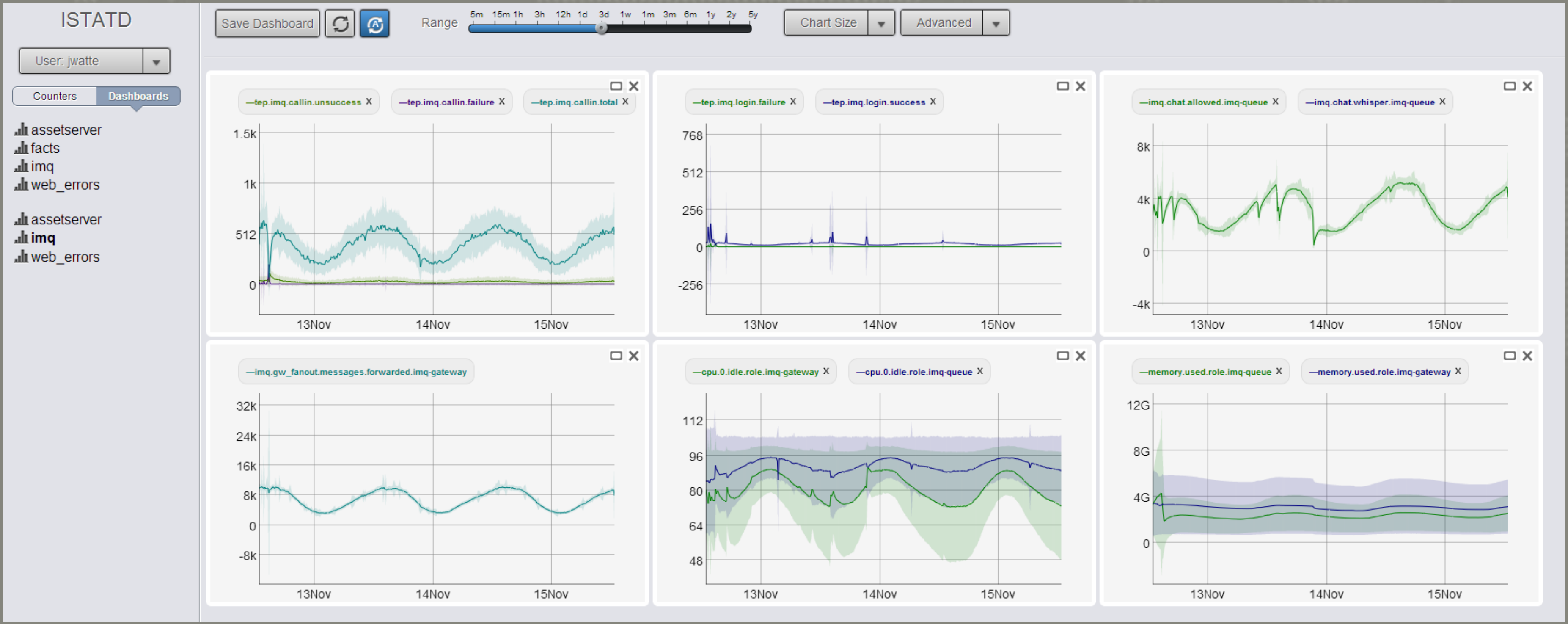
1~~0~~50,000 Couters, every 10 seconds
Native Linux throughput in reality

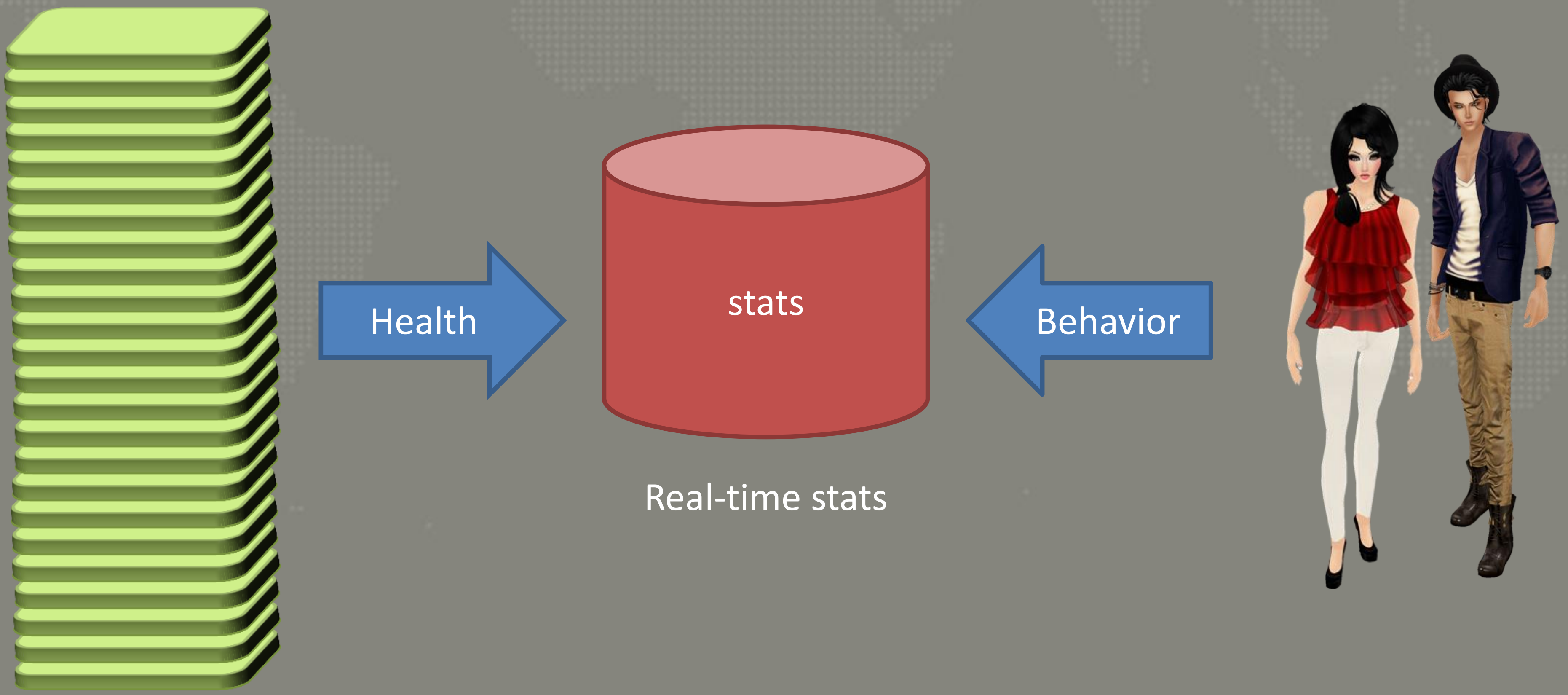
Jon Watte

Technical Director, IMVU Inc

Context


ISTATD
Save Dashboard
Range
5m 15m 1h 3h 12h 1d 3d 1w 1m 3m 6m 1y 2y 5y
Chart Size
Advanced
User: jwatte
Counters
Dashboards
assetserver
facts
imq
web_errors
assetserver
imq
web_errors
—tep.imq.callin.unsuccess X
—tep.imq.callin.failure X
—tep.imq.callin.total X
1.5k
1k
512
0
13Nov
14Nov
15Nov
—tep.imq.login.failure X
—tep.imq.login.success X
768
512
256
0
-256
13Nov
14Nov
15Nov
—imq.chat.allowed.imq-queue X
—imq.chat.whisper.imq-queue X
8k
4k
0
-4k
13Nov
14Nov
15Nov
—imq.gw_fanout.messages.forwarded.imq-gateway
32k
24k
16k
8k
0
-8k
13Nov
14Nov
15Nov
—cpu.0.idle.role.imq-gateway X
—cpu.0.idle.role.imq-queue X
112
96
80
64
48
13Nov
14Nov
15Nov
—memory.used.role.imq-queue X
—memory.used.role.imq-gateway X
12G
8G
4G
0
13Nov
14Nov
15Nov

Cluster Diagram



Servers

Health

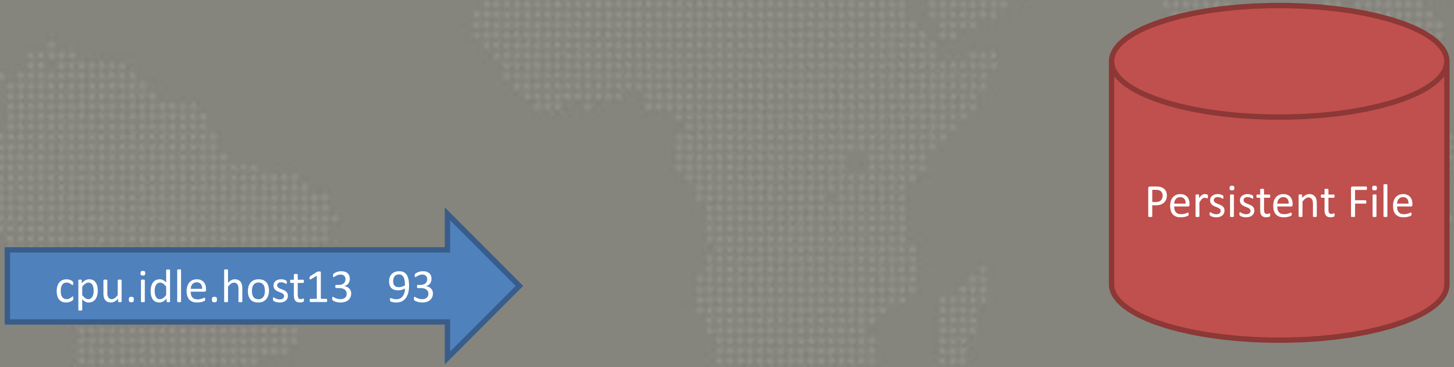
stats

Real-time stats

Behavior

Application

Order of magnitude

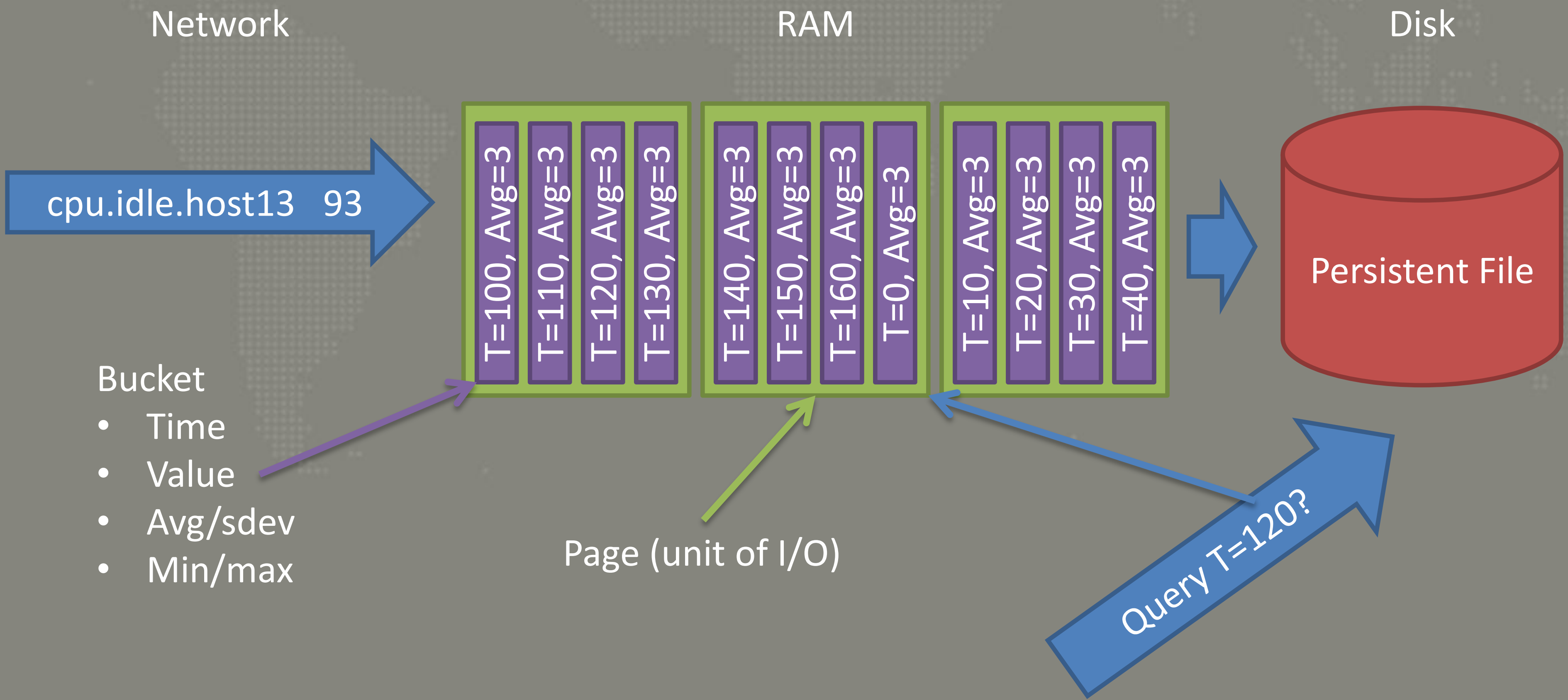

cpu.idle.host13   93

Persistent File

150,000 counter names
3 files each
100,000 events per second

Resolution	Retention	Size/Ctr
10 sec	10+ days	2.7 MB
5 min	1+ year	3.4 MB
1 hr	6+ years	1.7 MB

Istatd diagram



Bucket

- Time
- Value
- Avg/sdev
- Min/max

Page (unit of I/O)

Two Challenges

Latency Hierarchy

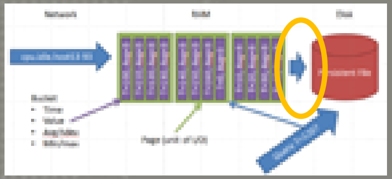
L1 Cache	1 ns
L3 Cache	10 ns
DRAM	100 ns
SSD	100,000 ns
Spinny Disk	10,000,000 ns

Amdahl’s Law

$$T = \frac{1}{(1-P)+\frac{P}{S}}$$

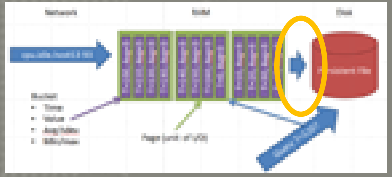
T is new throughput multiplier
P is proportion that is parallelized
S is parallel multiplier (up to 24x for 24-core)

Latency: Async File I/O



```
hFile = CreateFile(…,
 FILE_FLAG_OVERLAPPED, 0);

// Start I/O
OVERLAPPED olp = { … };
ReadFile(hFile, …, &olp);

// In worker thread
GetQueuedCompletionStatus(…);
// … Use data here
```

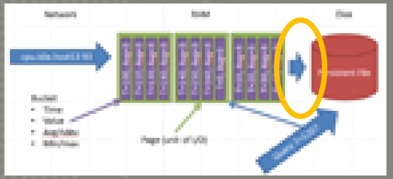
```
fd = open(…);

// Wait for ready
epoll_event ev = { … };
epoll_ctl(…, &ev);

// In worker thread
epoll_wait(…);
read(fd, …);
// … Use data here
```

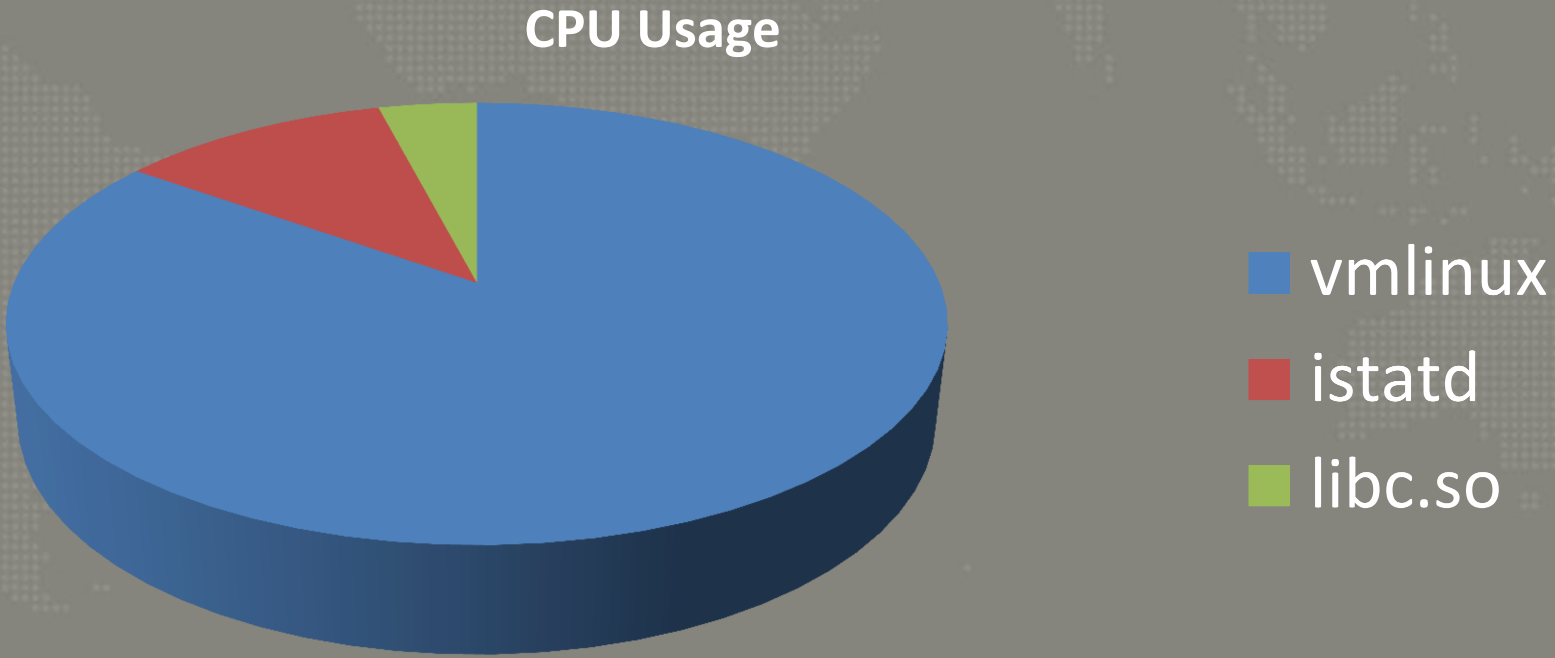


Fake Async: Using mmap()

```
fd = open(“name”, …);
void *ptr = mmap(0, size, PROT_READ|PROT_WRITE,
  MAP_SHARED, fd, offset);
madvise(ptr, length, MADV_WILLNEED);


// … do other stuff for a while …


// use ptr here
```

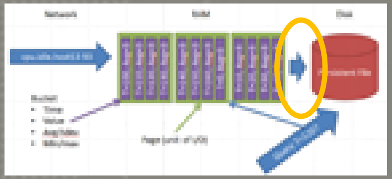
mmap() Limitations





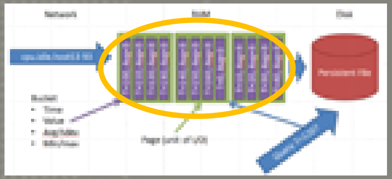
Linux VM mapping tree becomes deep and serializes

“Async-ish” I/O Compromise



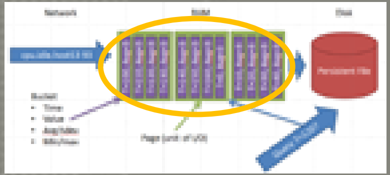
- Writing is “asynchronous” as long as there is free kernel buffer space
  - Use a task that cyclically flushes open files
- Over-commit on threads, and do synchronous reads
  - We know to pre-fetch the high-frequency counters

Contention: Serializing on Locks



- A single hash table for all counters
  - Same problem as Linux mmap()!
- Frequent operations on this table ended up serializing on the lock protecting the table

Solution: Sharded Locking

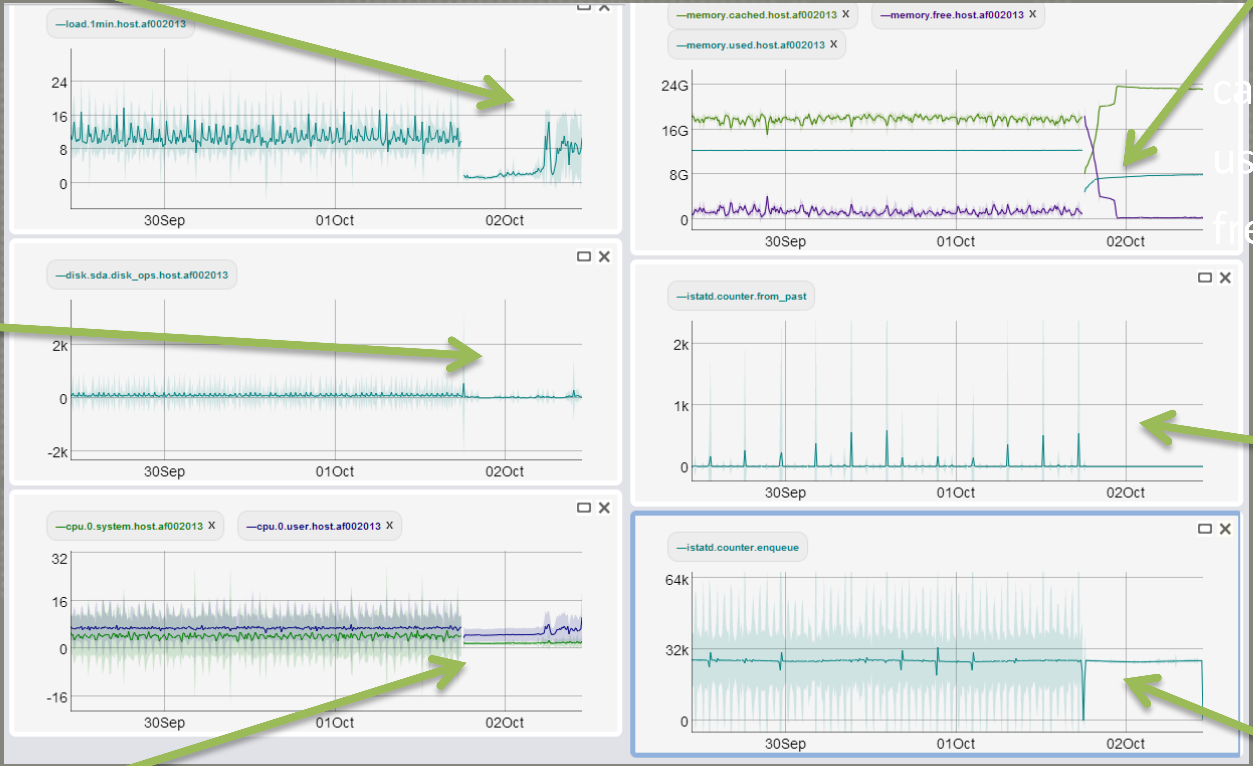


- If I was to farm out to 24 cores, I’d want 24 locks
- I can’t know exact 1:1 mapping from threads to locks
- Over-allocate locks, so most of them are not held
- 256 separate hash tables, each with 1 lock
  - In-memory sharded locking

Reality: Ambient Challenges

- Backing up a heavily loaded, real-time machine
  - The Replication Hack
- Occasional “network events”
  - Agent-side buffering
- Linux kernels move on
  - Actually an opportunity

2.6 -> 3.2 Upgrade



Load

Memory
Usage

cached

used

free

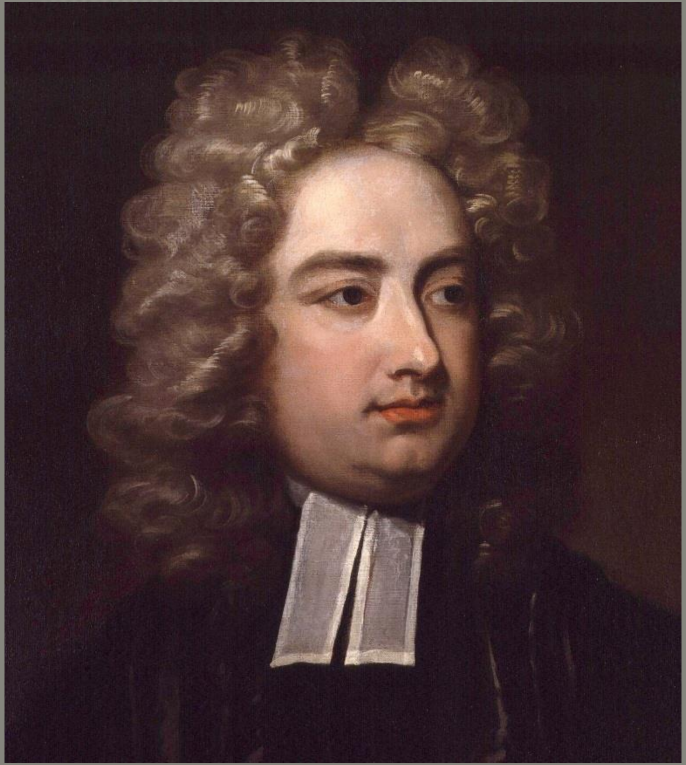
Disk Ops

Dropped
Samples

CPU usage

Variance

A Modest Proposal



Jonathan Swift

Questions?

https://github.com/imvu-open/istatd

@jwatte

jwatte@imvu.com