Performance Analysis:
new tools and concepts from the cloud

Brendan Gregg

Lead Performance Engineer, Joyent

brendan.gregg@joyent.com

whoami

- **I do performance analysis**
  - I also write performance tools out of necessity
- **Was Brendan @ Sun Microsystems, Oracle, now Joyent**

Joyent

- **Cloud computing provider**
- **Cloud computing software**
- **SmartOS**
  - host OS, and guest via OS virtualization
- **Linux, Windows**
  - guest via KVM

Agenda

Joyent

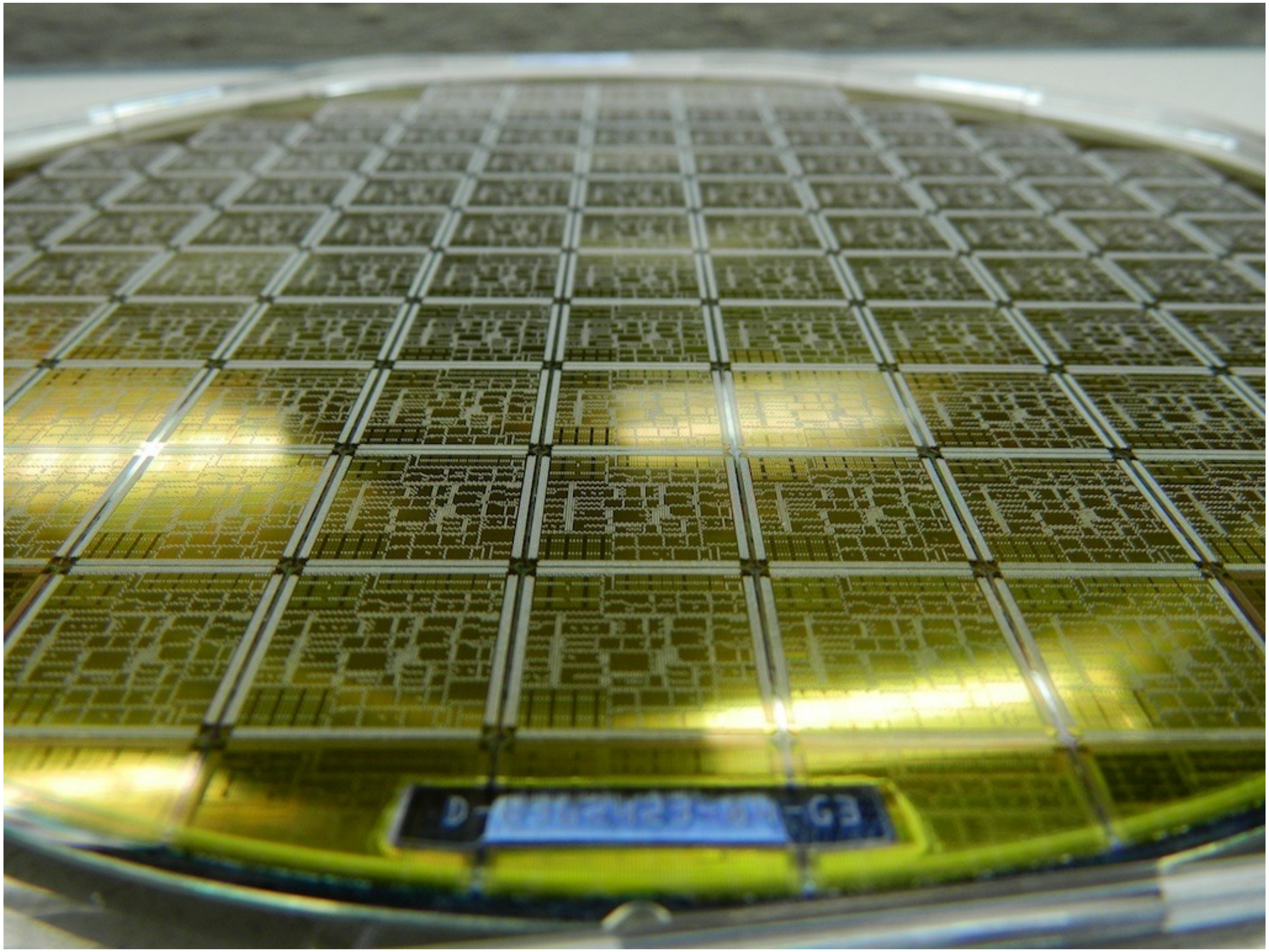
- **Data**
  - Example problems & solutions
  - How cloud environments complicate performance
- **Theory**
  - Performance analysis
  - Summarize new tools & concepts

- This talk uses SmartOS and DTrace to illustrate concepts that are applicable to most OSes.

Data

- **Example problems:**
  - CPU
  - Memory
  - Disk
  - Network
- **Some have neat solutions, some messy, some none**
  - This is real world
- **Some I’ve covered before, some I haven’t**

CPU



CPU utilization: problem

- **Would like to identify:**
  - single or multiple CPUs at 100% utilization
  - average, minimum and maximum CPU utilization
  - CPU utilization balance (tight or loose distribution)
  - time-based characteristics
    changing/bursting? burst interval, burst length
- **For small to large environments**
  - entire datacenters or clouds

CPU utilization

Joyent

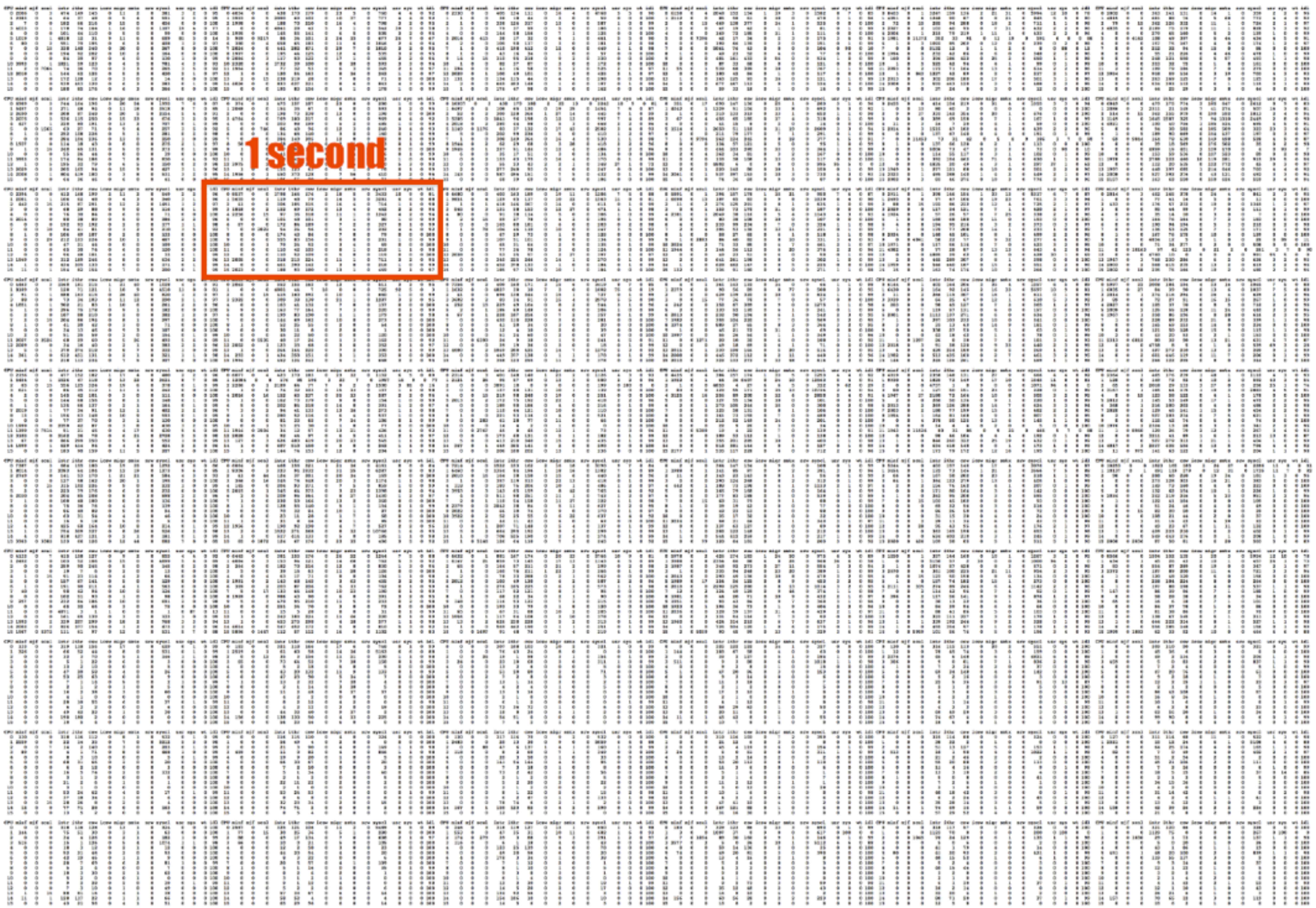
- mpstat(1) has the data. 1 second, 1 server (16 CPUs):

CPU	minf	mjf	xcal	intr	ithr	csw	icsw	migr	smtx	srw	syscl	usr	sys	wt	idl
0	1250	0	1	357	144	160	0	10	1	0	1267	3	2	0	95
1	0	0	0	5150	57	2	10	0	0	0	1000	94	6	0	0
2	1	0	0	1074	57	428	0	8	3	0	571	0	2	0	98
3	2670	0	6	361	100	225	0	21	11	0	954	2	3	0	95
4	0	0	15	123	50	158	0	8	0	0	134	0	0	0	100
5	1	0	0	157	74	182	0	15	1	0	273	0	0	0	100
6	2111	0	335	274	89	246	0	6	4	0	1142	2	2	0	96
7	0	0	2	114	42	96	0	7	4	0	92	0	1	0	99
8	396	0	2	117	56	141	0	9	5	0	876	1	2	0	97
9	0	0	0	30	13	22	0	6	1	0	36	0	0	0	100
10	0	0	0	84	39	94	0	6	0	0	66	0	0	0	100
11	0	0	0	88	41	86	0	5	0	0	103	0	0	0	100
12	0	0	0	223	111	227	0	3	0	0	179	1	0	0	99
13	1	0	1	339	192	244	0	9	2	0	328	0	1	0	99
14	1	0	0	455	354	97	0	1	0	0	178	1	0	0	99
15	0	0	5959	101	56	76	0	5	0	0	196	3	4	0	93

CPU utilization

Joyent

- **Scaling to 60 seconds, 1 server:**



CPU utilization

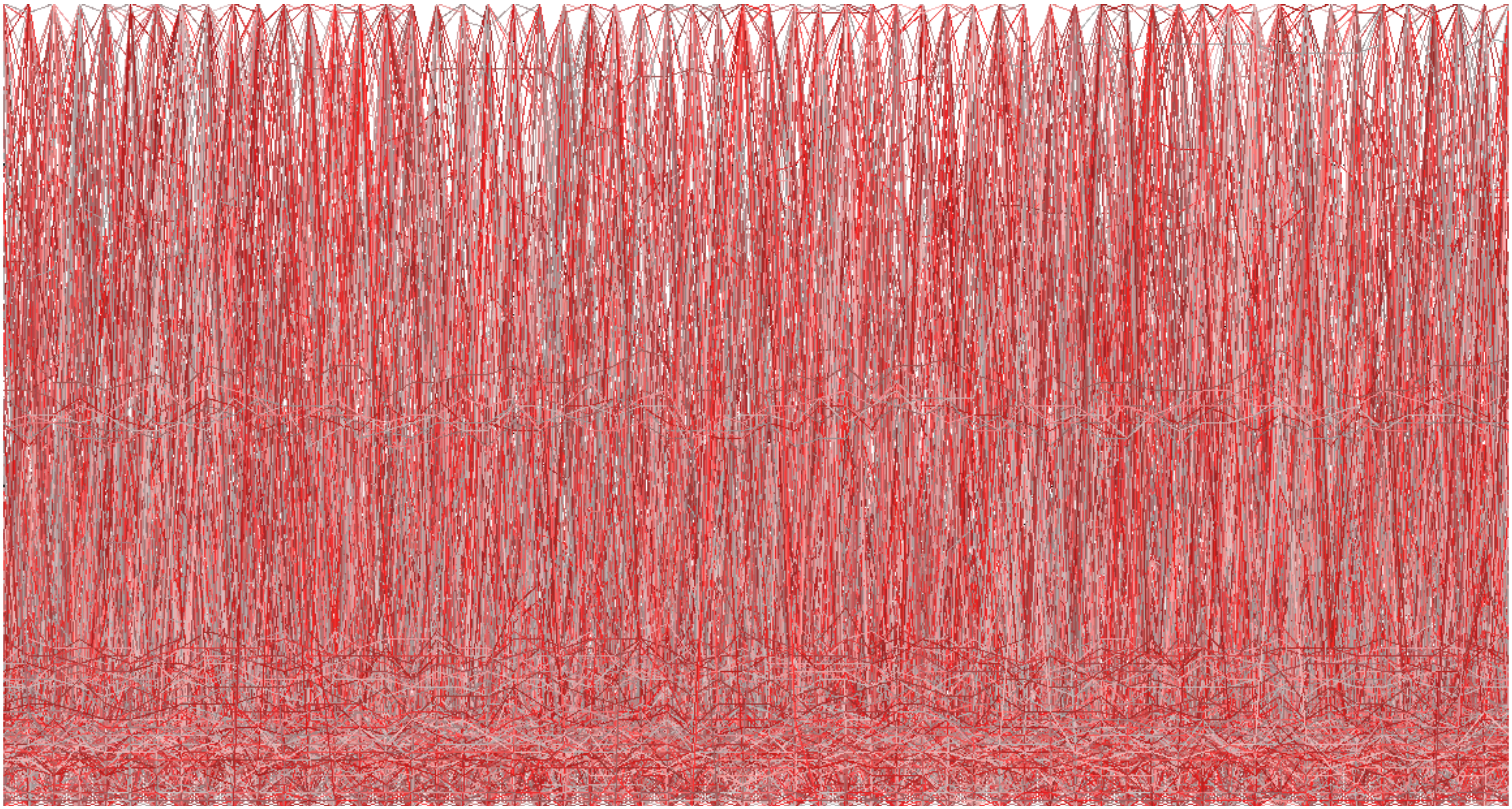
- **Scaling to entire datacenter, 60 secs, 5312 CPUs:**

CPU utilization

- **Line graphs can solve some problems:**
  - x-axis: time, 60 seconds
  - y-axis: utilization

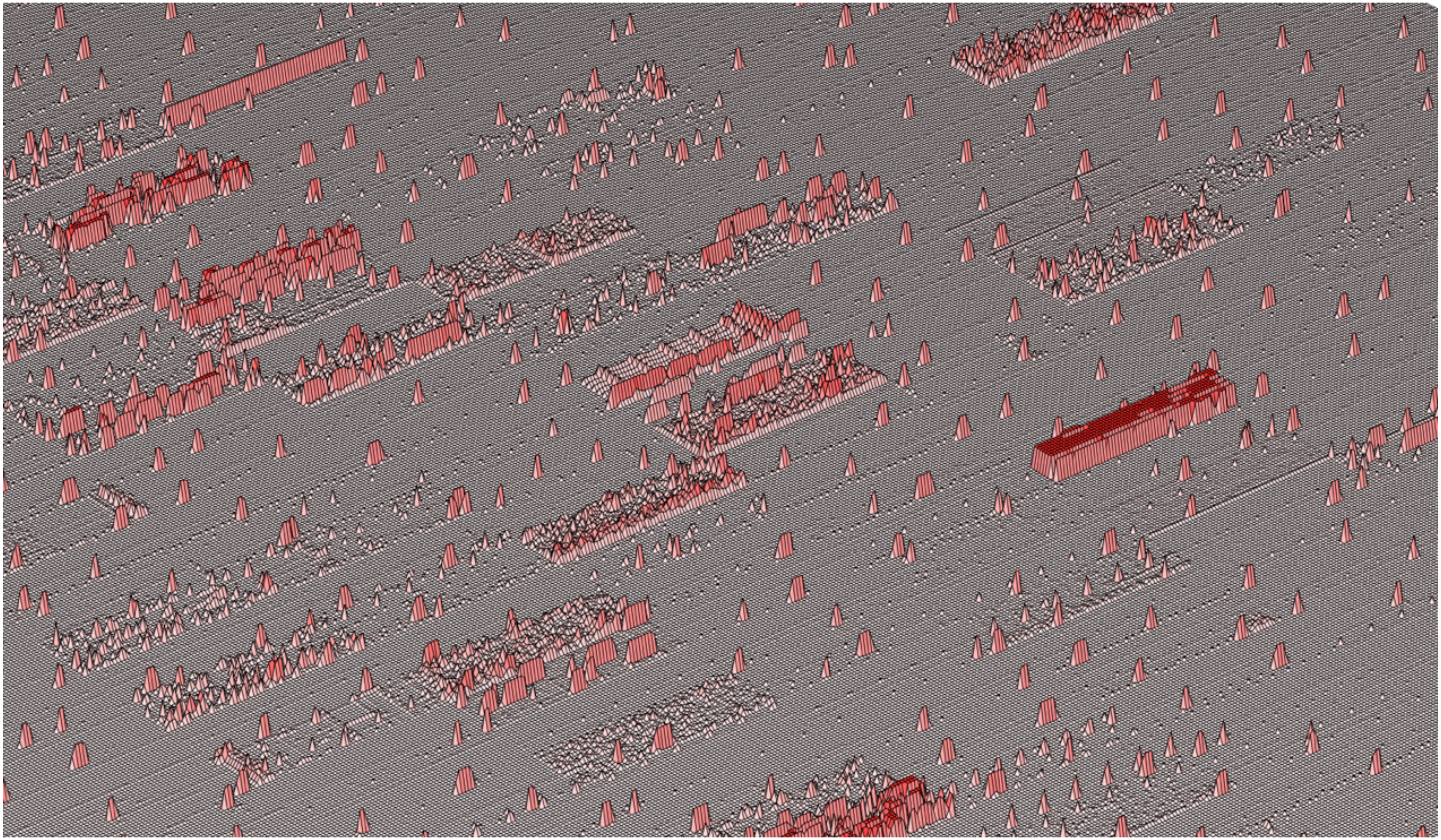
CPU utilization

- **... but don’t scale well to individual devices**
    - 5312 CPUs, each as a line:



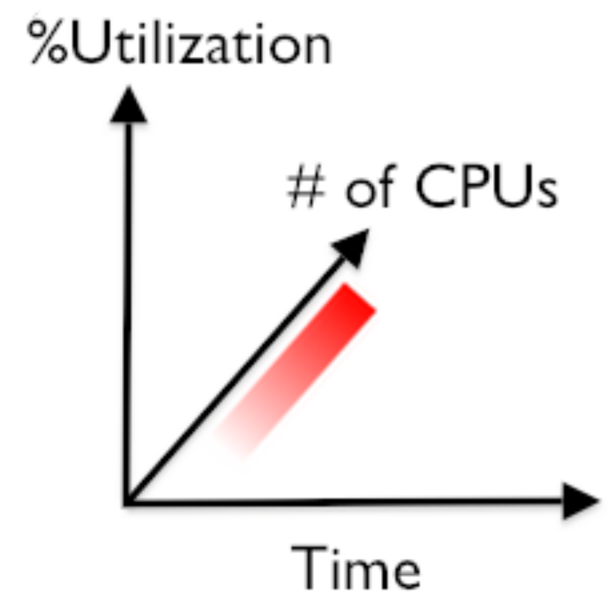
CPU utilization

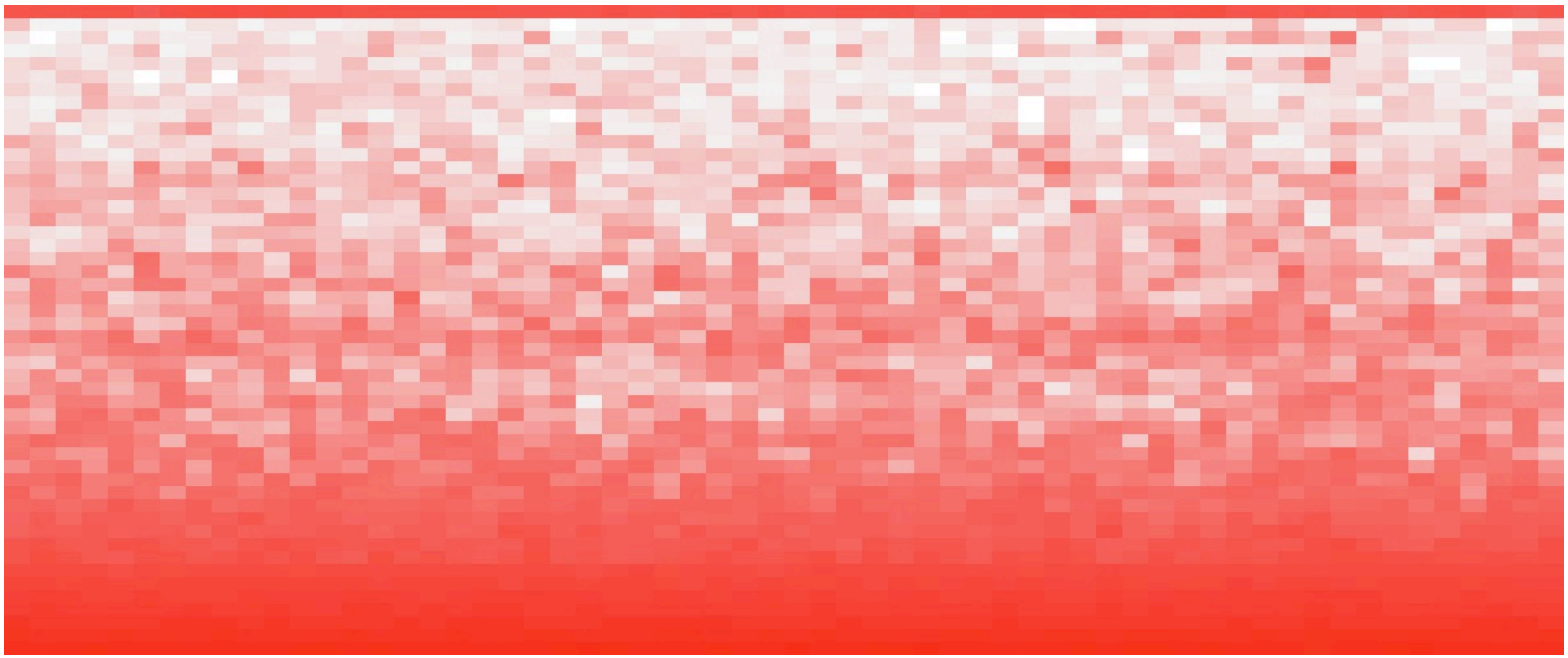
- Pretty, but scale limited as well:



CPU utilization

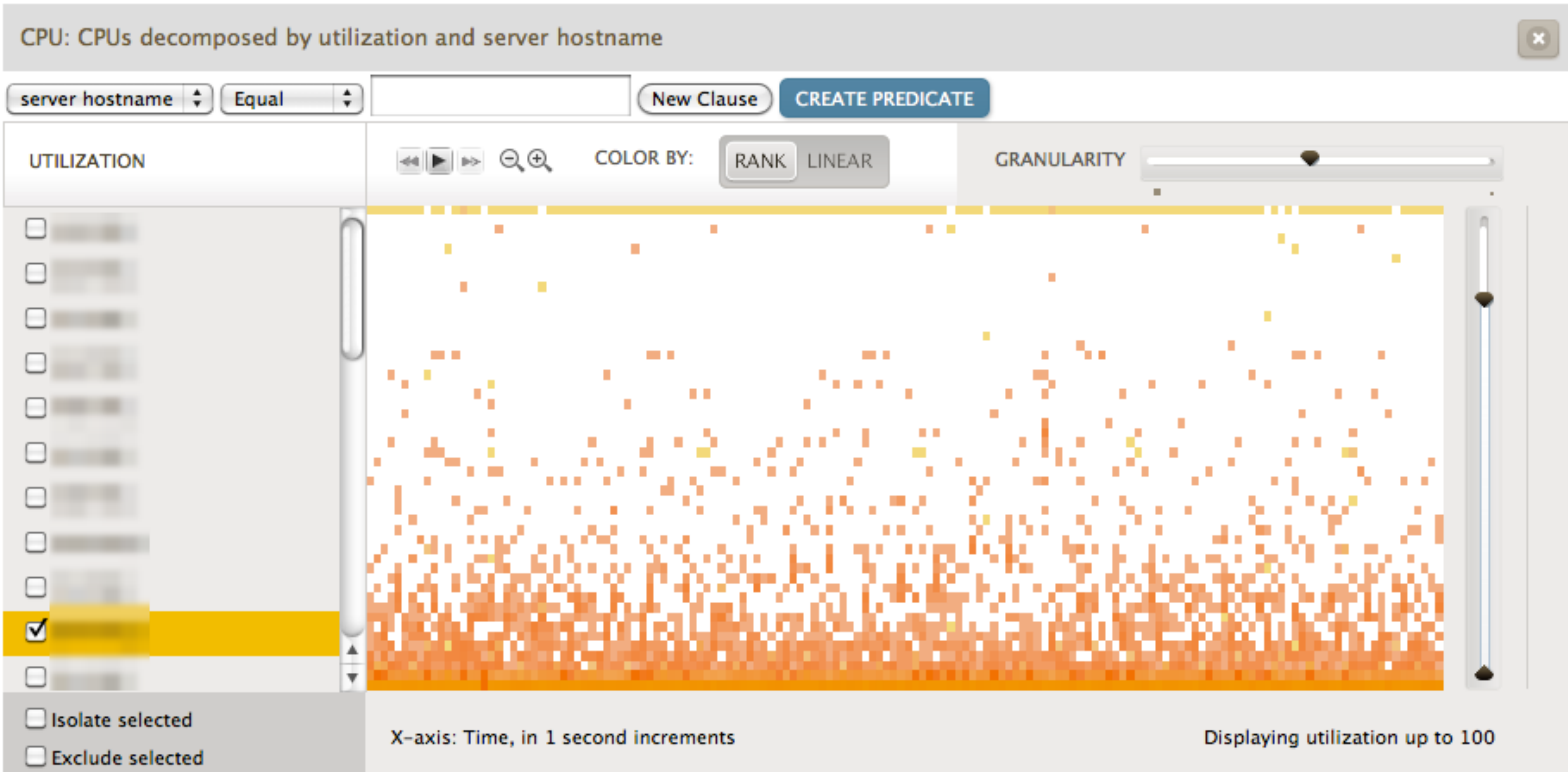
- **Utilization as a heat map:**
  - x-axis: time, y-axis: utilization
  - z-axis (color): number of CPUs





CPU utilization

- **Available in Cloud Analytics (Joyent)**
  - Clicking highlights and shows details; eg, hostname:



CPU utilization

- **Utilization heat map also suitable and used for:**
  - disks
  - network interfaces
- **Utilization as a metric can be a bit misleading**
  - really a percent busy over a time interval
  - devices may accept more work at 100% busy
  - may not directly relate to performance impact

CPU utilization: summary

- Data readily available
- Using a new visualization

CPU usage

- **Given a CPU is hot, what is it doing?**
  - Beyond just vmstat's usr/sys ratio
- **Profiling (sampling at an interval) the program counter or stack back trace**
  - user-land stack for %usr
  - kernel stack for %sys
- **Many tools can do this to some degree**
  - Developer Studios/DTrace/oprofile/...

CPU usage: profiling

- **Frequency count on-CPU user-land stack traces:**

```
# dtrace -x ustackframes=100 -n 'profile-997 /execname == "mysqld"/ {
    @[ustack()] = count(); } tick-60s { exit(0); }'
dtrace: description 'profile-997 ' matched 2 probes
CPU     ID                    FUNCTION:NAME
  1  75195                        :tick-60s
[...]
              libc.so.1`__priocntlset+0xa
              libc.so.1`getparam+0x83
              libc.so.1`pthread_getschedparam+0x3c
              libc.so.1`pthread_setschedprio+0x1f
              mysqld`_Z16dispatch_command19enum_server_commandP3THDPcj+0x9ab
              mysqld`_Z10do_commandP3THD+0x198
              mysqld`handle_one_connection+0x1a6
              libc.so.1`_thrp_setup+0x8d
              libc.so.1`_lwp_start
             1272

              mysqld`_Z13add_to_statusP17system_status_varS0_+0x47
              mysqld`_Z22calc_sum_of_all_statusP17system_status_var+0x67
              mysqld`_Z16dispatch_command19enum_server_commandP3THDPcj+0x1222
              mysqld`_Z10do_commandP3THD+0x198
              mysqld`handle_one_connection+0x1a6
              libc.so.1`_thrp_setup+0x8d
              libc.so.1`_lwp_start
             1643
```

CPU usage: profiling

Joyent

- **Frequency count on-CPU user-land stack traces:**

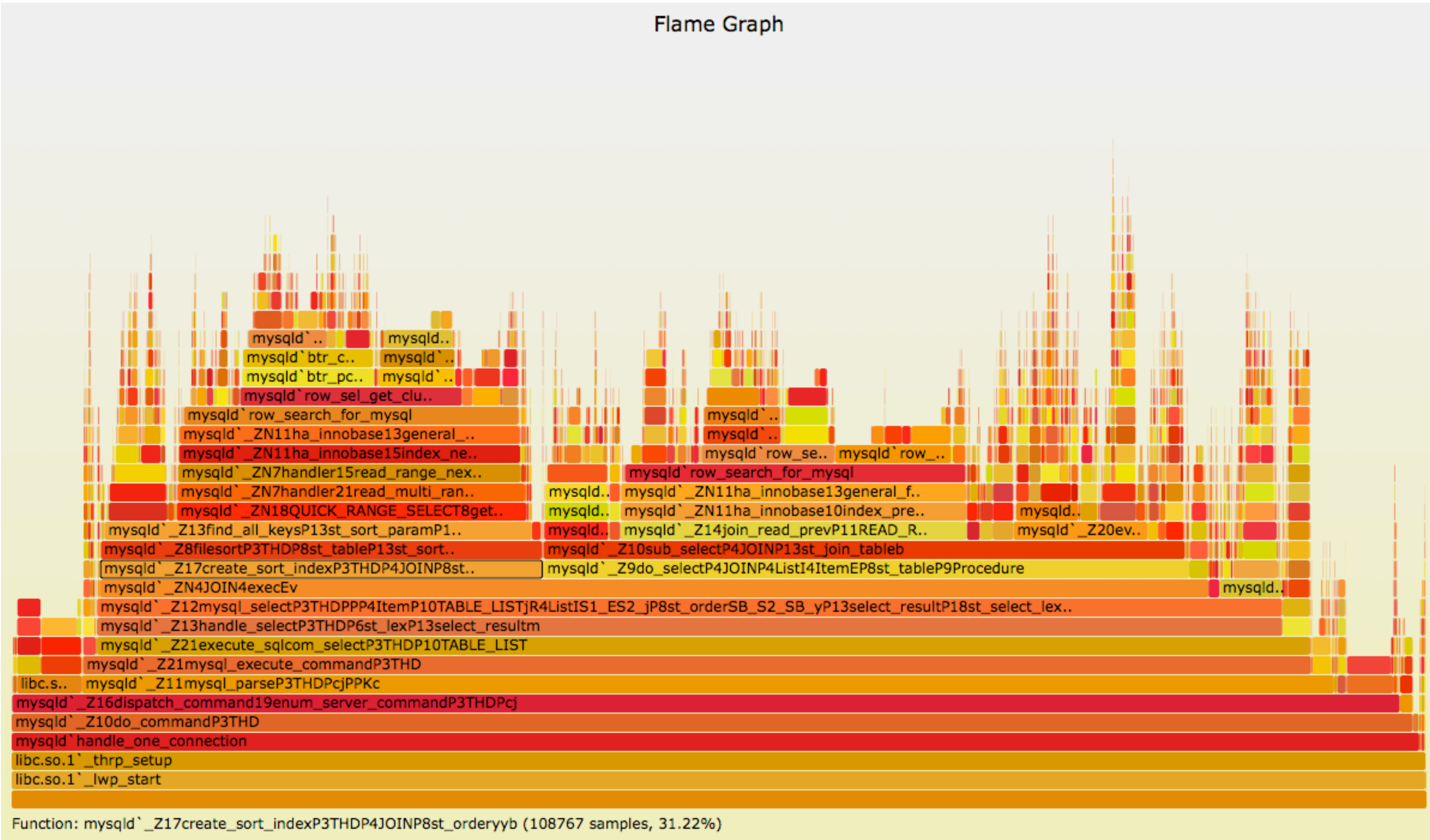
```
# dtrace -x ustackframes=100 -n 'profile-997 /execname == "mysqld"/ {
    @[ustack()] = count(); } tick-60s { exit(0); }'
dtrace: description 'profile-997 ' matched 2 probes
CPU     ID                    FUNCTION:NAME
  1  75195                        :tick-60s
[...]
              libc.so.1`__priocntlset+0xa
              libc.so.1`getparam+0x83
              libc.so.1`pthread_getschedparam+0x3c
              libc.so.1`pthread_setschedprio+0x1f
              mysqld`_Z16dispatch_command19enum_server_commandP3THDPcj+0x9ab
              mysqld`_Z10do_commandP3THD+0x198
              mysqld`handle_one_connection+0x1a6
              libc.so.1`_thrp_setup+0x8d
              libc.so.1`_lwp_start
             1272

              mysqld`_Z13add_to_statusP17system_status_varS0_+0x47
              mysqld`_Z22calc_sum_of_all_statusP17system_status_var+0x67
              mysqld`_Z16dispatch_command19enum_server_commandP3THDPcj+0x1222
              mysqld`_Z10do_commandP3THD+0x198
              mysqld`handle_one_connection+0x1a6
              libc.so.1`_thrp_setup+0x8d
              libc.so.1`_lwp_start
             1643
```

Over 500,000 lines truncated

CPU usage: profiling data

CPU usage: visualization

- Visualized as a “Flame Graph”:



CPU usage: Flame Graphs

- Just some Perl that turns DTrace output into an interactive SVG: mouse-over elements for details
- It's on github
  - http://github.com/brendangregg/FlameGraph
- Works on kernel stacks, and both user+kernel
- Shouldn't be hard to have it process oprofile, etc.

CPU usage: on the Cloud

- **Flame Graphs were born out of necessity on Cloud environments:**
  - Perf issues need quick resolution (you just got hackernews'd)
  - Everyone is running different versions of everything (don't assume you've seen the last of old CPU-hot code-path issues that have been fixed)

CPU usage: summary

- **Data can be available**
  - For cloud computing: easy for operators to fetch on OS virtualized environments; otherwise agent driven, and possibly other difficulties (access to CPU instrumentation counter-based interrupts)
- **Using a new visualization**

CPU latency

Joyent

- **CPU dispatcher queue latency**
  - thread is ready-to-run, and waiting its turn
- **Observable in coarse ways:**
  - vmstat's r
  - high load averages
- **Less course, with microstate accounting**
  - prstat -mL's LAT
- **How much is it affecting application performance?**

CPU latency: zonedispqlat.d

- **Using DTrace to trace kernel scheduler events:**

```
#./zonedisplat.d
Tracing...
Note: outliers (> 1 secs) may be artifacts due to the use of scalar globals
(sorry).

CPU disp queue latency by zone (ns):

  dbprod-045
           value  ------------- Distribution ------------- count
             512 |                                         0
            1024 |@@@@@@@@@@@@@@@@@@@@@@@@@@@@             10210
            2048 |@@@@@@@@@@                               3829
            4096 |@                                        514
            8192 |                                         94
           16384 |                                         0
           32768 |                                         0
           65536 |                                         0
          131072 |                                         0
          262144 |                                         0
          524288 |                                         0
         1048576 |                                         1
         2097152 |                                         0
         4194304 |                                         0
         8388608 |                                         1
        16777216 |                                         0
[...]
```

CPU latency: zonedispqlat.d

- **CPU dispatcher queue latency by zonename (zonedispqlat.d), *work in progress*:**

```
#!/usr/sbin/dtrace -s

#pragma D option quiet

dtrace:::BEGIN
{
        printf("Tracing...\n");
        printf("Note: outliers (> 1 secs) may be artifacts due to the ");
        printf("use of scalar globals (sorry).\n\n");
}

sched:::enqueue
{
        /* scalar global (I don't think this can be thread local) */
        start[args[0]->pr_lwpid, args[1]->pr_pid] = timestamp;
}

sched:::dequeue
/this->start = start[args[0]->pr_lwpid, args[1]->pr_pid]/
{
        this->time = timestamp - this->start;
        /* workaround since zonename isn't a member of args[1]... */
        this->zone = ((proc_t *)args[1]->pr_addr)->p_zone->zone_name;
        @[stringof(this->zone)] = quantize(this->time);
        start[args[0]->pr_lwpid, args[1]->pr_pid] = 0;
}

tick-1sec
{
        printf("CPU disp queue latency by zone (ns):\n");
        printa(@);
        trunc(@);
}
```

Save timestamp on enqueue; calculate delta on dequeue

CPU latency: zonedispqlat.d

Joyent

- **Instead of zonename, this could be process name, ...**
- **Tracing scheduler enqueue/dequeue events and saving timestamps costs CPU overhead**
  - they are frequent
- **I'd prefer to only trace dequeue, and reuse the existing microstate accounting timestamps**
  - but one problem is a clash between unscaled and scaled timestamps

CPU latency: on the Cloud

Joyent

- **With virtualization, you can have:**
  high CPU latency with idle CPUs
  due to an instance consuming their quota
- **OS virtualization**
  - not visible in vmstat r
  - is visible as part of prstat -mL's LAT
  - more kstats recently added to SmartOS
    including nsec_waitrq (total run queue wait by zone)
- **Hardware virtualization**
  - vmstat st (stolen)

CPU latency: caps

- **CPU cap latency from the host (zonecapslat.d):**

```
#!/usr/sbin/dtrace -s

#pragma D option quiet

sched:::cpucaps-sleep
{
	start[args[0]->pr_lwpid, args[1]->pr_pid] = timestamp;
}

sched:::cpucaps-wakeup
/this->start = start[args[0]->pr_lwpid, args[1]->pr_pid]/
{
	this->time = timestamp - this->start;
	/* workaround since zonename isn't a member of args[1]... */
	this->zone = ((proc_t *)args[1]->pr_addr)->p_zone->zone_name;
	@[stringof(this->zone)] = quantize(this->time);
	start[args[0]->pr_lwpid, args[1]->pr_pid] = 0;
}

tick-1sec
{
	printf("CPU caps latency by zone (ns):\n");
	printa(@);
	trunc(@);
}
```

CPU latency: summary

- **Partial data available**
- **New tools/metrics created**
  - although current DTrace solutions have overhead; we should be able to improve that
  - although, new kstats may be sufficient

Memory

Memory: problem

Joyent

- **Riak database has endless memory growth.**
  - expected 9GB, after two days:

```
$ prstat -c 1
Please wait...
   PID USERNAME  SIZE   RSS STATE  PRI NICE      TIME  CPU PROCESS/NLWP
 21722 103        43G   40G cpu0    59    0  72:23:41 2.6% beam.smp/594
 15770 root     7760K  540K sleep   57    0  23:28:57 0.9% zoneadmd/5
    95 root        0K    0K sleep   99  -20   7:37:47 0.2% zpool-zones/166
 12827 root      128M   73M sleep  100    -   0:49:36 0.1% node/5
 10319 bgregg     10M 6788K sleep   59    0   0:00:00 0.0% sshd/1
 10402 root       22M  288K sleep   59    0   0:18:45 0.0% dtrace/1
[...]
```

- **Eventually hits paging and terrible performance**
  - needing a restart
- **Is this a *memory leak*? Or application growth?**

Memory: scope

- **Identify the subsystem and team responsible**

Subsystem	Team
Application	Voxer
Riak	Basho
Erlang	Ericsson
SmartOS	Joyent

Memory: heap profiling

- **What is in the heap?**

```
$ pmap 14719
14719:   beam.smp
0000000000400000       2168K r-x--  /opt/riak/erts-5.8.5/bin/beam.smp
000000000062D000        328K rw---  /opt/riak/erts-5.8.5/bin/beam.smp
000000000067F000    4193540K rw---  /opt/riak/erts-5.8.5/bin/beam.smp
00000001005C0000    4194296K rw---    [ anon ]
00000002005BE000    4192016K rw---    [ anon ]
0000000300382000    4193664K rw---    [ anon ]
00000004002E2000    4191172K rw---    [ anon ]
00000004FFFD3000    4194040K rw---    [ anon ]
00000005FFF91000    4194028K rw---    [ anon ]
00000006FFF4C000    4188812K rw---    [ anon ]
00000007FF9EF000     588224K rw---    [ heap ]
[...]
```

- **... and why does it keep growing?**
- **Would like to answer these in production**
    - Without restarting apps. Experimentation (backend=mmap, other allocators) wasn't working.

Memory: heap profiling

- **libumem was used for multi-threaded performance**
  - libumem == user-land slab allocator
- **detailed observability can be enabled, allowing heap profiling and leak detection**
  - While designed with speed and production use in mind, it still comes with some cost (time and space), and aren't on by default.
  - UMEM_DEBUG=audit

Memory: heap profiling

- **libumem provides some default observability**
  - Eg, slabs:

```
> ::umem_malloc_info
CACHE             BUFSZ MAXMAL BUFMALLC  AVG_MAL  MALLOCED  OVERHEAD  %OVER
0000000000707028      8      0        0        0         0         0   0.0%
000000000070b028     16      8     8730        8     69836   1054998 1510.6%
000000000070c028     32     16     8772       16    140352   1130491 805.4%
000000000070f028     48     32  1148038       25  29127788 156179051 536.1%
0000000000710028     64     48   344138       40  13765658  58417287 424.3%
0000000000711028     80     64       36       62      2226      4806 215.9%
0000000000714028     96     80     8934       79    705348   1168558 165.6%
0000000000715028    112     96  1347040       87 117120208 190389780 162.5%
0000000000718028    128    112   253107      111  28011923  42279506 150.9%
000000000071a028    160    144    40529      118   4788681   6466801 135.0%
000000000071b028    192    176      140      155     21712     25818 118.9%
000000000071e028    224    208       43      188      8101      6497  80.1%
000000000071f028    256    240      133      229     30447     26211  86.0%
0000000000720028    320    304       56      276     15455     12276  79.4%
0000000000723028    384    368       35      335     11726      7220  61.5%
[...]
```

Memory: heap profiling

- ... and heap (captured @14GB RSS):

```
> ::vmem
ADDR             NAME                       INUSE        TOTAL   SUCCEED  FAIL
fffffd7ffebed4a0 sbrk_top              9090404352  14240165888   4298117 84403
fffffd7ffebee0a8   sbrk_heap           9090404352   9090404352   4298117     0
fffffd7ffebeecb0     vmem_internal      664616960    664616960     79621     0
fffffd7ffebef8b8       vmem_seg         651993088    651993088     79589     0
fffffd7ffebf04c0       vmem_hash         12583424     12587008        27     0
fffffd7ffebf10c8       vmem_vmem            46200        55344        15     0
00000000006e7000     umem_internal      352862464    352866304     88746     0
00000000006e8000       umem_cache          113696       180224        44     0
00000000006e9000       umem_hash         13091328     13099008        86     0
00000000006ea000     umem_log                   0            0         0     0
00000000006eb000     umem_firewall_va           0            0         0     0
00000000006ec000       umem_firewall            0            0         0     0
00000000006ed000     umem_oversize     5218777974   5520789504   3822051     0
00000000006f0000     umem_memalign              0            0         0     0
0000000000706000     umem_default      2552131584   2552131584    307699     0
```

- **The heap is 9 GB (as expected), but sbrk_top total is 14 GB (equal to RSS). And growing.**
  - Are there Gbyte-sized malloc()/free()s?

Memory: malloc() profiling

Joyent

```
# dtrace -n 'pid$target::malloc:entry { @ = quantize(arg0); }' -p 17472
dtrace: description 'pid$target::malloc:entry ' matched 3 probes
^C
           value  ------------- Distribution ------------- count
               2 |                                         0
               4 |                                         3
               8 |@                                        5927
              16 |@@@@                                     41818
              32 |@@@@@@@@@                                81991
              64 |@@@@@@@@@@@@@@@@@@                       169888
             128 |@@@@@@@                                  69891
             256 |                                         2257
             512 |                                         406
            1024 |                                         893
            2048 |                                         146
            4096 |                                         1467
            8192 |                                         755
           16384 |                                         950
           32768 |                                         83
           65536 |                                         31
          131072 |                                         11
          262144 |                                         15
          524288 |                                         0
         1048576 |                                         1
         2097152 |                                         0
```

- No huge malloc()s, but RSS continues to climb.

Memory: malloc() profiling

Joyent

```
# dtrace -n 'pid$target::malloc:entry { @ = quantize(arg0); }' -p 17472
dtrace: description 'pid$target::malloc:entry ' matched 3 probes
^C
           value  ------------- Distribution ------------- count
               2 |                                         0
               4 |                                         3
               8 |@                                        5927
              16 |@@@@                                     41818
              32 |@@@@@@@@@                                81991
              64 |@@@@@@@@@@@@@@@@@@                       169888
             128 |@@@@@@@                                  69891
             256 |                                         2257
             512 |                                         406
            1024 |                                         893
            2048 |                                         146
            4096 |                                         1467
            8192 |                                         755
           16384 |                                         950
           32768 |                                         83
           65536 |                                         31
          131072 |                                         11
          262144 |                                         15
          524288 |                                         0
         1048576 |                                         1
         2097152 |                                         0
```

This tool (one-liner) profiles malloc() request sizes

- **No huge malloc()s, but RSS continues to climb.**

Memory: heap growth

- **Tracing why the heap grows via brk():**

```
# dtrace -n 'syscall::brk:entry /execname == "beam.smp"/ { ustack(); }'
dtrace: description 'syscall::brk:entry ' matched 1 probe
CPU     ID                    FUNCTION:NAME
 10     18                       brk:entry
              libc.so.1`_brk_unlocked+0xa
              libumem.so.1`vmem_sbrk_alloc+0x84
              libumem.so.1`vmem_xalloc+0x669
              libumem.so.1`vmem_alloc+0x14f
              libumem.so.1`vmem_xalloc+0x669
              libumem.so.1`vmem_alloc+0x14f
              libumem.so.1`umem_alloc+0x72
              libumem.so.1`malloc+0x59
              libstdc++.so.6.0.14`_Znwm+0x20
              libstdc++.so.6.0.14`_Znam+0x9
              eleveldb.so`_ZN7leveldb9ReadBlockEPNS_16RandomAccessFileERKNS_11Rea...
              eleveldb.so`_ZN7leveldb5Table11BlockReaderEPvRKNS_11ReadOptionsERKN...
              eleveldb.so`_ZN7leveldb12_GLOBAL__N_116TwoLevelIterator13InitDataBl...
              eleveldb.so`_ZN7leveldb12_GLOBAL__N_116TwoLevelIterator4SeekERKNS_5...
              eleveldb.so`_ZN7leveldb12_GLOBAL__N_116TwoLevelIterator4SeekERKNS_5...
              eleveldb.so`_ZN7leveldb12_GLOBAL__N_115MergingIterator4SeekERKNS_5S...
              eleveldb.so`_ZN7leveldb12_GLOBAL__N_16DBIter4SeekERKNS_5SliceE+0xcc
              eleveldb.so`eleveldb_get+0xd3
              beam.smp`process_main+0x6939
              beam.smp`sched_thread_func+0x1cf
              beam.smp`thr_wrapper+0xbe
```

This shows the user-land stack trace for every heap growth

Memory: heap growth

- **More DTrace showed the size of the malloc()s causing the brk()s:**

```
# dtrace -x dynvarsize=4m -n '
pid$target::malloc:entry { self->size = arg0; }
syscall::brk:entry /self->size/ { printf("%d bytes", self->size); }
pid$target::malloc:return { self->size = 0; }' -p 17472

dtrace: description 'pid$target::malloc:entry ' matched 7 probes
CPU     ID                    FUNCTION:NAME
  0     44                       brk:entry 8343520 bytes
  0     44                       brk:entry 8343520 bytes
[...]
```

- **These 8 Mbyte malloc()s grew the heap**
  - Even though the heap has Gbytes not in use
  - This is starting to look like an OS issue

Memory: allocator internals

Joyent

- **More tools were created:**
  - Show memory entropy (+ malloc - free)
    along with heap growth, over time
  - Show codepath taken for allocations
    compare successful with unsuccessful (heap growth)
  - Show allocator internals: sizes, options, flags
- **And run in the production environment**
  - Briefly; tracing *frequent* allocs does cost overhead
- **Casting light into what was a black box**

Memory: allocator internals

```
  4            <- vmem_xalloc                                   0
  4            -> _sbrk_grow_aligned                         4096
  4            <- _sbrk_grow_aligned                  17155911680
  4            -> vmem_xalloc                             7356400
  4             | vmem_xalloc:entry           umem_oversize
  4              -> vmem_alloc                            7356416
  4                -> vmem_xalloc                         7356416
  4                 | vmem_xalloc:entry       sbrk_heap
  4                  -> vmem_sbrk_alloc                   7356416
  4                    -> vmem_alloc                      7356416
  4                      -> vmem_xalloc                   7356416
  4                       | vmem_xalloc:entry   sbrk_top
  4                        -> vmem_reap                  16777216
  4                        <- vmem_reap          3178535181209758
  4                       | vmem_xalloc:return vmem_xalloc() == NULL, vm:
sbrk_top, size: 7356416, align: 4096, phase: 0, nocross: 0, min: 0, max: 0,
vmflag: 1
              libumem.so.1`vmem_xalloc+0x80f
              libumem.so.1`vmem_sbrk_alloc+0x33
              libumem.so.1`vmem_xalloc+0x669
              libumem.so.1`vmem_alloc+0x14f
              libumem.so.1`vmem_xalloc+0x669
              libumem.so.1`vmem_alloc+0x14f
              libumem.so.1`umem_alloc+0x72
              libumem.so.1`malloc+0x59
              libstdc++.so.6.0.3`_Znwm+0x2b
              libstdc++.so.6.0.3`_ZNSs4_Rep9_S_createEmmRKSaIcE+0x7e
```

Memory: solution

Joyent

- **These new tools and metrics pointed to the allocation algorithm "instant fit"**
  - Someone had suggested this earlier; the tools provided solid evidence that this really was the case here
- **A new version of libumem was built to force use of VM_BESTFIT**
  - and added by Robert Mustacchi as a tunable: UMEM_OPTIONS=allocator=best
- **Customer restarted Riak with new libumem version**
  - Problem solved

Memory: on the Cloud

Joyent

- **With OS virtualization, you can have: *Paging without scanning***
  - paging == swapping blocks with physical storage
  - swapping == swapping entire threads between main memory and physical storage
- **Resource control paging is unrelated to the page scanner, so, no vmstat scan rate (sr) despite anonymous paging**
- **More new tools: DTrace sysinfo:::anonpgin by process name, zonename**

Memory: summary

- **Superficial data available, detailed info not**
  - not by default
- **Many new tools were created**
  - not easy, but made possible with DTrace

Disk

Disk: problem

Joyent

- Application performance issues
- Disks look busy (iostat)
- Blame the disks?

```
$ iostat -xnz 1
[...]
                    extended device statistics
    r/s    w/s   kr/s   kw/s wait actv wsvc_t asvc_t  %w  %b device
  124.0  334.9 15677.2 40484.9  0.0  1.0    0.0    2.2   1  69 c0t1d0
                    extended device statistics
    r/s    w/s   kr/s   kw/s wait actv wsvc_t asvc_t  %w  %b device
  114.0  407.1 14595.9 49409.1  0.0  0.8    0.0    1.5   1  56 c0t1d0
                    extended device statistics
    r/s    w/s   kr/s   kw/s wait actv wsvc_t asvc_t  %w  %b device
   85.0  438.0 10814.8 53242.1  0.0  0.8    0.0    1.6   1  57 c0t1d0
```

- Many graphical tools are built upon iostat

Disk: on the Cloud

- **Tenants can't see each other**
  - Maybe a neighbor is doing a backup?
  - Maybe a neighbor is running a benchmark?
  - Can't see their processes (top/prstat)
- **Blame what you can't see**

Disk: VFS

- **Applications usually talk to a file system**
  - and are hurt by file system latency
- **Disk I/O can be:**
  - *unrelated* to the application: asynchronous tasks
  - *inflated* from what the application requested
  - *deflated* “ “
  - *blind* to issues caused higher up the kernel stack

Disk: issues with iostat(1)

- **Unrelated:**
  - other applications / tenants
  - file system prefetch
  - file system dirty data fushing
- **Inflated:**
  - rounded up to the next file system record size
  - extra metadata for on-disk format
  - read-modify-write of RAID5

Disk: issues with iostat(1)

- **Deflated:**
  - read caching
  - write buffering
- **Blind:**
  - lock contention in the file system
  - CPU usage by the file system
  - file system software bugs
  - file system queue latency

Disk: issues with iostat(1)

- **blind (continued):**
  - disk cache flush latency (if your file system does it)
  - file system I/O throttling latency
- **I/O throttling is a new ZFS feature for cloud environments**
  - adds artificial latency to file system I/O to throttle it
  - added by Bill Pijewski and Jerry Jelenik of Joyent

Disk: file system latency

- **Using DTrace to summarize ZFS read latency:**

```
$ dtrace -n 'fbt::zfs_read:entry { self->start = timestamp; }
fbt::zfs_read:return /self->start/ {
    @["ns"] = quantize(timestamp - self->start); self->start = 0; }'
dtrace: description 'fbt::zfs_read:entry ' matched 2 probes
^C

  ns
           value  ------------- Distribution ------------- count
             512 |                                         0
            1024 |@                                        6
            2048 |@@                                       18
            4096 |@@@@@@@                                  79
            8192 |@@@@@@@@@@@@@@@@@                        191
           16384 |@@@@@@@@@@                               112
           32768 |@                                        14
           65536 |                                         1
          131072 |                                         1
          262144 |                                         0
          524288 |                                         0
         1048576 |                                         0
         2097152 |                                         0
         4194304 |@@@                                      31
         8388608 |@                                        9
        16777216 |                                         0
```

Disk: file system latency

- **Using DTrace to summarize ZFS read latency:**

```
$ dtrace -n 'fbt::zfs_read:entry { self->start = timestamp; }
fbt::zfs_read:return /self->start/ {
    @["ns"] = quantize(timestamp - self->start); self->start = 0; }'
dtrace: description 'fbt::zfs_read:entry ' matched 2 probes
^C

  ns
           value  ------------- Distribution ------------- count
             512 |                                         0
            1024 |@                                        6
            2048 |@@                                       18
            4096 |@@@@@@@                                  79
            8192 |@@@@@@@@@@@@@@@@@   Cache reads          191
           16384 |@@@@@@@@@@                               112
           32768 |@                                        14
           65536 |                                         1
          131072 |                                         1
          262144 |                                         0
          524288 |                                         0
         1048576 |                                         0
         2097152 |                                         0
         4194304 |@@@   Disk reads                         31
         8388608 |@                                        9
        16777216 |                                         0
```

Disk: file system latency

- **Tracing zfs events using zfsslower.d:**

```
# ./zfsslower.d 10
TIME                 PROCESS     D    KB    ms FILE
2011 May 17 01:23:12 mysqld      R    16    19 /z01/opt/mysql5-64/data/xxxxx/xxxxx.ibd
2011 May 17 01:23:13 mysqld      W    16    10 /z01/var/mysql/xxxxx/xxxxx.ibd
2011 May 17 01:23:33 mysqld      W    16    11 /z01/var/mysql/xxxxx/xxxxx.ibd
2011 May 17 01:23:33 mysqld      W    16    10 /z01/var/mysql/xxxxx/xxxxx.ibd
2011 May 17 01:23:51 httpd       R    56    14 /z01/home/xxxxx/xxxxx/xxxxx/xxxxx/xxxxx
^C
```

- **Argument is the minimum latency in milliseconds**

Disk: file system latency

- **Can trace this from other locations too:**
  - VFS layer: filter on desired file system types
  - syscall layer: filter on file descriptors for file systems
  - application layer: trace file I/O calls

Disk: file system latency

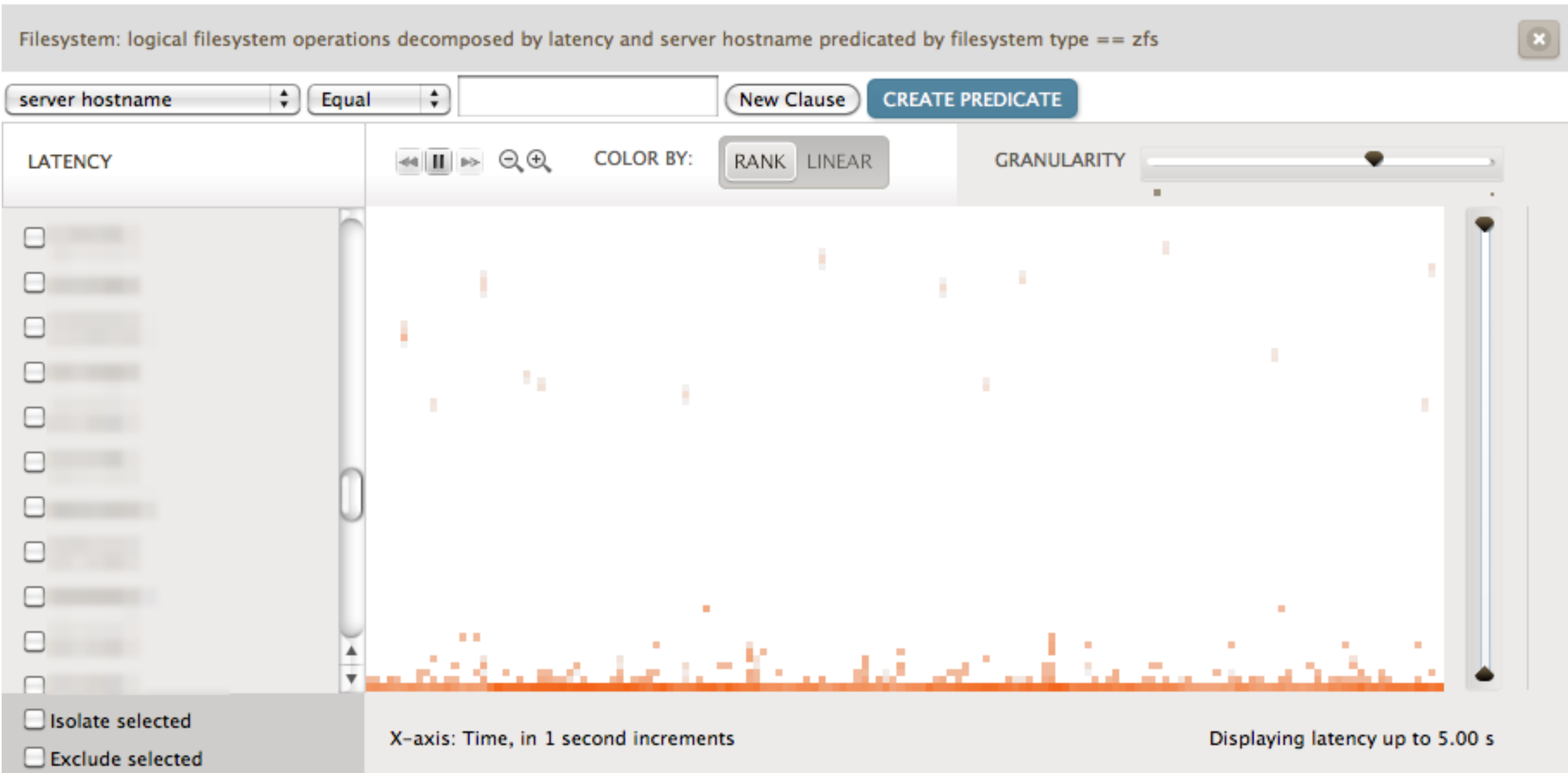
- **And using SystemTap:**

```
# ./vfsrlat.stp
Tracing... Hit Ctrl-C to end
^C
[..]
   ext4 (ns):
 value |-------------------------------------------------- count
   256 |                                                       0
   512 |                                                       0
  1024 |                                                      16
  2048 |                                                      17
  4096 |                                                       4
  8192 |@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@  16321
 16384 |                                                      50
 32768 |                                                       1
 65536 |                                                      13
131072 |                                                       0
262144 |                                                       0
```

- **Traces vfs.read to vfs.read.return, and gets the FS type via: $file->f_path->dentry->d_inode->i_sb->s_type->name**
  - Warning: this script has crashed ubuntu/CentOS; I'm told RHEL is better

Disk: file system visualizations

- **File system latency as a heat map (Cloud Analytics):**

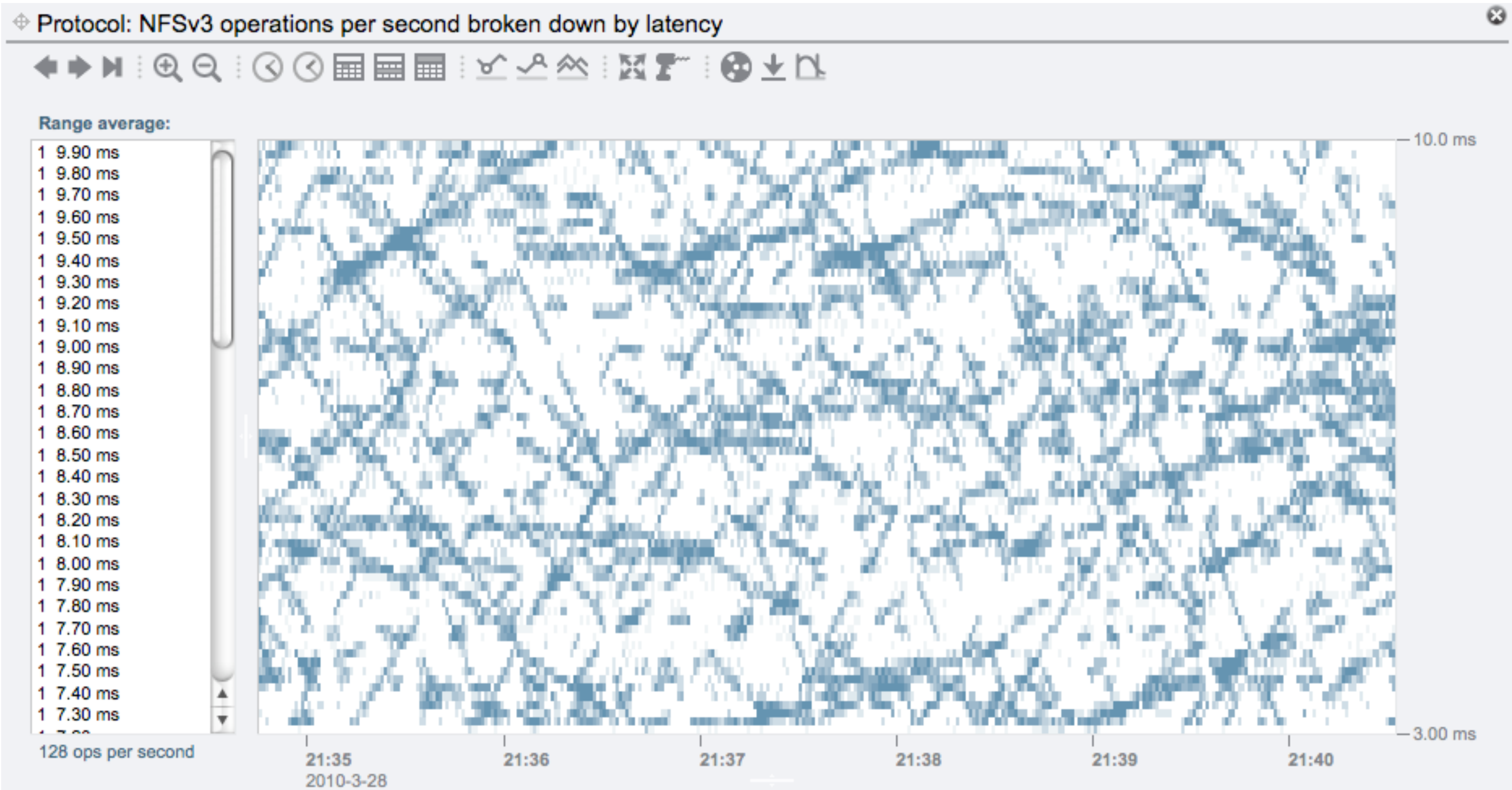


- **This screenshot shows severe outliers**

Disk: file system visualizations

Joyent

- **Sometimes the heat map is very surprising:**



  - This screenshot is from the Oracle ZFS Storage Appliance

Disk: summary

- Misleading data available
- New tools/metrics created
- Latency visualizations

Network

Network: problem

Joyent

- **TCP SYNs queue in-kernel until they are accept()ed**
- **The queue length is the TCP listen backlog**
    - may be set in listen()
    - and limited by a system tunable (usually 128)
        - on SmartOS: tcp_conn_req_max_q
- **What if the queue remains full**
    - eg, application is overwhelmed with other work,
    - or CPU starved
- **... and another SYN arrives?**

Network: TCP listen drops

Joyent

- **Packet is dropped by the kernel**
  - fortunately a counter is bumped:

```
$ netstat -s | grep Drop
        tcpTimRetransDrop   =    56     tcpTimKeepalive     =  2582
        tcpTimKeepaliveProbe=  1594     tcpTimKeepaliveDrop =    41
        tcpListenDrop       =3089298    tcpListenDropQ0     =     0
        tcpHalfOpenDrop     =     0     tcpOutSackRetrans   =1400832
        icmpOutDrops        =     0     icmpOutErrors       =     0
        sctpTimRetrans      =     0     sctpTimRetransDrop  =     0
        sctpTimHearBeatProbe=     0     sctpTimHearBeatDrop =     0
        sctpListenDrop      =     0     sctpInClosed        =     0
```

- **Remote host waits, and then retransmits**
  - TCP retransmit interval; usually 1 or 3 seconds

Network: predicting drops

- **How do we know if we are close to dropping?**
  - An early warning

Network: tcpconnreqmaxq.d

- DTrace script traces drop events, if they occur:

```
# ./tcpconnreqmaxq.d
Tracing... Hit Ctrl-C to end.
2012 Jan 19 01:37:52 tcp_input_listener:tcpListenDrop cpid:11504
2012 Jan 19 01:37:52 tcp_input_listener:tcpListenDrop cpid:11504
2012 Jan 19 01:37:52 tcp_input_listener:tcpListenDrop cpid:11504
2012 Jan 19 01:37:52 tcp_input_listener:tcpListenDrop cpid:11504
2012 Jan 19 01:37:52 tcp_input_listener:tcpListenDrop cpid:11504
2012 Jan 19 01:37:52 tcp_input_listener:tcpListenDrop cpid:11504
2012 Jan 19 01:37:52 tcp_input_listener:tcpListenDrop cpid:11504
2012 Jan 19 01:37:52 tcp_input_listener:tcpListenDrop cpid:11504
2012 Jan 19 01:37:52 tcp_input_listener:tcpListenDrop cpid:11504
2012 Jan 19 01:37:52 tcp_input_listener:tcpListenDrop cpid:11504
2012 Jan 19 01:37:52 tcp_input_listener:tcpListenDrop cpid:11504
2012 Jan 19 01:37:52 tcp_input_listener:tcpListenDrop cpid:11504
2012 Jan 19 01:37:52 tcp_input_listener:tcpListenDrop cpid:11504
2012 Jan 19 01:37:52 tcp_input_listener:tcpListenDrop cpid:11504
[...]
```

- ... and when Ctrl-C is hit:

Network: tcpconnreqmaxq.d

```
tcp_conn_req_cnt_q distributions:

  cpid:3063                                             max_q:8
           value  ------------- Distribution ------------- count
              -1 |                                         0
               0 |@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@ 1
               1 |                                         0

  cpid:11504                                            max_q:128
           value  ------------- Distribution ------------- count
              -1 |                                         0
               0 |@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@      7279
               1 |@@                                       405
               2 |@                                        255
               4 |@                                        138
               8 |                                         81
              16 |                                         83
              32 |                                         62
              64 |                                         67
             128 |                                         34
             256 |                                         0

tcpListenDrops:
   cpid:11504                    max_q:128                    34
```

Network: tcpconnreqmaxq.d

Joyent

```
tcp_conn_req_cnt_q distributions:

  cpid:3063                                                   max_q:8
           value  ------------- Distribution ------------- count
              -1 |                                         0
               0 |@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@ 1
               1 |                                         0

  cpid:11504                                                  max_q:128
           value  ------------- Distribution ------------- count
              -1 |                                         0
               0 |@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@      7279
               1 |@@                                       405
               2 |@                                        255
               4 |@                                        138
               8 |                                         81
              16 |                                         83
              32 |                                         62
              64 |                                         67
             128 |                                         34
             256 |                                         0

tcpListenDrops:
   cpid:11504                  max_q:128                        34
```

value in use

Length of queue measured on SYN event

Network: tcplistendrop.d

- **More details can be fetched as needed:**

```
# ./tcplistendrop.d
TIME                     SRC-IP            PORT     DST-IP            PORT
2012 Jan 19 01:22:49  10.17.210.103    25691 -> 192.192.240.212    80
2012 Jan 19 01:22:49  10.17.210.108    18423 -> 192.192.240.212    80
2012 Jan 19 01:22:49  10.17.210.116    38883 -> 192.192.240.212    80
2012 Jan 19 01:22:49  10.17.210.117    10739 -> 192.192.240.212    80
2012 Jan 19 01:22:49  10.17.210.112    27988 -> 192.192.240.212    80
2012 Jan 19 01:22:49  10.17.210.106    28824 -> 192.192.240.212    80
2012 Jan 19 01:22:49  10.12.143.16     65070 -> 192.192.240.212    80
2012 Jan 19 01:22:49  10.17.210.100    56392 -> 192.192.240.212    80
2012 Jan 19 01:22:49  10.17.210.99     24628 -> 192.192.240.212    80
2012 Jan 19 01:22:49  10.17.210.98     11686 -> 192.192.240.212    80
2012 Jan 19 01:22:49  10.17.210.101    34629 -> 192.192.240.212    80
[...]
```

- **Just tracing the drop code-path**
  - Don’t need to pay the overhead of sniffing all packets

Network: DTrace code

- **Key code from tcplistendrop.d:**

```
fbt::tcp_input_listener:entry  { self->mp = args[1]; }
fbt::tcp_input_listener:return { self->mp = 0; }

mib:::tcpListenDrop
/self->mp/
{
        this->iph = (ipha_t *)self->mp->b_rptr;
        this->tcph = (tcph_t *)(self->mp->b_rptr + 20);
        printf("%-20Y  %-18s %-5d -> %-18s %-5d\n", walltimestamp,
            inet_ntoa(&this->iph->ipha_src),
            ntohs(*(uint16_t *)this->tcph->th_lport),
            inet_ntoa(&this->iph->ipha_dst),
            ntohs(*(uint16_t *)this->tcph->th_fport));
}
```

- **This uses the unstable interface fbt provider**
    - a stable tcp provider now exists, which is better for more common tasks - like connections by IP

Network: summary

Joyent

- **For TCP, while many counters are available, they are system wide integers**
- **Custom tools can show more details**
  - addresses and ports
  - kernel state
  - needs kernel access and dynamic tracing

Data Recap

- **Problem types**
  - CPU utilization — scaleability
  - CPU usage — scaleability
  - CPU latency — observability
  - Memory — observability
  - Disk — observability
  - Network — observability

Data Recap

Joyent

- **Problem types, solution types**
  - CPU utilization — scaleability
  - CPU usage — scaleability
  - CPU latency — observability
  - Memory — observability
  - Disk — observability
  - Network — observability

visualizations

metrics

Theory

Performance Analysis

Joyent

- **Goals**
  - Capacity planning
  - Issue analysis

Performance Issues

- **Strategy**
  - Step 1: is there a problem?
  - Step 2: which subsystem/team is responsible?
- **Difficult to get past these steps without reliable metrics**

Problem Space

- **Myths**
  - Vendors provide good metrics with good coverage
  - The problem is to line-graph them
- **Realities**
  - Metrics can be wrong, incomplete and misleading, requiring time and expertise to interpret
  - Line graphs can hide issues

Problem Space

Joyent

- **Cloud computing confuses matters further:**
  - hiding metrics from neighbors
  - throttling performance due to invisible neighbors

Example Problems

- **Included:**
  - Understanding utilization across 5,312 CPUs
  - Using disk I/O metrics to explain application performance
  - A lack of metrics for memory growth, packet drops, ...

Example Solutions: tools

- Device utilization heat maps for CPUs
- Flame graphs for CPU profiling
- CPU dispatcher queue latency by zone
- CPU caps latency by zone
- malloc() size profiling
- Heap growth stack backtraces
- File system latency distributions
- File system latency tracing
- TCP accept queue length distribution
- TCP listen drop tracing with details

Key Concepts

Joyent

- **Visualizations**
  - heat maps for device utilization and latency
  - flame graphs
- **Custom metrics often necessary**
  - Latency-based for issue analysis
  - If coding isn't practical/timely, use dynamic tracing
- **Cloud Computing**
  - Provide observability (often to show what the problem *isn't*)
  - Develop new metrics for resource control effects

DTrace

- **Many problems were only solved thanks to DTrace**
- **In the SmartOS cloud environment:**
  - The compute node (global zone) can DTrace everything (except for KVM guests, for which it has a limited view: resource I/O + some MMU events, so far)
  - SmartMachines (zones) have the DTrace syscall, profile (their user-land only), pid and USDT providers
  - Joyent Cloud Analytics uses DTrace from the global zone to give extended details to customers

Performance

Joyent

- The more you know, the more you don't
- Hopefully I've turned some unknown-unknowns into known-unknowns

Thank you

- **Resources:**
  - http://dtrace.org/blogs/brendan
    - More CPU utilization visualizations: http://dtrace.org/blogs/brendan/2011/12/18/visualizing-device-utilization/
    - Flame Graphs: http://dtrace.org/blogs/brendan/2011/12/16/flame-graphs/ and http://github.com/brendangregg/FlameGraph
    - More iostat(1) & file system latency discussion: http://dtrace.org/blogs/brendan/tag/filesystem-2/
  - Cloud Analytics:
    - OSCON slides: http://dtrace.org/blogs/dap/files/2011/07/ca-oscon-data.pdf
    - Joyent: http://joyent.com
  - brendan@joyent.com