SDC 19

September 23-26, 2019
Santa Clara, CA

Average Storage Latency is Lacking. Let's Fix It!

Richard Elling
Viking Enterprise Solutions
a Division of Sanmina Corporation

The Problem

VIKING™ Enterprise Solutions

SDC 19

Storage system latency does not have a normal distribution

- HDDs seek & rotate
- Flash SSDs garbage collection
- Fabrics
- Ordering & Re-ordering
- Multiple, serial queues
- Error recovery and retry

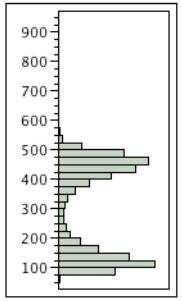
Flash SSD Latency Anomaly

VIKING™ Enterprise Solutions

SDC 19

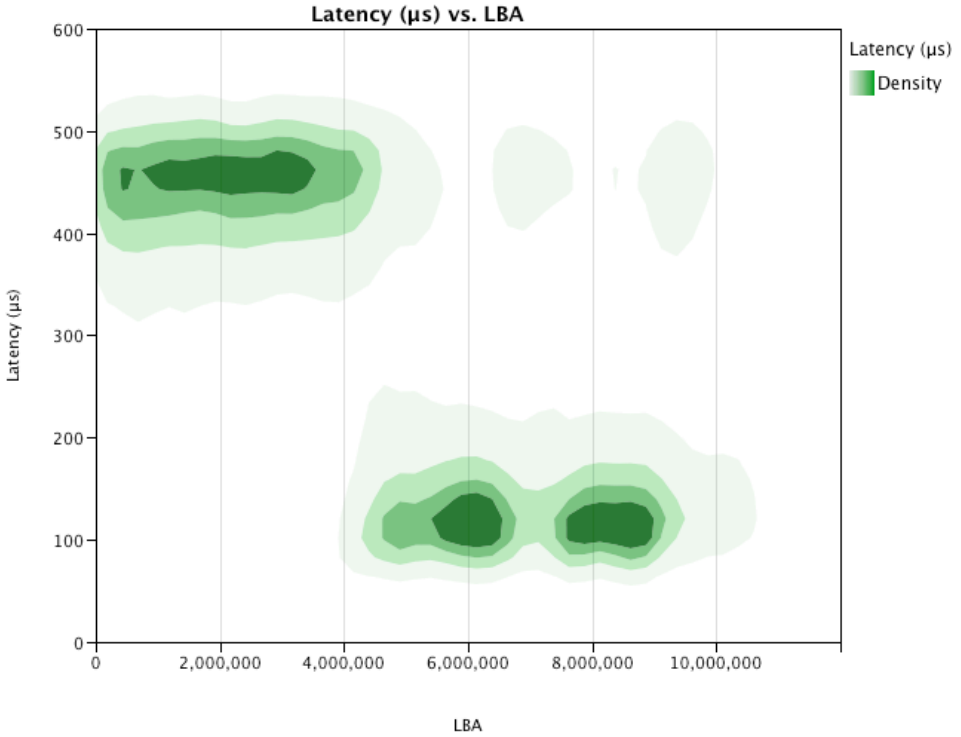
Distributions

Latency (µs)

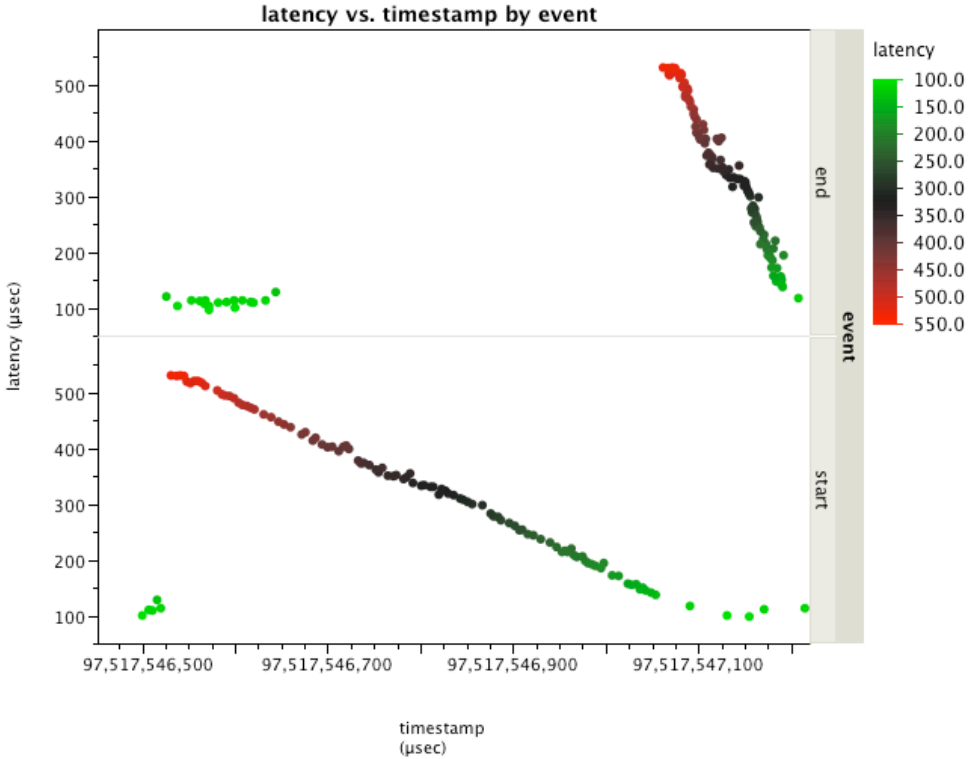


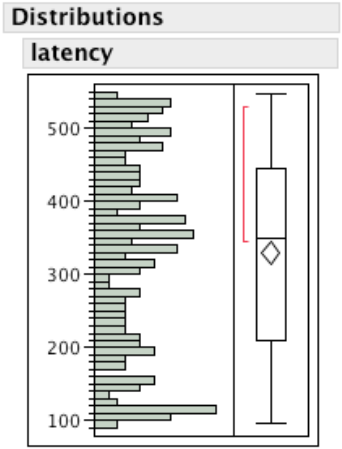
Quantiles		
100.0%	maximum	931
99.5%		535
97.5%		509
90.0%		483
75.0%	quartile	453
50.0%	median	374
25.0%	quartile	130
10.0%		103
2.5%		81
0.5%		76
0.0%	minimum	69

Moments	
Mean	303.49726
Std Dev	159.29382
Std Err Mean	0.2662102
Upper 95% Mean	304.01902
Lower 95% Mean	302.97549
N	358054



Ordering Problem





Popular Tools: iostat

Ubuntu

```
$ iostat -x
Linux 4.15.0-13-generic (elvis)          09/13/2019          _x86_64_          (3 CPU)
...
Device  r/s  w/s  rkB/s wkB/s rrqm/s wrqm/s %rrqm %wrqm r_await w_await aqu-sz rareq-sz wareq-sz svctm  %util
sda     0.35 0.26  3.53 10.47   0.07   0.62 16.38 70.24    0.76   14.33   0.00    10.17    40.10  0.43   0.03
...
```

OSX

```
$ iostat -o
      disk0       disk2       cpu    load average
 sps tps msps  sps tps msps  us sy id   1m   5m   15m
3092 177  0.0 1042  10  0.0   7  6 87  1.46 1.89 1.93
```

Note: ignore Linux svctm, it is broken

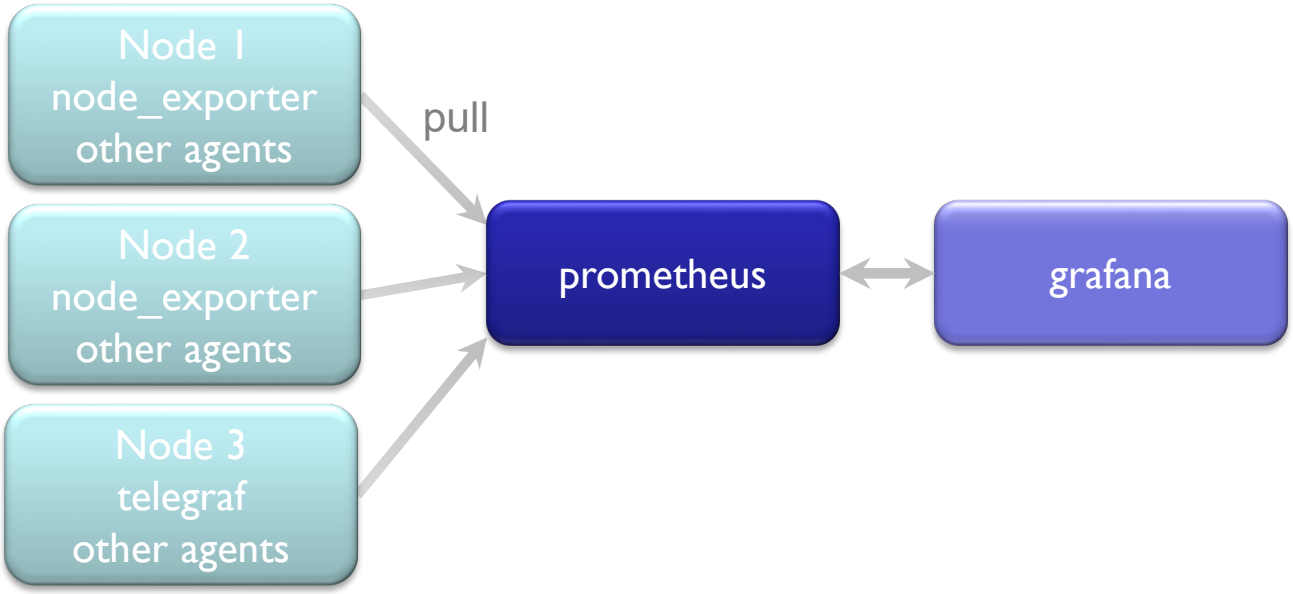
Linux /proc/diskstats

VIKING™ Enterprise Solutions

```
# cat /proc/diskstats
...
 8 0 sda 94376 18486 1919722 71620 70919 167617 5696296 1017520 0 71476 1102048
...
```

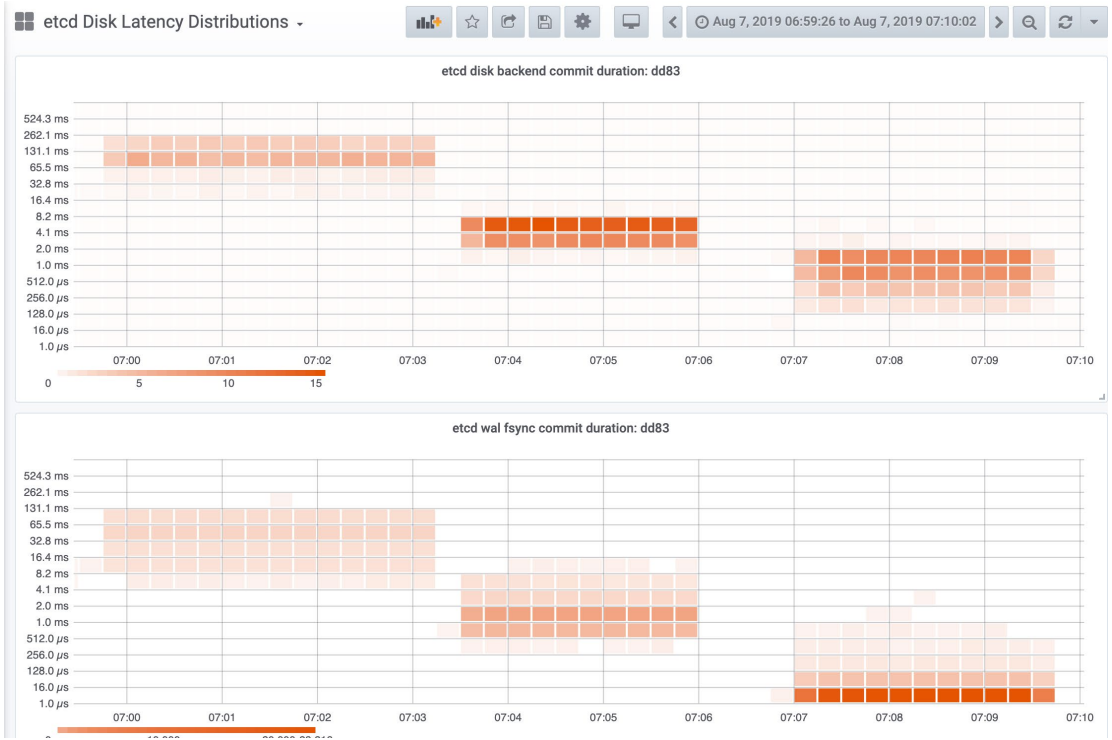
Field	Description	Field	Description
1	major number	8	# of writes completed
2	minor number	9	# of writes merged
3	device name	10	# of sectors written
4	# of completed reads	11	# of milliseconds spent writing
5	# of reads merged	12	# of I/Os in progress
6	# of sectors read	13	# of milliseconds spent doing I/Os
7	# of milliseconds spent reading	14	weighted # of milliseconds spent doing I/Os

Popular Tools: prometheus



prometheus Example

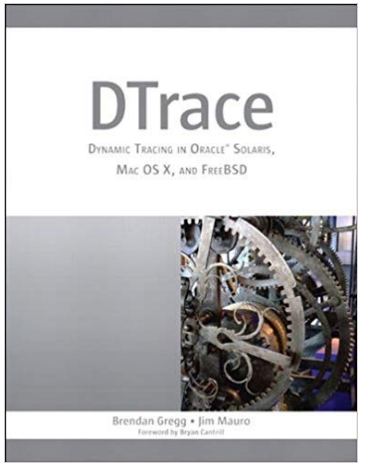
VIKING™ Enterprise Solutions



Sampling: dtrace

VIKING™ Enterprise Solutions

- Dynamic Tracing (dtrace)
- Ported from Solaris to FreeBSD and OSX
- Halfway ported to OEL, see BPF for another path forward



dtrace Latency Example

VIKING™ Enterprise Solutions

SDC 19

Santa Clara, CA

```
# cat iolatency.d
#!/usr/sbin/dtrace -s

io:::start
{
        start[arg0] = timestamp;
}

io:::done
/start[arg0]/
{
        @time["disk I/O latency (ns)"] = quantize(timestamp - start[arg0]);
        start[arg0] = 0;
}
```

dtrace Latency Example

VIKING™ Enterprise Solutions

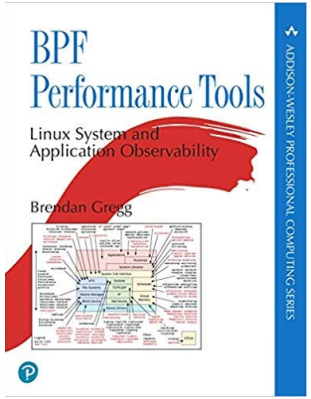
SDC 19

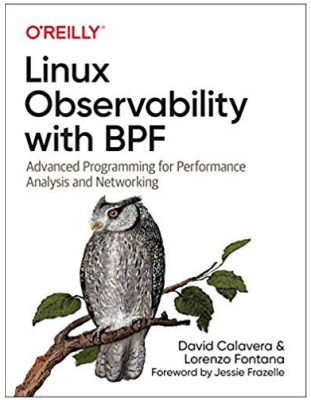
```
# ./iolatency.d
dtrace: script './iolatency.d' matched 2 probes
^C

  disk I/O latency (ns)
         value  ------------- Distribution ------------- count
         32768 |                                         0
         65536 |                                         94
        131072 |@@                                       361
        262144 |@@@@@@@@@@@@@@@@@@@@@@@@@@@              4349
        524288 |@@@@@@@@@                                1617
       1048576 |@@                                       366
       2097152 |@                                        178
       4194304 |@                                        197
       8388608 |@                                        296
      16777216 |@@                                       313
      33554432 |@                                        273
      67108864 |                                         34
     134217728 |                                         0
     268435456 |                                         2
     536870912 |                                         0
```

Sampling: BPF

- No complete port of dtrace to Linux, so…
- Berkeley Packet Filter (BPF) meets the Linux kernel
  - Also known as extended BPF (eBPF)
- Similar to dtrace, systemtap, and friends
- Vibrant community with many good scripts to learn from and adapt

www.iovisor.org/technology/ebpf





2 Good Ideas: eBPF_exporter

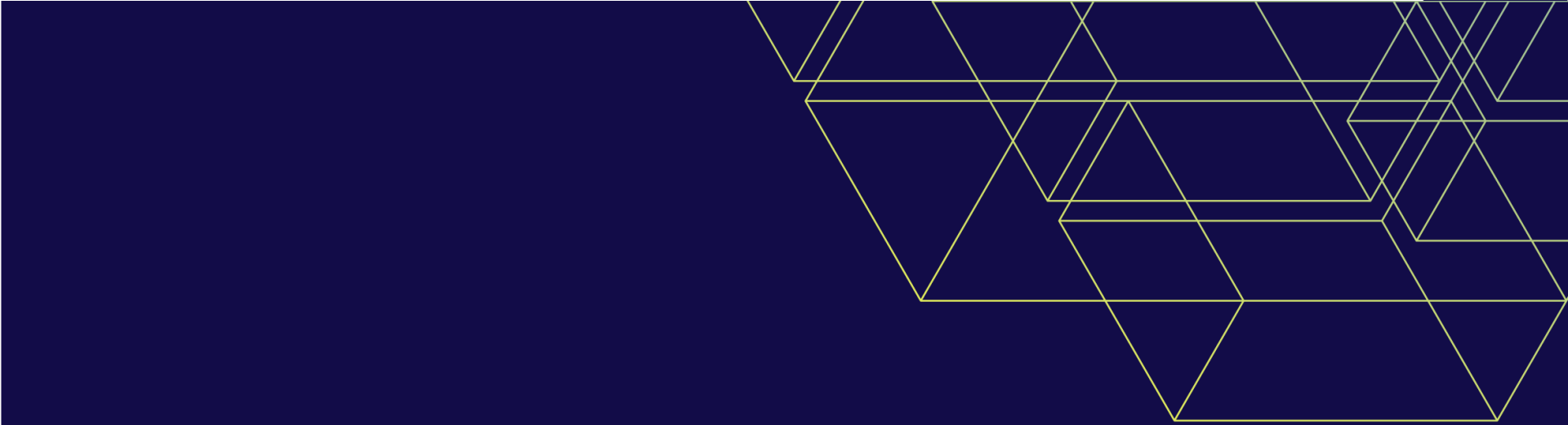
VIKING Enterprise Solutions

SDC 19

- Descibe eBPF as YAML
- ebpf_exporter deamon
  - reads YAML
  - runs BPF
  - results are available to prometheus
- Use to sample latency distributions on almost anything… userland or kernel

Samplers Aren't Perfect

- When the going gets tough, samplers are adversely affected
  - Can miss samples
  - Can commit suicide (protect the system)
- Require elevated roles -- may conflict with production security policies
- Can impact overall performance



Good idea: use samplers for debug and prototyping then add useful latency distributions in the code

Exemplar: chrome

VIKING™ Enterprise Solutions

Try it! chrome://histograms

```
Histogram: Event.Latency.OS.MOUSE_WHEEL recorded 18505 samples, mean = 1176.9 (flags = 0x41)
   0 ...
 258 O (1 = 0.0%) {0.0%}
 340 -O (52 = 0.3%) {0.0%}
 448 ----------------------------------O (2524 = 13.6%) {0.3%}
 590 ------------------------------------------------------------------------O (4909 = 26.5%) {13.9%}
 777 ------------------------------------------------------------------------O (5178 = 28.0%) {40.5%}
1023 -------------------------------O (2234 = 12.1%) {68.4%}
1347 -----------O (763 = 4.1%) {80.5%}
1774 ----------O (684 = 3.7%) {84.6%}
2336 -------------O (931 = 5.0%) {88.3%}
3077 --------------O (987 = 5.3%) {93.4%}
4053 ---O (194 = 1.0%) {98.7%}
5338 O (21 = 0.1%) {99.7%}
7031 O (14 = 0.1%) {99.9%}
9260 O (7 = 0.0%) {99.9%}
12196 O (2 = 0.0%) {100.0%}
16063 O (1 = 0.0%) {100.0%}
21156 O (0 = 0.0%) {100.0%}
27864 O (3 = 0.0%) {100.0%}
36699 ...
```

Realities of Kernel

- chrome is cool, but javascript uses float64
  - Number.MAX_SAFE_INTEGER (2^{53} -1)
  - Number.MIN_SAFE_INTEGER (-(2^{53} – 1))
- Kernels tend to only do integer math
  - Integer math is more CPU efficient
  - Many performance counters are uint64
    - elapsed nanoseconds * # of I/Os = big number

Example: ZFSonLinux

VIKING™ Enterprise Solutions

```
# zpool iostat -w

testpool    total_wait    disk_wait    syncq_wait    asyncq_wait
latency     read  write  read  write  read  write  read  write  scrub   trim
----------  -----  -----  -----  -----  -----  -----  -----  -----  -----  -----
1ns         0     0     0     0     0     0     0     0     0     0
...
2us         0     0     0     0    37    31     0    36     2     0
4us         0     0     0     0    33    25     0   117    39     0
8us         0     0     0     0     7     6     0    11    16     0
16us        0     0     0     0     1     0     0     6     2     0
32us        0     0     0     0     1     0     0     7     5     0
65us        0     0     0     0     0     0     0    20     4     0
131us       6     7     6    31     2     0     0    30     2     0
262us      23    73    25   381     2     0     0    47     8     0
524us      36   158    44   465     0     0     0    88    21     0
1ms        40   194    56   187     0     0     0   143    37     0
2ms        48   282    65    57     0     0     0   242    33     0
4ms        56   320    99     9     0     0     0   267    50     0
8ms        94    96    41     7     0     0     0    58    40     0
16ms       30     8     6     1     0     0     0     4     5     0
33ms        9     0     5     0     0     0     0     0     6     0
...
```

Choosing Buckets

- Simplest and fastest is log2

```
uint64_t elapsed = NOW() - start;
bucket = 64 - __builtin_clzll(elapsed);  // 64 – count leading zeroes
```

  - ok for storage or many orders of magnitude
  - sometimes more precision is desired
- Linear
  - ok when range is 1 or 2 orders of magnitude
  - division is 10-40x slower than addition/subtraction
  - range checking can cause branches (slow)
- Log-linear
  - mix of log10 with 10 buckets per decade
  - slower than linear, but good resolution
- Natural log
  - floating point – not suitable for kernel

Arrays of Buckets

- Simplest implementation is a global array for latency buckets
  - used by OpenZFS
  - can lead to cache line stalls – measure instructions per cycle (IPC)
  - can be space efficient
  - for consistency, use atomics – but IPC can suffer
- Better implementation is one array per processor
  - eliminates cache line stalls = better IPC
  - no need for atomics
  - requires reader to sum across processors

Gotchas

VIKING™ Enterprise Solutions

SDC 19

- Getting time can be expensive
- uint64_t counters are convenient in kernel, but some tools, like prometheus, only support float64 values
- In-flight requests can't be easily measured
  - requests that do not complete have no end time
- Most iostat implementations are for HDDs
  - lack resolution for SSDs

One More Thing…

VIKING™ Enterprise Solutions

If you can do only one more thing…

… at least add a latency threshold counter

- Pick a (tunable) unexpected large latency threshold value
- Increment a counter when latency exceeds the threshold
- Make the counter accessible to instrumentation tools (node_exporter, telegraf, et.al.)

Conclusion

VIKING™ Enterprise Solutions

SDC 19

- Average latencies are readily available, but are lacking in information
- Latency histograms can reveal latency problems
- Samplers can measure elapsed time between function calls or function entry/return
- But samplers are not suitable for running continuously
- Better solution is to collect latency histograms in kernel modules or applications
- We can do this! Let's do it!

Thanks

Richard.Elling@VikingEntperise.com