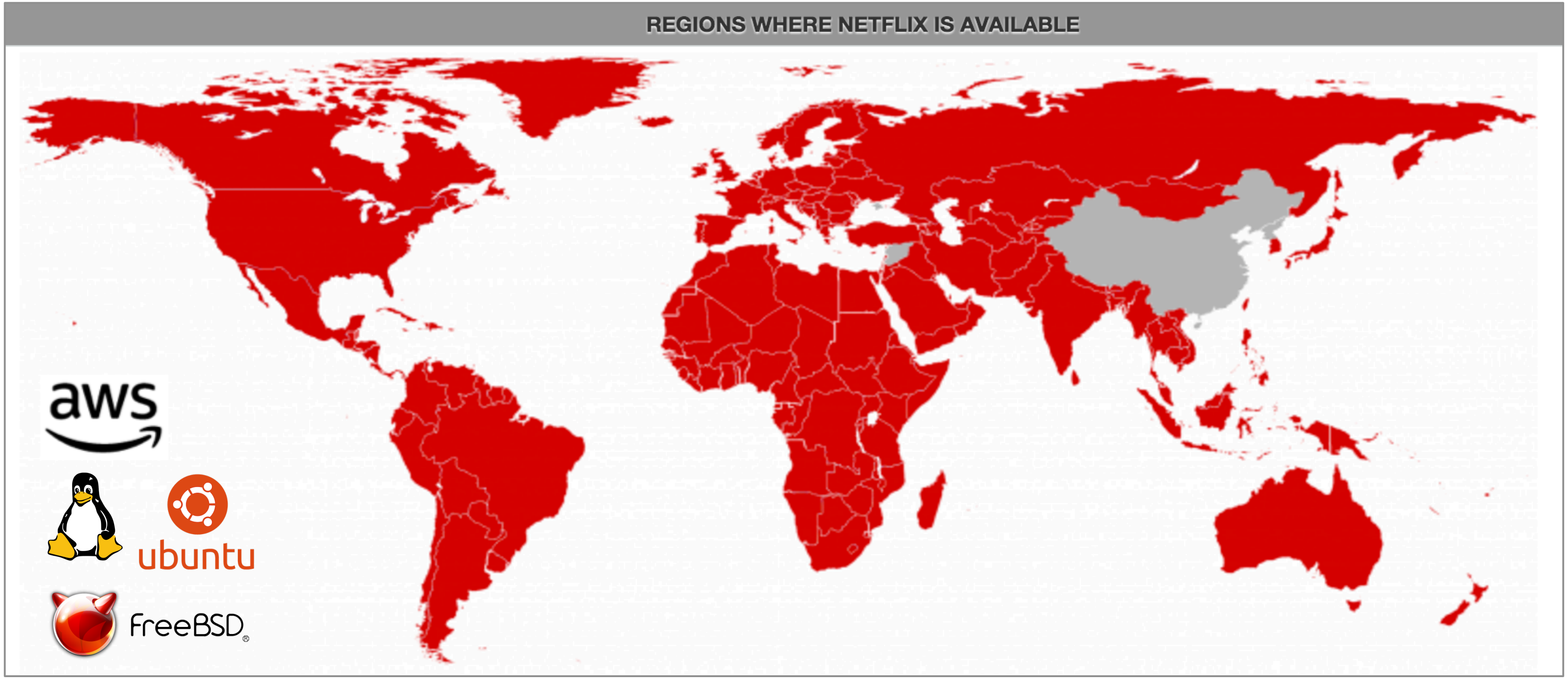
How Netflix Tunes EC2 Instances for Performance

Brendan Gregg, Performance and OS Engineering Team

November 28, 2017

AWS re:Invent

NETFLIX



AWS re:Invent

Netflix performance and operating systems team

- Evaluate technology
  - Instance types, Amazon Elastic Compute Cloud (EC2) options
- Recommendations and best practices
  - Instance kernel tuning, assist app tuning
- Develop performance tools
  - Develop tools for observability and analysis
- Project support
  - New database, programming language, software change
- Incident response
  - Performance issues, scalability issues

AWS re:Invent

Agenda

1. Instance selection
2. Amazon EC2 features
3. Kernel tuning
4. Methodologies
5. Observability

AWS re:Invent

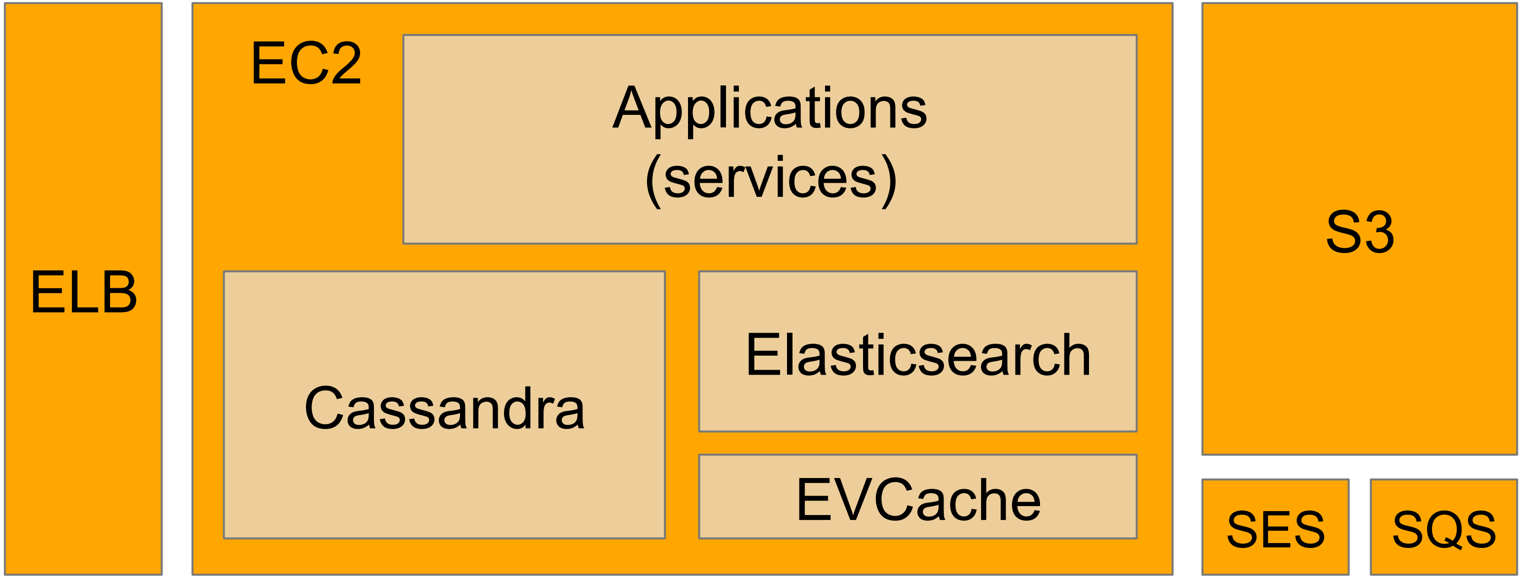
Warnings

- This is what’s in our medicine cabinet
- Consider these “best before: 2018”
- Take only if prescribed by a performance engineer

1. Instance selection

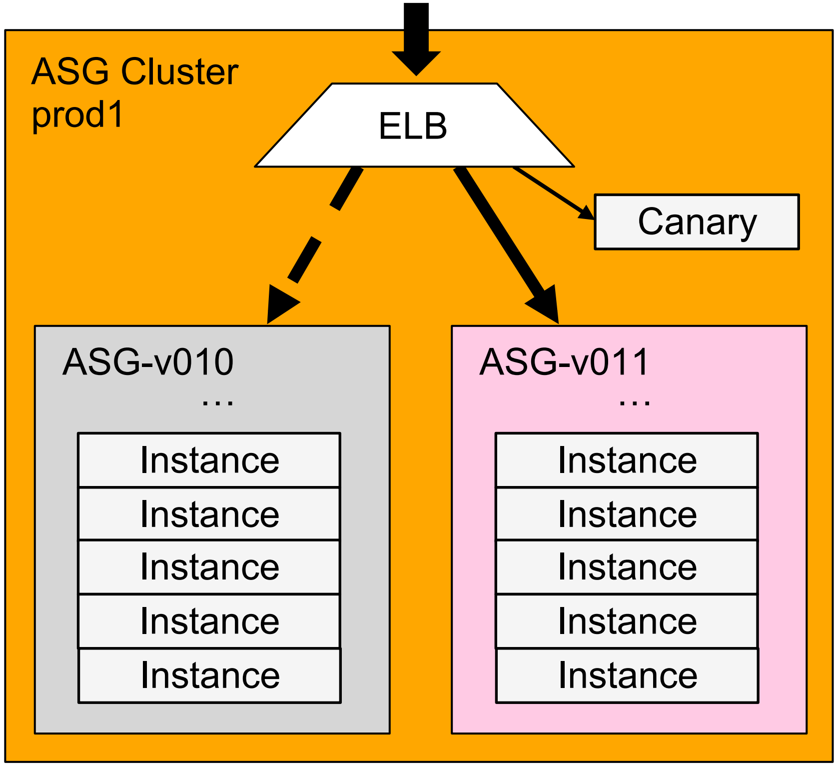
The Netflix cloud

Many application workloads: Compute, storage, caching…



Netflix AWS environment

- Elastic Load Balancing allows real load testing
  1. Single instance canary, then,
  2. Auto scaling group
- Much better than micro-benchmarking alone, which is error prone



Current generation instances

- Families:
  - **m4**: General purpose
    - Balanced
  - **c5**: Compute-optimized
    - Latest CPUs, lowest price/compute perf
  - **i3, d2**: Storage-optimized
    - SSD large capacity storage
  - **r4, x1**: Memory optimized
    - Lowest cost/Gbyte
  - **p2, g3, f1**: Accelerated computing
    - GPUs, FPGAs…
- Types: Range from medium to 16x large+, depending on family
- Netflix uses over 30 different instance types

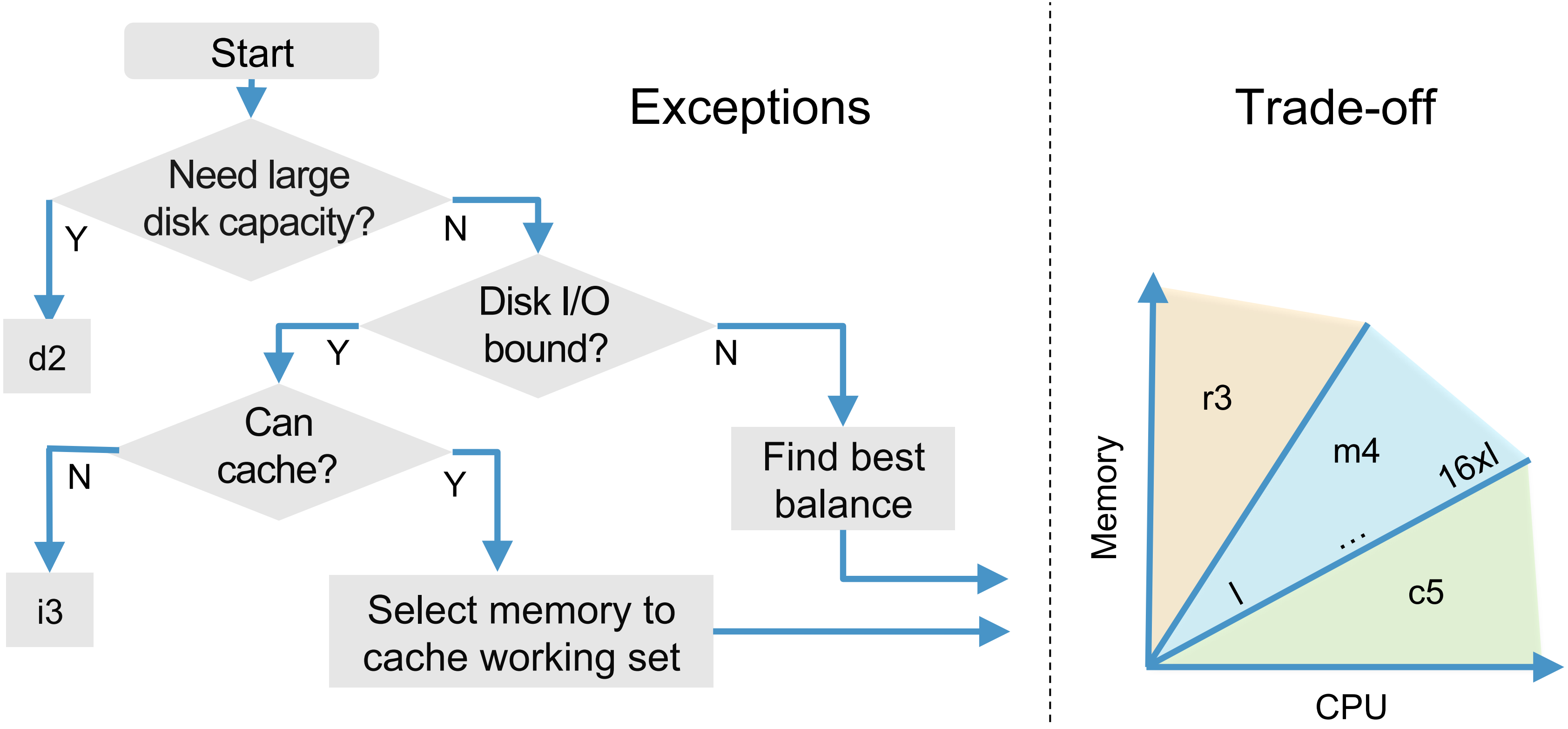
i3.8xlarge

AWS re:Invent

Netflix instance type selection

A. Flow chart
B. By-resource
C. Brute force

A. Instance selection flow chart



B. By-resource approach

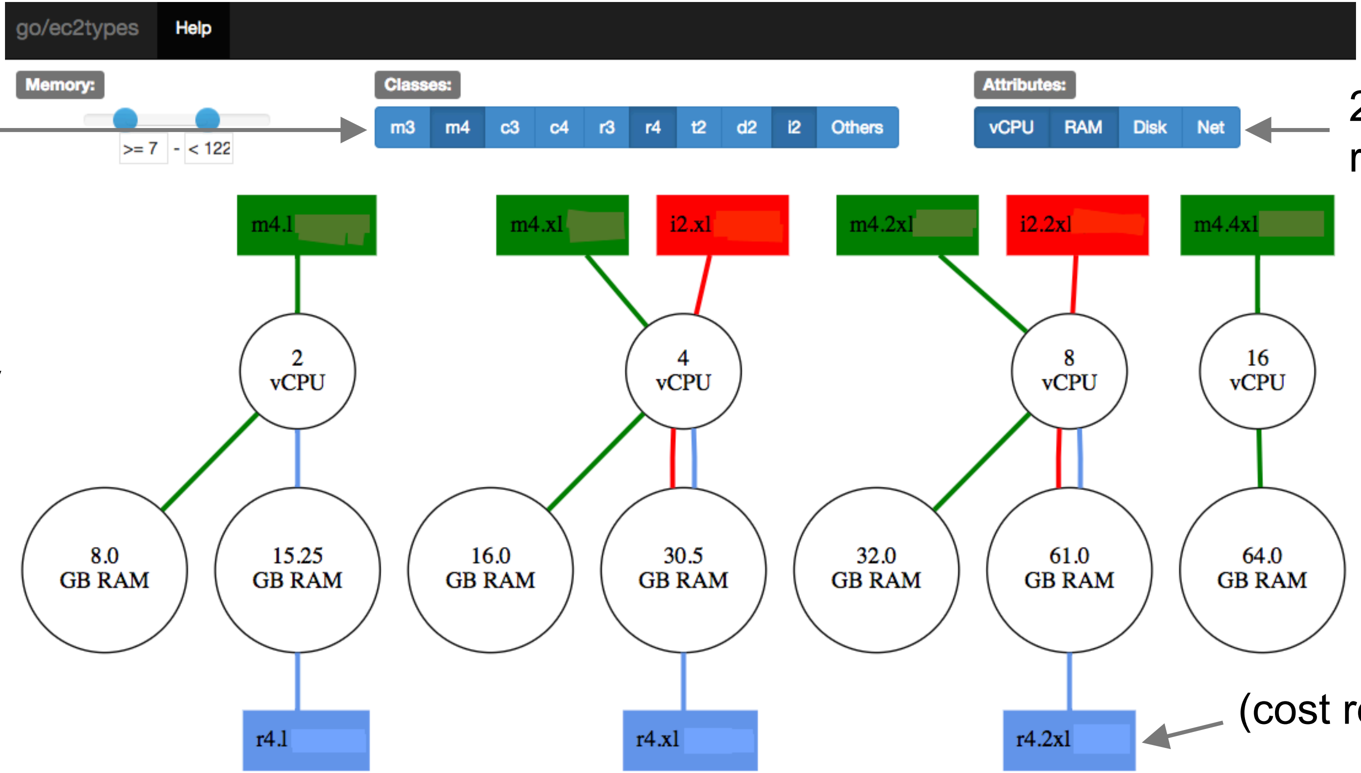
1. Determine bounding resource
   - E.g.: CPU, disk I/O, or network I/O
   - Found using:
     - Estimation (expertise)
     - Resource observability with an existing real workload
     - Resource observability with a benchmark or load test (experimentation)
2. Choose instance type for the bounding resource
   - If disk I/O, consider caching, and a memory-optimized type
   - We have tools to aid this choice: Nomogram Visualization

This focuses on optimizing a given workload

More efficiency can be found by adjusting the workload to suit instance types

Nomogram Visualization tool

1. Select instance families



2. Select resources

3. From any resource, see types and cost

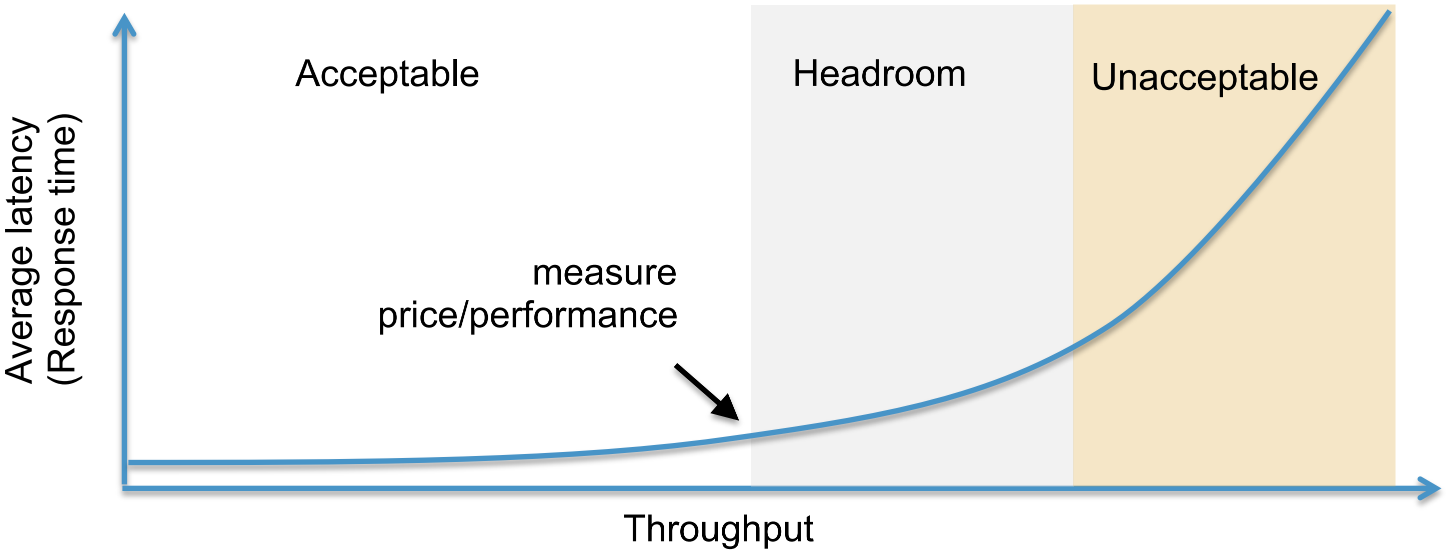
(cost redacted)

C. Brute force choice

1. Run load test on ALL instance types
   – Optionally, different workload configurations as well
2. Measure throughput
   – And check for acceptable latency
3. Calculate price/performance for all types
4. Choose most efficient type

Latency requirements

- Check for an acceptable latency distribution when optimizing for price/performance



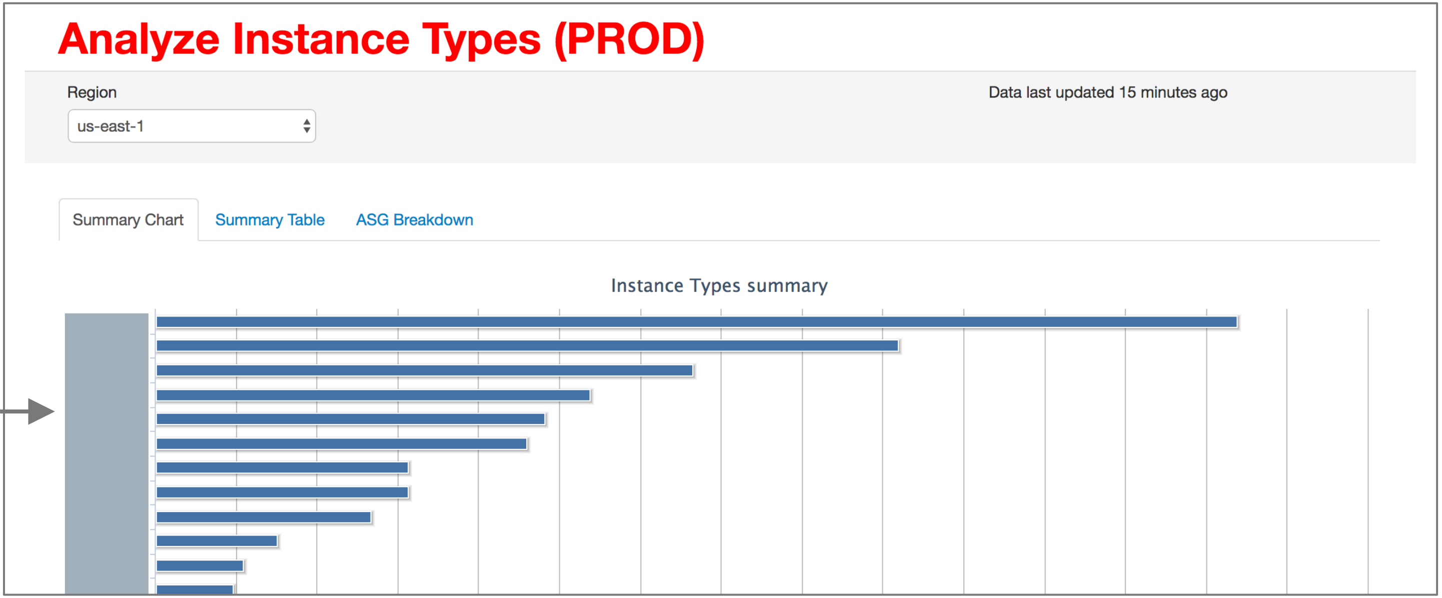
Netflix instance type re-selection

A. Usage
B. Cost
C. Variance

A. Instance usage

- Older instance types can be identified, analyzed, and upgraded to newer types

Types (redacted)



AWS re:Invent

B. Instance cost

- Also checked regularly. Tuning the price in price/perf.

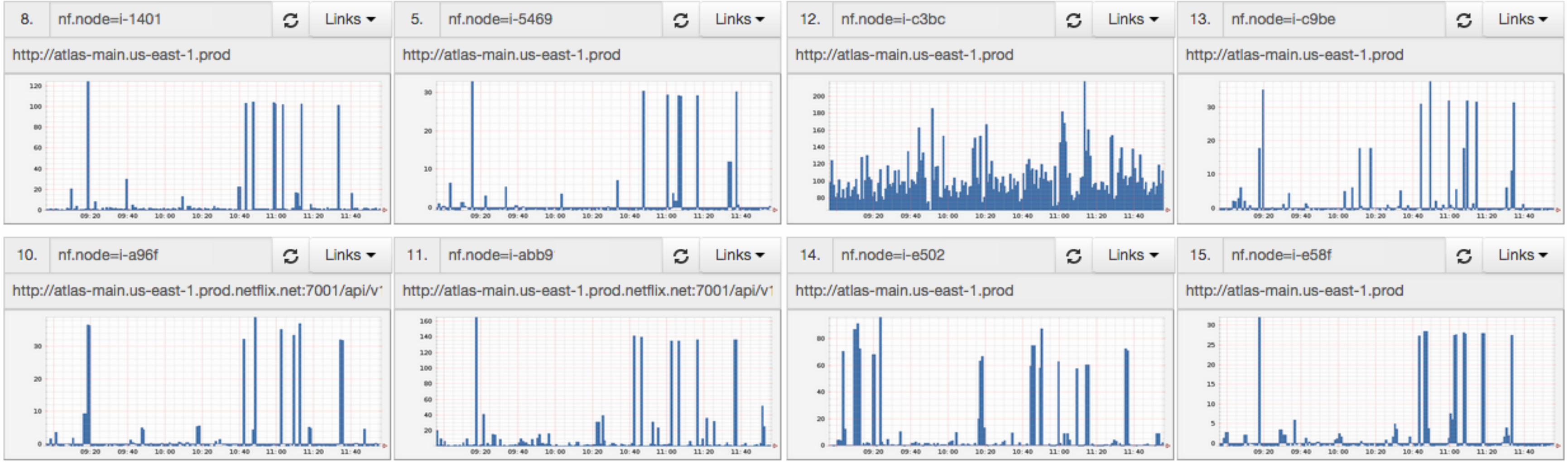
Breakdowns

Cost per hour

Details (redacted)

C. Instance variance

- An instance type may be resource-constrained only occasionally, or after warmup, or a code change
- Continually monitor performance, analyze variance/outliers



AWS re:Invent

aws

2. Amazon EC2 features

EC2 virtualization

- Bare-metal performance
- Near-metal performance
- Optimized performance
- Poor performance

Importance: Most → Least

Old → New

Tech	Type	With	CPU, Memory	Network I/O	Local Storage I/O	Remote Storage I/O	Interrupts, Timers	Motherboard, Boot
VM	Fully Emulated		VS	VS	VS	VS	VS	VS
VM	Xen HVM 3.0	PV drivers	VH	P	P	P	VS	VS
VM	Xen HVM 4.0.1	PVHVM drivers	VH	P	P	P	P	VS
VM	Xen AWS 2013	PVHVM + SR-IOV(net)	VH	VH	P	P	P	VS
VM	Xen AWS 2017	PVHVM + SR-IOV(net, stor.)	VH	VH	VH	P	P	VS
VM	AWS Nitro 2017		VH	VH	VH	VH	VH	VS
HW	AWS Bare Metal 2017		H	H	H	H	H	H
	Bare Metal		H	H	H	H	H	H

VM: Virtual Machine. HW: Hardware.
VS: Virt. in software. VH: Virt. in hardware. P: Paravirt. Not all combinations shown.
SR-IOV(net): ixgbe/ena driver. SR-IOV(storage): nvme driver.

slide updated after talk. see: http://www.brendangregg.com/blog/2017-11-29/aws-ec2-virtualization-2017.html

Networking SR-IOV

- AWS "enhanced networking"
  - Uses SR-IOV: Single Root I/O Virtualization
  - PCIe device provides virtualized instances
  - Some instance types, VPC only
- "Bare metal" network access
  - Higher network throughput, reduced RTT and jitter
  - ixgbe driver types: Up to 10 Gbps
  - ena driver types: Up to 25 Gbps

Storage SR-IOV

- New in 2017, first used by i3s
- Should be called "enhanced storage"
  - Some instance types only
  - Accesses NVMe attached storage (faster transport than SATA)
  - Uses VT-d for I/O virtualization
- "Bare metal" disk access
  - i3.16xl can exceed 3 million IOPS

https://aws.amazon.com/blogs/aws/now-available-i3-instances-for-demanding-io-intensive-applications/

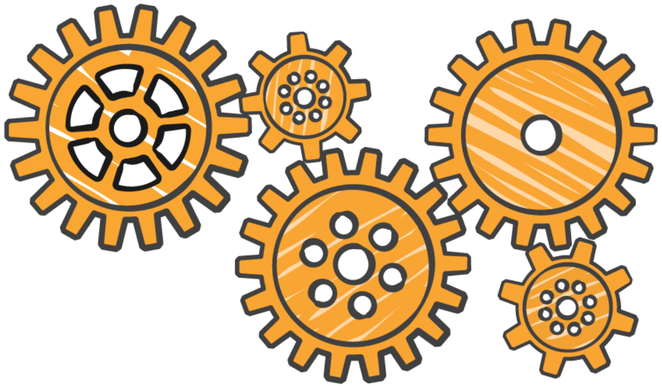
3. Kernel tuning

Kernel tuning

- Typically 1-30% wins, for average performance
  - Adds up to significant savings for the Netflix cloud
- Bigger wins when reducing latency outliers
- Deploying tuning:
  - Generic performance tuning is baked into our base AMI
  - Experimental tuning is a package add-on (nflx-kernel-tunables)
  - Workload-specific tuning is configured in application AMIs
  - Remember to tune the workload with the tunables
- We run Ubuntu Linux

Tuning targets

1. CPU scheduler
2. Virtual memory
3. Huge pages
4. NUMA
5. File System
6. Storage I/O
7. Networking
8. Hypervisor (Xen)



AWS re:Invent

1. CPU scheduler

- Tunables:
  - Scheduler class, priorities, migration latency, tasksets…
- Usage:
  - Some apps benefit from reducing migrations using taskset(1), numactl(8), cgroups, and tuning sched_migration_cost_ns
  - Some Java apps have benefited from SCHED_BATCH, to reduce context switching. E.g.:

```
# schedtool –B PID
```

2. Virtual memory

- Tunables:
  - Swappiness, overcommit, OOM behavior…
- Usage:
  - Swappiness is set to zero to disable swapping and favor ditching the file system page cache first to free memory. (This tunable doesn't make much difference, as swap devices are usually absent.)

```
vm.swappiness = 0                                  # from 60
```

AWS re:Invent

3. Huge pages

- Tunables:
  - Explicit huge page usage, transparent huge pages (THPs)
  - Using 2 or 4 Mbytes, instead of 4k, should reduce various CPU overheads and improve MMU page translation cache reach
- Usage:
  - THPs (enabled in later Ubuntu kernels) depending on the workload and CPUs, sometimes improve perf on HVM instances (~5% lower CPU), but sometimes hurt perf (~25% higher CPU during %usr, and more during %sys refrag)
  - We switched it back to madvise:

```
# echo madvise > /sys/kernel/mm/transparent_hugepage/enabled
```

4. NUMA

- Tunables:
  - NUMA balancing
- Usage:
  - On multi-NUMA systems (largest instances) and earlier kernels (around 3.13), NUMA page rebalance was too aggressive, and could consume 60% CPU alone.
  - We disable it. Will re-enable/tune later.

```
kernel.numa_balancing = 0
```

AWS re:Invent

5. File system

- Tunables:
  - Page cache flushing behavior, file system type and its own tunables (e.g., ZFS on Linux)
- Usage:
  - Page cache flushing is tuned to provide a more even behavior: Background flush earlier, aggressive flush later
  - Access timestamps disabled, and other options depending on the FS

```
vm.dirty_ratio = 80                        # from 40
vm.dirty_background_ratio = 5              # from 10
vm.dirty_expire_centisecs = 12000          # from 3000
mount -o defaults,noatime,discard,nobarrier …
```

6. Storage I/O

- Tunables:
  - Read ahead size, number of in-flight requests, I/O scheduler, volume stripe width…
- Usage:
  - Some workloads, e.g., Cassandra, can be sensitive to read ahead size
  - SSDs can perform better with the “noop” scheduler (if not default already)
  - Tuning md chunk size and stripe width to match workload

```
/sys/block/*/queue/rq_affinity    2
/sys/block/*/queue/scheduler      noop
/sys/block/*/queue/nr_requests    256
/sys/block/*/queue/read_ahead_kb  256
mdadm –chunk=64 ...
```

7. Networking

- Tunables:
  - TCP buffer sizes, TCP backlog, device backlog, TCP reuse…
- Usage:

```
net.core.somaxconn = 1024
net.core.netdev_max_backlog = 5000
net.core.rmem_max = 16777216
net.core.wmem_max = 16777216
net.ipv4.tcp_wmem = 4096 12582912 16777216
net.ipv4.tcp_rmem = 4096 12582912 16777216
net.ipv4.tcp_max_syn_backlog = 8096
net.ipv4.tcp_slow_start_after_idle = 0
net.ipv4.tcp_tw_reuse = 1
net.ipv4.ip_local_port_range = 10240 65535
net.ipv4.tcp_abort_on_overflow = 1    # maybe
```

8. Hypervisor (Xen)

- Tunables:
  - PV/HVM (baked into AMI)
  - Kernel clocksource. From slow to fast: hpet, xen, tsc
- Usage:
  - We've encountered a Xen clocksource regression in the past (Ubuntu Trusty). Fixed by tuning clocksource to TSC (although beware of clock drift).
  - Best case example (so far): CPU usage reduced by 30%, and average app latency reduced by 43%.

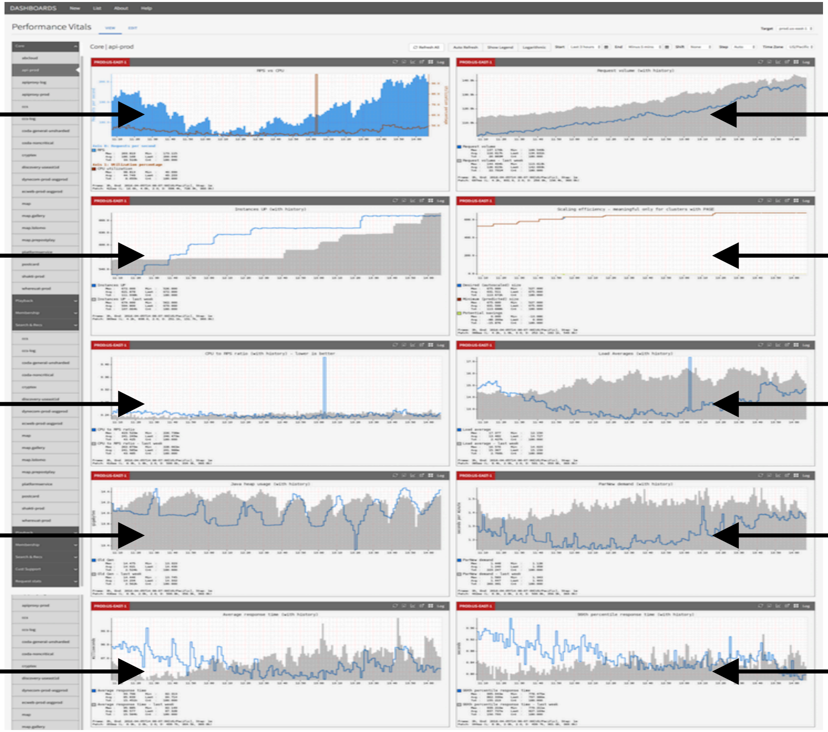
```
echo tsc > /sys/devices/system/clocksource/clocksource0/current_clocksource
```

4. Methodologies

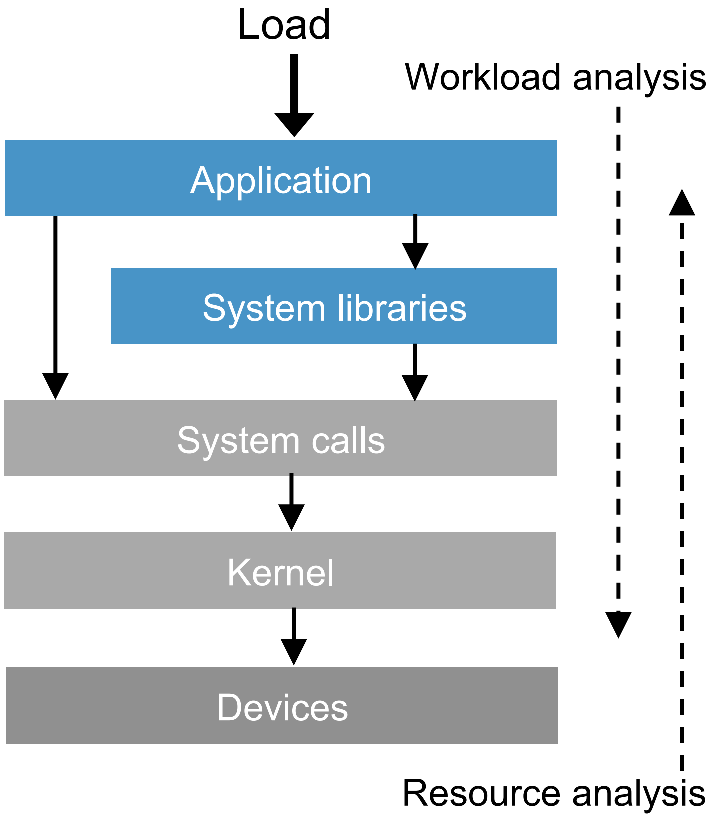
Techniques of performance analysis

Checklists: e.g., Netflix perf vitals dashboard

1. RPS, CPU
2. Volume
3. Instances
4. Scaling
5. CPU/RPS
6. Load avg
7. Java heap
8. ParNew
9. Latency
10. 99th tile

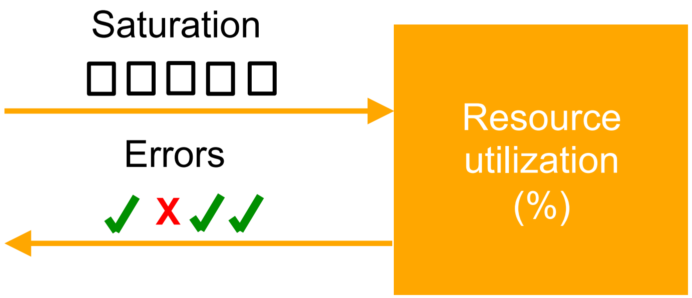


Analysis perspectives



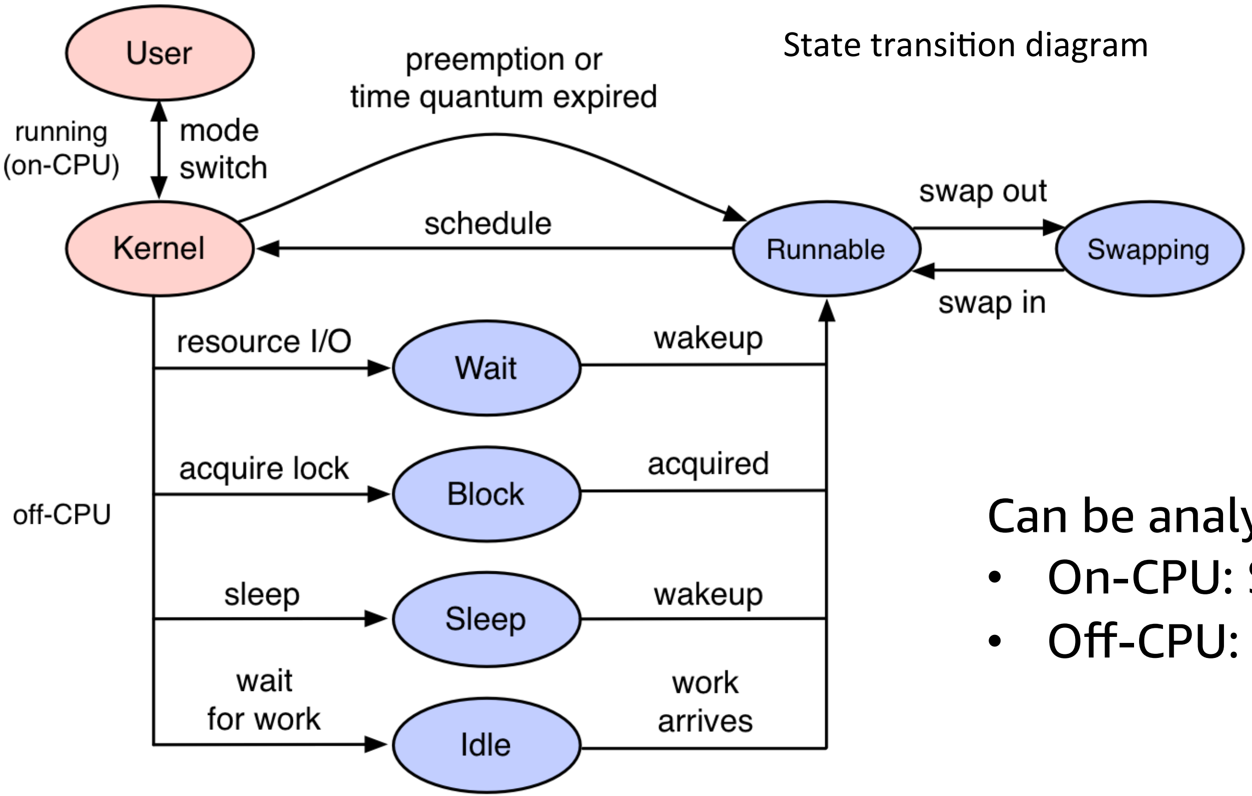
USE Method

- For every hardware and software resource, check:
  1. Utilization
  2. Saturation
  3. Errors



- Resource constraints show as saturation or high utilization
  - Resize or change instance type
  - Investigate tunables for the resource
- The USE Method poses questions to answer

On-CPU and off-CPU analysis



State transition diagram

Can be analyzed using:

- On-CPU: Sampling
- Off-CPU: Scheduler tracing

5. Observability

Finding, quantifying, and confirming tunables

Discovering system wins (5-25%'s) and application wins (2-10x's)

AWS re:Invent

Statistical tools

- vmstat, pidstat, sar, etc., used mostly normally

```
$ sar -n TCP,ETCP,DEV 1
Linux 3.2.55 (test-e4f1a80b)        08/18/2014       _x86_64_ (8 CPU)

09:10:43 PM  IFACE   rxpck/s   txpck/s    rxkB/s    txkB/s rxcmp/s txcmp/s  rxmcst/s
09:10:44 PM     lo     14.00     14.00      1.34      1.34    0.00    0.00      0.00
09:10:44 PM   eth0   4114.00   4186.00   4537.46 28513.24    0.00    0.00      0.00

09:10:43 PM  active/s passive/s    iseg/s    oseg/s
09:10:44 PM     21.00      4.00   4107.00  22511.00

09:10:43 PM  atmptf/s  estres/s retrans/s isegerr/s   orsts/s
09:10:44 PM      0.00      0.00     36.00      0.00      1.00
[…]
```

Host perf analysis in 60s

1. `uptime` --------------------------------▶ load averages
2. `dmesg | tail` -----------------------▶ kernel errors
3. `vmstat 1` ----------------------------▶ overall stats by time
4. `mpstat -P ALL 1` ---------------▶ CPU balance
5. `pidstat 1` --------------------------▶ process usage
6. `iostat -xz 1` ----------------------▶ disk I/O
7. `free -m` -----------------------------▶ memory usage
8. `sar -n DEV 1` ----------------------▶ network I/O
9. `sar -n TCP,ETCP 1` -------------▶ TCP stats
10. `top` ------------------------------------▶ check overview

http://techblog.netflix.com/2015/11/linux-performance-analysis-in-60s.html

AWS re:Invent

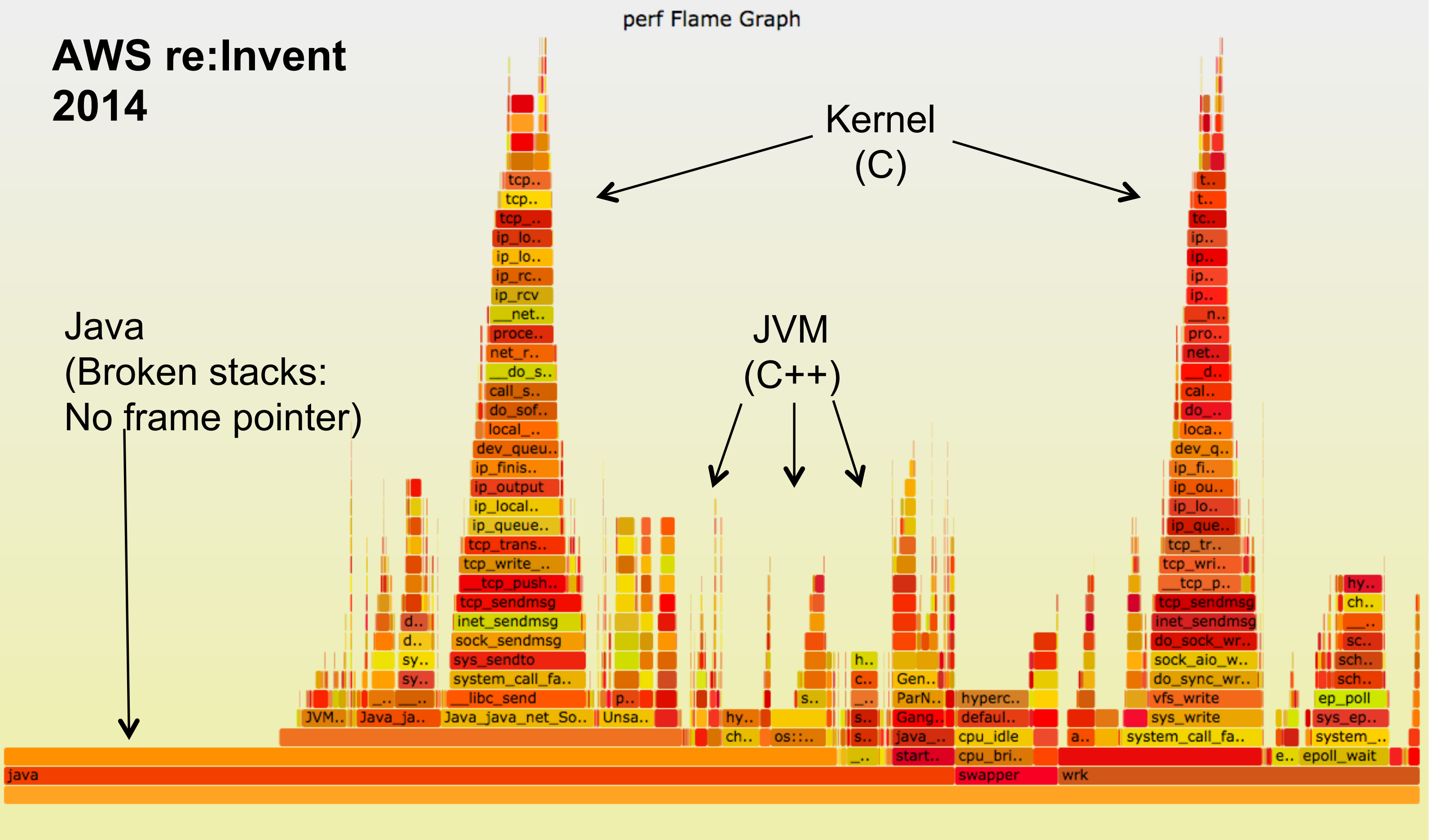
aws

System profilers

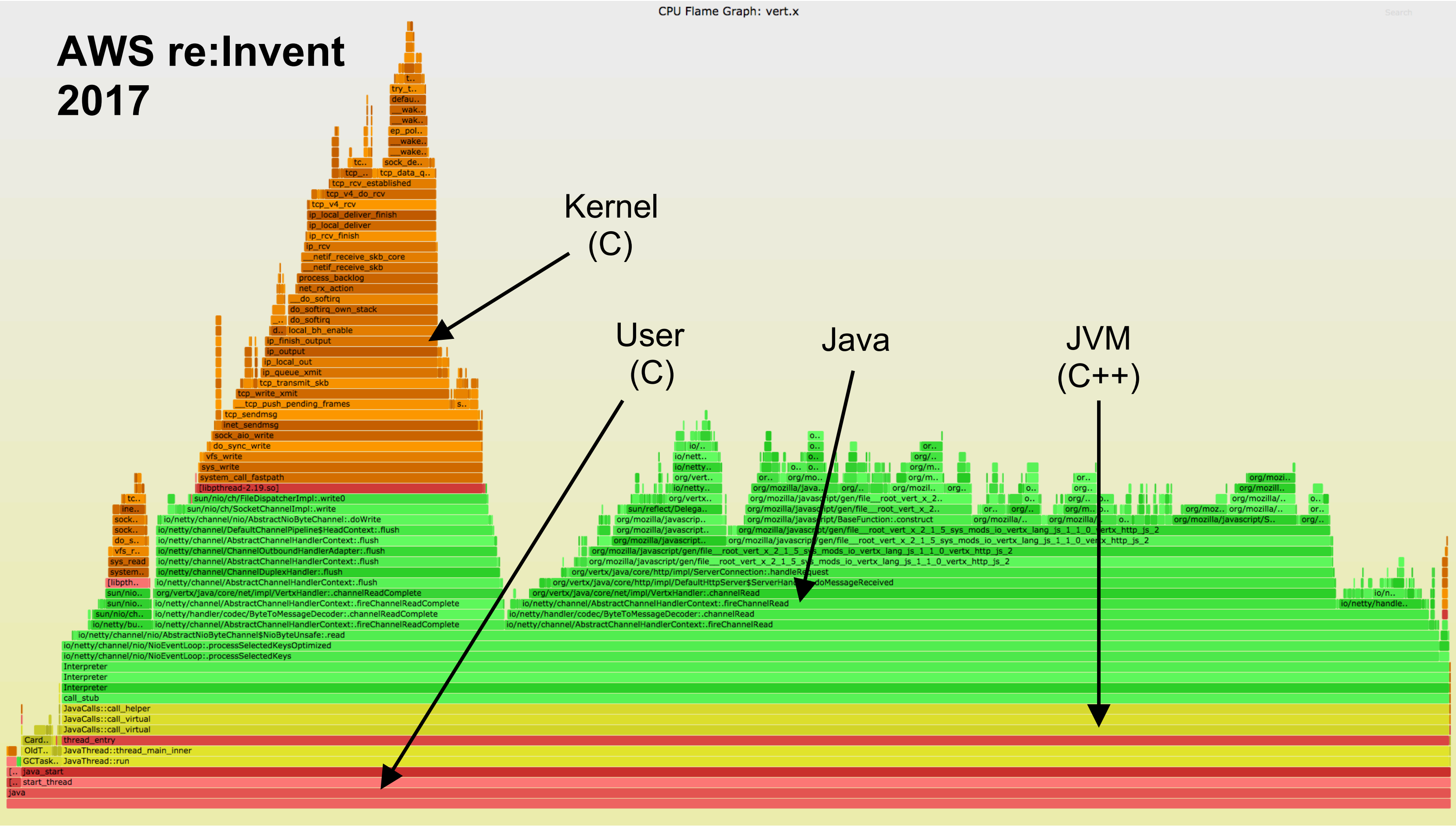
- perf
  - Standard Linux profiler. In the Linux source tree.
  - Interval sampling, CPU performance counter events.
  - User and kernel static and dynamic tracing.
- perf CPU flame graphs:

```
# git clone https://github.com/brendangregg/FlameGraph
# cd FlameGraph
# perf record -F 49 -ag -- sleep 30
# perf script | ./stackcollapse-perf.pl | ./flamegraph.pl > perf.svg
```

https://medium.com/netflix-techblog/java-in-flames-e763b3d32166


perf Flame Graph
AWS re:Invent
2014
Kernel
(C)
Java
(Broken stacks:
No frame pointer)
JVM
(C++)
tcp..
tcp_..
ip_lo..
ip_rc..
ip_rcv
__net..
proce..
net_r..
__do_s..
call_s..
do_sof..
local_..
dev_queu..
ip_finis..
ip_output
ip_local..
ip_queue..
tcp_trans..
tcp_write_..
__tcp_push..
tcp_sendmsg
inet_sendmsg
sock_sendmsg
sys_sendto
system_call_fa..
__libc_send
Java_java_net_So..
JVM..
Java_ja..
Unsa..
hy..
ch..
os::..
Gen..
ParN..
Gang..
java_..
start..
hyperc..
defaul..
cpu_idle
cpu_bri..
swapper
do_sock_wr..
sock_aio_w..
do_sync_wr..
vfs_write
sys_write
system_call_fa..
ep_poll
sys_ep..
epoll_wait
java
wrk

AWS re:Invent 2017



Tracing Tools: ftrace

- Part of the Linux kernel
  - First added in 2.6.27 (2008), and enhanced in later releases
  - Already available in all Netflix Linux instances
- Front-end tools aid usage: perf-tools collection
  - https://github.com/brendangregg/perf-tools
  - Unsupported hacks: see WARNINGs
  - Also see the trace-cmd front-end, as well as perf

ftrace tool: iosnoop

```
# /apps/perf-tools/bin/iosnoop -ts
Tracing block I/O. Ctrl-C to end.
STARTs         ENDs           COMM       PID   TYPE DEV     BLOCK      BYTES  LATms
5982800.302061 5982800.302679 supervise  1809  W    202,1   17039600   4096    0.62
5982800.302423 5982800.302842 supervise  1809  W    202,1   17039608   4096    0.42
5982800.304962 5982800.305446 supervise  1801  W    202,1   17039616   4096    0.48
5982800.305250 5982800.305676 supervise  1801  W    202,1   17039624   4096    0.43
[…]
```

```
# /apps/perf-tools/bin/iosnoop -h
USAGE: iosnoop [-hQst] [-d device] [-i iotype] [-p PID] [-n name] [duration]
                 -d device       # device string (eg, "202,1)
                 -i iotype       # match type (eg, '*R*' for all reads)
                 -n name         # process name to match on I/O issue
                 -p PID          # PID to match on I/O issue
                 -Q              # include queueing time in LATms
                 -s              # include start time of I/O (s)
                 -t              # include completion time of I/O (s)
[…]
```

Tracing tools: perf

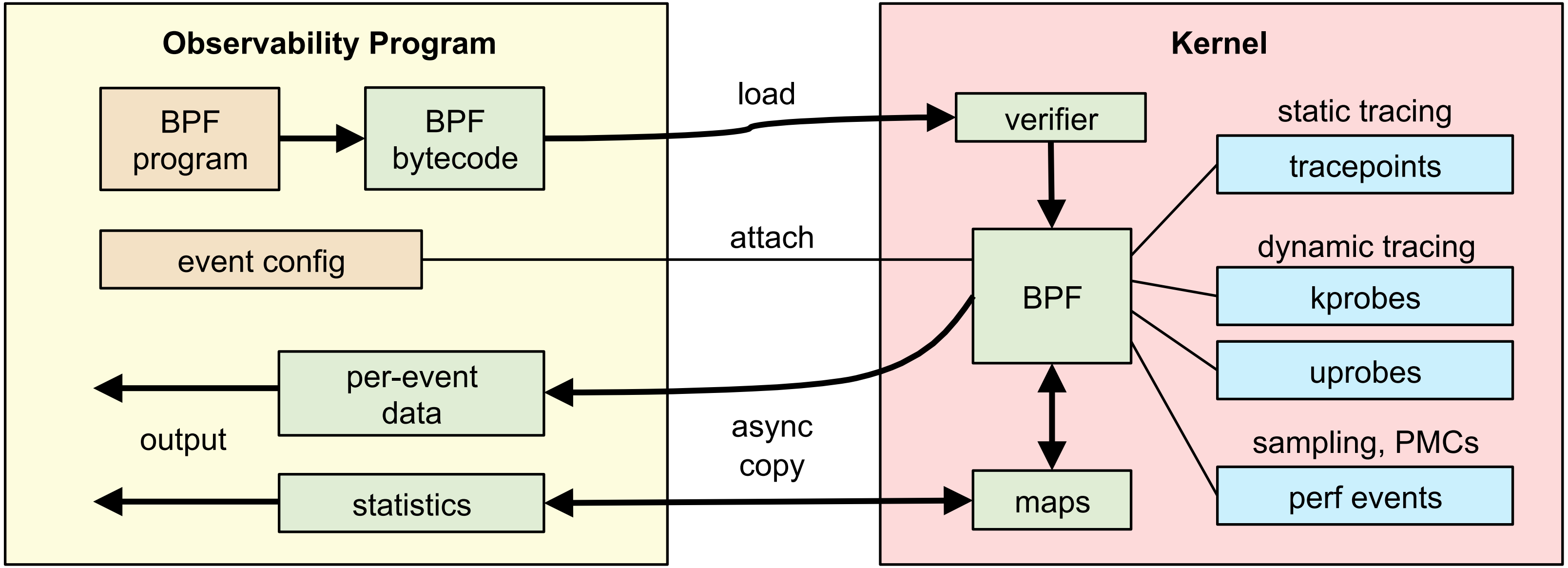
```
# perf record -e skb:consume_skb -ag -- sleep 10
# perf report
[...]
    74.42%  swapper  [kernel.kallsyms]  [k] consume_skb
            |
            --- consume_skb
                arp_process
                arp_rcv
                __netif_receive_skb_core
                __netif_receive_skb
                netif_receive_skb
                virtnet_poll
                net_rx_action
                __do_softirq
                irq_exit
                do_IRQ
                ret_from_intr
[…]
```

Summarizing stack traces for a tracepoint

perf can do many things, it is hard to pick just one example

Tracing tools: BPF

- Enhanced Berkeley Packet Filter (BPF, aka eBPF)
  - Safe, efficient, advanced, production tracing. Best on Linux 4.9+.



BPF: tcplife

```
# /usr/share/bcc/tools/tcplife
PID   COMM       LADDR           LPORT RADDR           RPORT TX_KB RX_KB MS
2509  java       100.82.34.63    8078  100.82.130.159  12410     0     0 5.44
2509  java       100.82.34.63    8078  100.82.78.215   55564     0     0 135.32
2509  java       100.82.34.63    60778 100.82.207.252  7001      0    13 15126.87
2509  java       100.82.34.63    38884 100.82.208.178  7001      0     0 15568.25
2509  java       127.0.0.1       4243  127.0.0.1       42166     0     0 0.61
12030 upload-mes 127.0.0.1       34020 127.0.0.1       8078     11     0 3.38
12030 upload-mes 127.0.0.1       21196 127.0.0.1       7101      0     0 12.61
3964  mesos-slav 127.0.0.1       7101  127.0.0.1       21196     0     0 12.64
12021 upload-sys 127.0.0.1       34022 127.0.0.1       8078    372     0 15.28
2509  java       127.0.0.1       8078  127.0.0.1       34022     0   372 15.31
2235  dockerd    100.82.34.63    13730 100.82.136.233  7002      0     4 18.50
2235  dockerd    100.82.34.63    34314 100.82.64.53    7002      0     8 56.73
[...]
```

Dynamic tracing of TCP set state only; does *not* trace send/receive
https://github.com/iovisor/bcc includes other TCP tools

AWS re:Invent

aws

Hardware counters

- Model Specific Registers (MSRs)
  - Basic details: Timestamp clock, temperature, power
  - Some are available in Amazon EC2
- Performance Monitoring Counters (PMCs)
  - Advanced details: Cycles, stall cycles, cache misses…
  - Availability depends on instance type: either none, some, or all
- Root cause CPU usage at the cycle level
  - E.g., higher CPU usage due to more memory stall cycles

MSRs

- Can be used to verify real CPU clock rate
  - Can vary with turboboost. Important to know for perf comparisons.
  - Tool from https://github.com/brendangregg/msr-cloud-tools:

```
ec2-guest# ./showboost
CPU MHz      : 2500
Turbo MHz    : 2900 (10 active)
Turbo Ratio  : 116% (10 active)
CPU 0 summary every 5 seconds...

TIME       C0_MCYC      C0_ACYC        UTIL  RATIO    MHz
06:11:35   6428553166   7457384521      51%   116%   2900
06:11:40   6349881107   7365764152      50%   115%   2899
06:11:45   6240610655   7239046277      49%   115%   2899
[...]
```

Real CPU MHz

AWS re:Invent

PMCs: Architectural

- Some instance types (e.g., m4.16xl) support the PMC architectural set:

Event Name	UMask	Event Select	Example Event Mask Mnemonic
UnHalted Core Cycles	00H	3CH	CPU_CLK_UNHALTED.THREAD_P
Instruction Retired	00H	C0H	INST_RETIRED.ANY_P
UnHalted Reference Cycles	01H	3CH	CPU_CLK_THREAD_UNHALTED.REF_XCLK
LLC Reference	4FH	2EH	LONGEST_LAT_CACHE.REFERENCE
LLC Misses	41H	2EH	LONGEST_LAT_CACHE.MISS
Branch Instruction Retired	00H	C4H	BR_INST_RETIRED.ALL_BRANCHES
Branch Misses Retired	00H	C5H	BR_MISP_RETIRED.ALL_BRANCHES

http://www.brendangregg.com/blog/2017-05-04/the-pmcs-of-ec2.html

PMCs: All

- All PMCs are available on this c5.18xl:

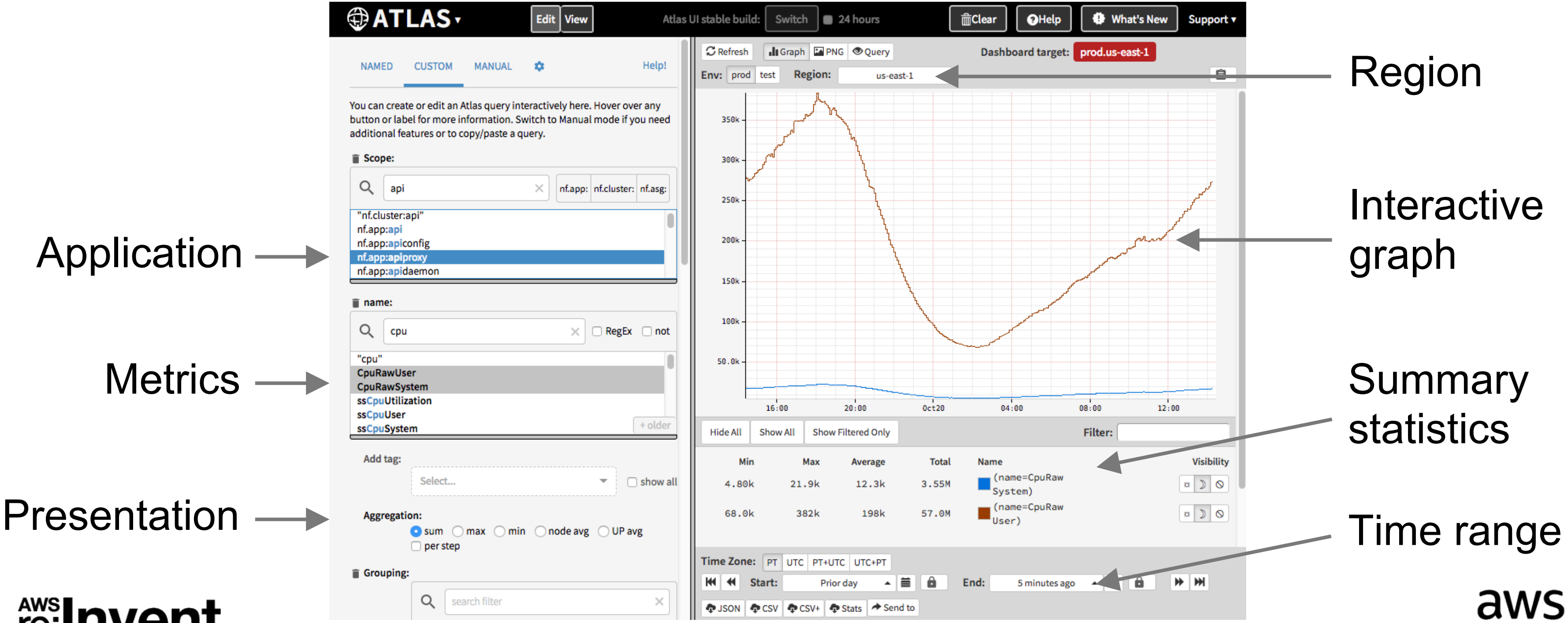
```
# perf stat -d -a -- sleep 5

 Performance counter stats for 'system wide':

     360195.435103      cpu-clock (msec)          #   71.454 CPUs utilized
            38,733      context-switches          #    0.108 K/sec
               504      cpu-migrations            #    0.001 K/sec
           861,393      page-faults               #    0.002 M/sec
     2,275,234,239      cycles                    #    0.006 GHz
   191,859,050,716      instructions              #   84.32  insn per cycle
    38,989,119,249      branches                  #  108.244 M/sec
       152,913,791      branch-misses             #    0.39% of all branches
    40,262,604,776      L1-dcache-loads           #  111.780 M/sec
       283,924,939      L1-dcache-load-misses     #    0.71% of all L1-dcache hits
[...]
```

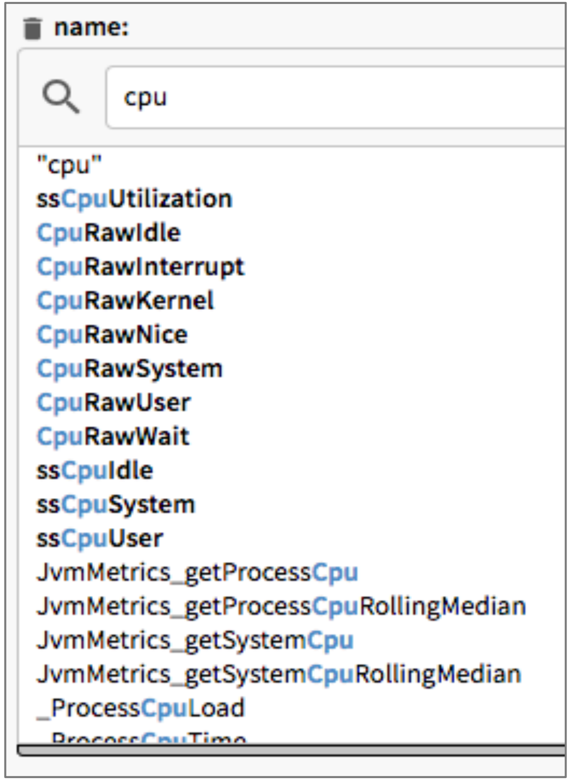
Netflix Atlas

- Cloud-wide and instance monitoring:



Netflix Atlas

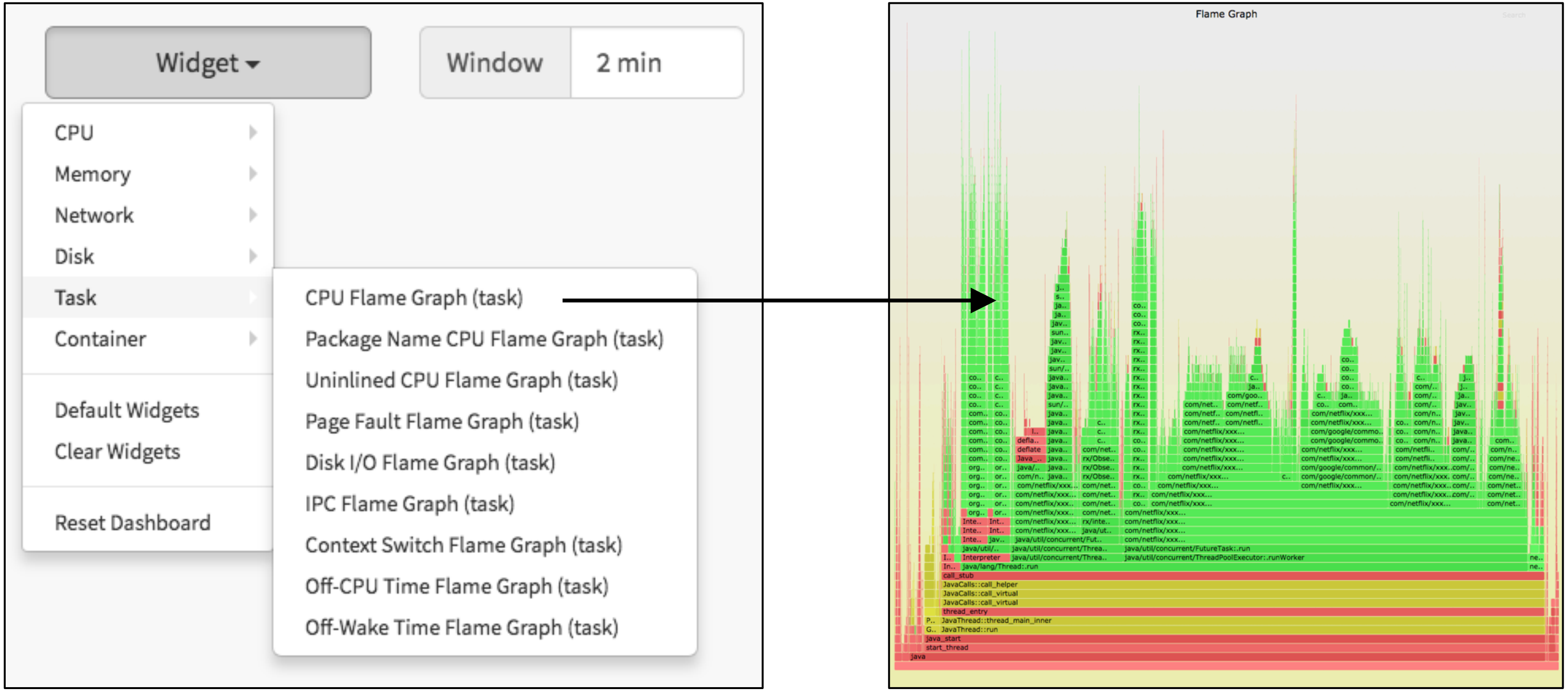
- All metrics in one system
- System metrics:
  - CPU usage, disk I/O, memory…
- Application metrics:
  - Requests completed, latency percentiles, errors…
- Filters/breakdowns by region, application, ASG, metric, instance



Netflix Vector

- Real-time per-second instance metrics:

Vector on-demand flame graphs



AWS re:Invent

aws

Vector

- Given an instance, analyze low-level performance
- On-demand flame graphs
  - CPU, off-CPU, context switch, IPC, page fault, disk I/O
  - These use perf or BPF
- Quick
  - GUI-driven root cause analysis
- Scalable
  - Other teams can use it easily

Summary

1. Instance selection
2. Amazon EC2 features
3. Kernel tuning
4. Methodologies
5. Observability

References & links

- Amazon EC2:
  - http://aws.amazon.com/ec2/instance-types/
  - http://docs.aws.amazon.com/AWSEC2/latest/UserGuide/instance-types.html
  - http://docs.aws.amazon.com/AWSEC2/latest/UserGuide/enhanced-networking.html
  - https://www.slideshare.net/AmazonWebServices/cmp402-amazon-ec2-instances-deep-dive
  - http://www.brendangregg.com/blog/2017-05-04/the-pmcs-of-ec2.html
- Netflix on EC2:
  - http://www.slideshare.net/cpwatson/cpn302-yourlinuxamioptimizationandperformance
  - http://www.brendangregg.com/blog/2014-09-27/from-clouds-to-roots.html
  - http://techblog.cloudperf.net/2016/05/2-million-packets-per-second-on-public.html
  - http://techblog.cloudperf.net/2017/04/3-million-storage-iops-on-aws-cloud.html
- Performance Analysis:
  - http://www.brendangregg.com/linuxperf.html
  - http://techblog.netflix.com/2015/11/linux-performance-analysis-in-60s.html
  - https://github.com/iovisor/bcc https://github.com/brendangregg/perf-tools
  - https://www.slideshare.net/brendangregg/velocity-2015-linux-perf-tools
  - http://www.brendangregg.com/USEmethod/use-linux.html
  - https://medium.com/netflix-techblog/java-in-flames-e763b3d32166
  - http://www.brendangregg.com/FlameGraphs/cpuflamegraphs.html#Java
  - https://github.com/brendangregg/FlameGraph

Netflix talks @ re:Invent

Monday
- 10:45am ARC208:Walking the tightrope: Balancing Innovation, Reliability, Security, and Efficiency (Venetian)
- 12:15pm SID206: Best Practices for Managing Security on AWS (MGM)

Tuesday
- 10:45am ARC209: A Day in the Life of a Netflix Engineer (Venetian)
- 11:30am CMP325: How Netflix Tunes EC2 Instances for Performance (Venetian)

Wednesday
- 11:30am MCL317: Orchestrating ML Training for Netflix Recommendations (Venetian)
- 12:15pm NET303: A day in the life of a Cloud Network Engineer at Netflix (Venetian)
- 1:00pm ARC312: Why Regional Reservations are a Game Changer for Netflix (Venetian)
- 1:00pm SID304: SecOps 2021 Today: Using AWS Services to Deliver SecOps (MGM)
- 1:45pm DEV334: Performing Chaos at Netflix Scale (Venetian)
- 4:45pm SID316: Using Access Advisor to Strike the Balance Between Security and Usability (MGM)

Thursday
- 12:15pm CMP311: Auto Scaling Made Easy: How Target Tracking Scaling Policies Hit the Bullseye (Palazzo)
- 12:15pm DAT308: Codex: Conditional Modules Strike Back (Venetian)
- 12:55pm CMP309: How Netflix Encodes at Scale (Venetian)
- 5:00pm ABD401: How Netflix Monitors Applications Real Time with Kinesis (Aria)

Friday
- 8:30am ABD319: Tooling Up For Efficiency: DIY Solutions @ Netflix (Aria)
- 10:00am ABD401: Netflix Keystone SPaaS - Real-time Stream Processing as a Service (Aria)

CMP325

Thank you!

Brendan Gregg, Netflix Performance and Operating Systems Team

@brendangregg