Indie DevOps and Analytics

Building big games with tiny teams.

John Bergman

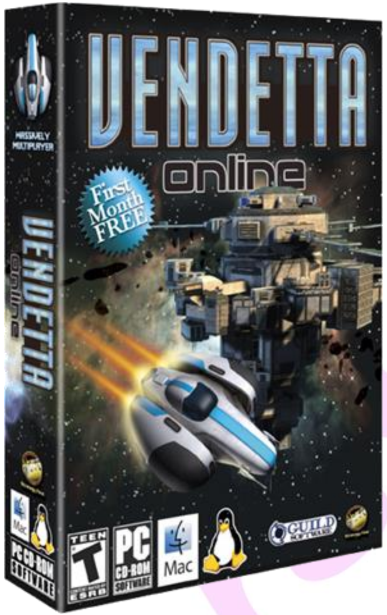
Founder, CEO – Guild Software

Who Am I?

- Used to build Big ISP Networks, Services.
- Started *Guild Software* circa 1998.
- Personally implemented most of the services I’m discussing today.

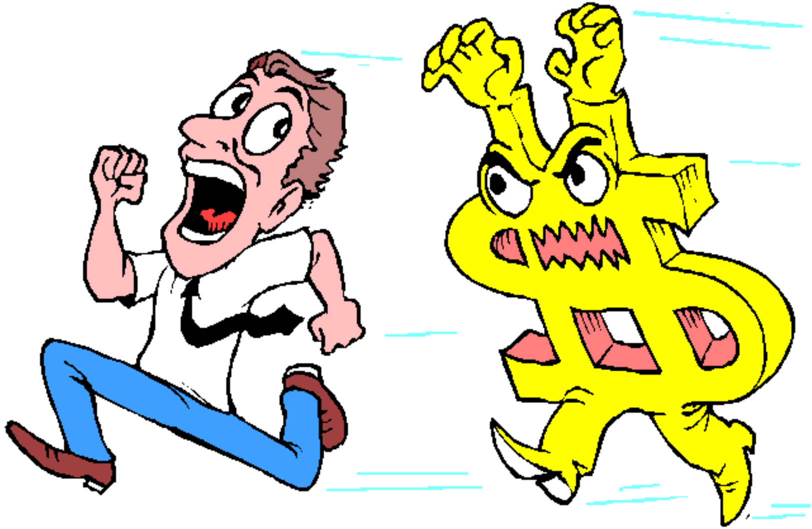
Who are We?

- *Vendetta Online* ships weekly updates across 5 major platforms.
- Yearly game-available target of 99.95%
- MMORPG has been online since 2002.
- Proprietary 3D and client/server engine.
- *Four-person dev team.*



So, you have a tiny Dev Team?

Cost vs Time



Types of Costs: Up-front vs Recurring

Up-front sucks immediately.

Recurring sucks.. ***FOREVER***.

Minimizing both Time and Cost

- If someone else will reliably manage the problem for free.. *Let them!*
  - *(But be aware of any long-term tradeoffs).*
- Reduces ***recurring*** maintenance time, security headaches, etc.

Free, Low-Cost Service Examples:

• Company Email	- Gmail
• Bulk Email	- Amazon SES, Sendy, Mandrill
• Monitoring/Webhooks	- UptimeRobot, StatusCake
• Client Metrics	- Flurry, Google Analytics, etc.
• Server Metrics	- New Relic, NodeQuery, etc.
• DNS (hosting)	- CloudFlare, Hurricane Electric
• DNS (GeoIP)	- Rage4, NSOne, etc.
• CDN	- CloudFlare, KeyCDN, MaxCDN

Minimizing Cost

- When outsourcing is too costly, DIY starts to look a lot better.

- But, always be aware of the ***recurring maintenance*** time-vs-cost tradeoff.

Server Infrastructure

Indie Server Scale Challenges

- Many F2P games require huge player bases to be financially successful.

- Huge player bases are a scalability problem.

- “Feature” transient loads can be large: 1 million new users per week.

Rules of Reduced Server Costs:

1. **Minimize** your per-node footprint (RAM, Disk, CPU cores).

2. **Building *fault-tolerance*** into your app architecture allows cheaper infrastructure.

3. **Faster** *server code* means more capacity per node.

Have an Optimization Plan

- **Plan** for compiled libs, JITs, something..?
- Keep in mind, C++ is ~600x faster than Ruby.
  - See *"Computer Language Benchmarks Game"*
- Capacity test for rough ideas of scalability.

Other Server Tradeoffs

- 32bit vs 64bit OS with VO: 40% ram savings, 10% CPU hit.

- FYI: Garbage Collecting VMs can blow up on long runtimes.

The Cloud

Cloud Competition is Intense

Type I - Largest Infrastructure

amazon web services™

Google Cloud Platform

Microsoft Azure

Type II – Enterprise Focus

SOFTLAYER®
an IBM Company

rackspace®

servercentral®

hp Cloud

Joyent

Type III – Public and Dev focus

DigitalOcean

aruba CLOUD

VULTR®

atlantic.net

linode

RamNode

Type IV – Mystery VPS Providers

(Riskier, but can sometimes be decent).

time4vps
BUILD YOUR OWN SERVER

HostUS

HostHatch

TANSTAAFL

- **Type I has great features.** Amazon DynamoDB, Route53, ElastiCache, etc.
- **Type II has more Managed options** for services.
- **Type III is inexpensive**, but more DIY oriented.

DigitalOcean vs Amazon

- ***Amazon bandwidth costs 4X more.***
- DO disk IO is ~4X faster (average).
- Amazon CPU/IO *may* be more consistent.
- DigitalOcean CPUs are just as fast in small instances as in large ones.

Ten Node Cost Comparison:

40 CPUs, ~80GB of ram, 800GB SSD, 10TB xfer

Amazon, c4.xlarge

- OnDemand: $1500/mo
- EBS 800GB SSD: $80/mo
- 10TB $0.09/GB: $921/mo

- **Total: $2501/mo**

DigitalOcean, 8GB/4cpu

- OnDemand: $800/mo
- SSD Disk: included
- 5TB included, +$102*
  - **(DO doesn't actually charge)*

- **Total: $902/mo**
  - **($800 current actual)*

Virtualization Tradeoffs

- ***Xen*** may exhibit timing instability (PLL).
- ***OpenVZ*** is Linux-only, “burstable ram” options, no kernel tweaks. Ram-efficient.
- ***VMWare*** has cool features, but can’t monitor Steal Time.
- ***KVM*** allows kernel tweaking, zRam, etc.

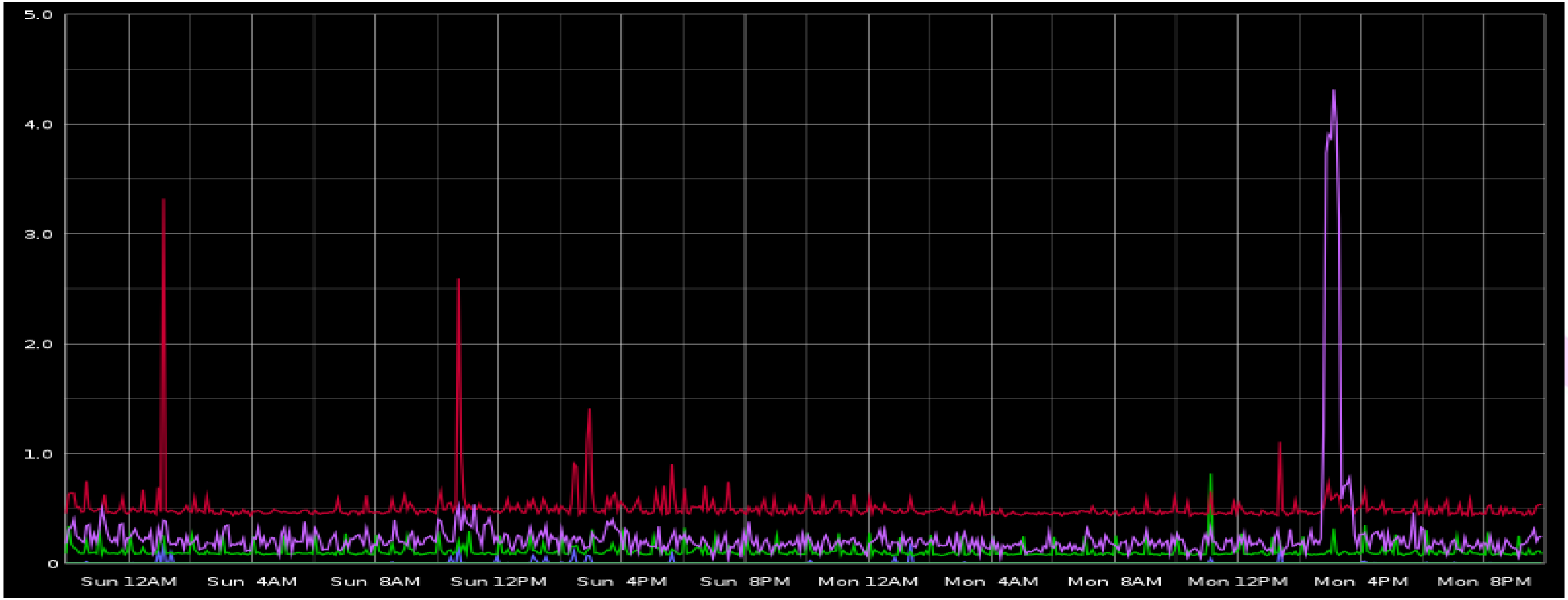
Providers by Hypervisor Type

- **Xen** - AWS, Linode, RackSpace, SoftLayer
- **KVM** - Google Compute, DO, Vultr, Altantic.net, RamNode
- **Hyper-V** - Azure, SoftLayer
- **VMWare** - ServerCentral, SoftLayer, Aruba
- **OpenVZ** - RamNode

Monitoring "*Steal*" Time

- "Steal" CPU time on KVM, OVZ, Xen: neighbors causing processes to wait.
- *Netflix 2011*: Dump the node and re-create.

Small 4% CPU spike in Steal

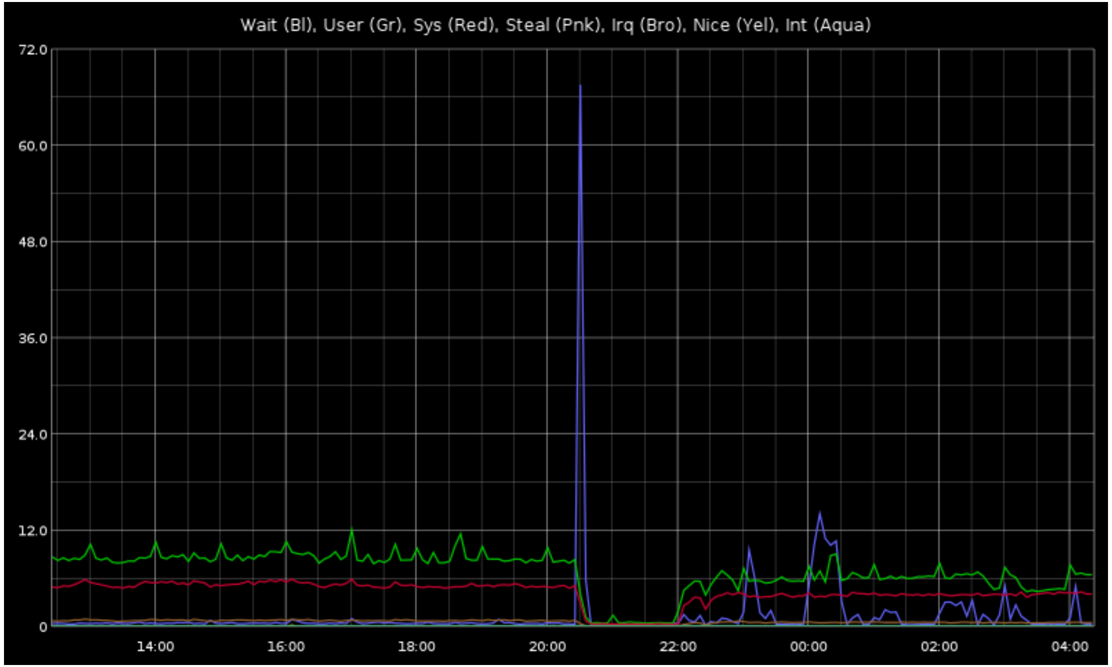


Monitoring “*IOwait*” Time

- IOwait (all hypervisors): degree to which processes are stalled on disk IO.

- May trigger from bad neigbors, backups, hypervisor bugs.

IOwait Example



- Cross-cluster IO stall.
- Stalled game for 1.5 minutes.
- Resulted in game outage.
- Drove peak load to ~5.5.

IOwait: Async Logging

- Don’t do classic “printf to file” logging in the cloud.
- IO-stalls will block your entire process.
- Instead, use syslog, or other async option.

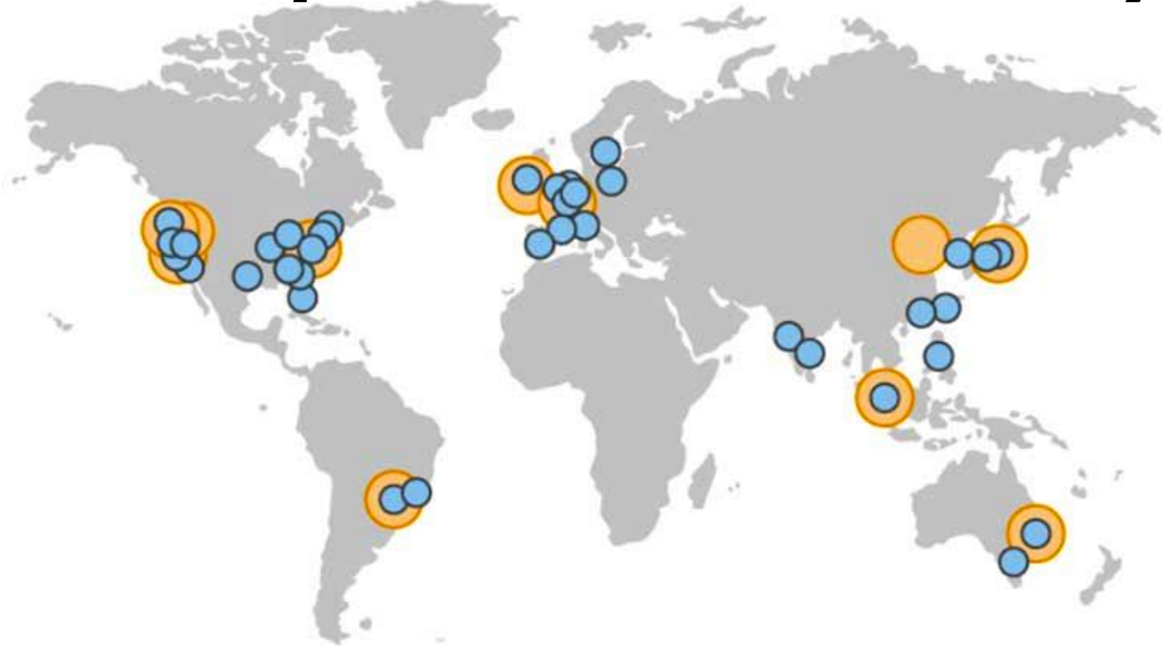
Don’t be a dick

- Target less than 50% CPU usage per node.
- Some providers will knock you off for over-use of CPU, I/O.

The Cloud Is Finite

Cloudburst Capacity:

What other providers are nearby?



Multiple Providers?

Choose a Widely Available OS variant.

Low-Cost Dedicated Hardware

- ***Kimsufi*** *(OVH)* – Xeon 8c, 16GB ECC, 100Mbp *unmetered*: $28/month.
- ***Scaleway*** – ARM 4c, 2GB ram, 50GB SSD, 200Mbp *unmetered*: €2.99/month.
- (100Mbp = ~32TB/month)
  - *32TB/month is $2785 from Amazon*
- But.. limited location options, etc.

Colocation *can* be Cheaper

- A full cabinet in the US is ~$1500/mo.
- Unmetered bandwidth by the gigabit is ~$1/meg ($0.003/GB/mo. *US, major network point).*
- Cheap off-lease cloud servers on Ebay.
- **Hardware can be a big hassle. But possible option *at scale*.**

Case Study:
Vendetta Online Patch Distribution.

Proprietary Delta-Patch Server Network

- Server cluster must be:
  - Globally Distributed
  - Good local-region bandwidth (tiny patches per user, but fast downloads, many users).
  - Resilient to outages (network/nodes/service).
  - Inexpensive!

Geo-Distribution Options

- **Anycast:** the "Right Way", but requires a big network and AS for BGP route advertisements. Used internally by Google, etc.
- **Amazon Route53 LBR** allocates by relative latency to an AWS DC.
- **GeoIP:** the "Cheap Way", works well with caveats. Used by Wikipedia, Akamai, many others.

VO “PatchStorm” Cluster

- Network of virtual nodes in US, Europe, Asia.
- GeoIP through Anycast DNS provider (Rage4).
- Client-side failover to backup cluster in ~5 seconds.
- Server status monitored by UptimeRobot
- Server-side down-node removal within ~600 seconds.

Resulting Performance

- Excellent localized proprietary TCP service in many regions (easily expanded).
- Many GB per day pushed at locally-fast speeds, minimal cost.
- Only took a few days to initially set up.
- Very fault tolerant: Node stability less critical.

Resulting Network

UptimeRobot monitoring:	Free!
Number of Servers:	12
Included Monthly Bandwidth:	~15TB
Separate Domain:	$15/year
Rage4 usage:	~$2/month
Total Cost:	***~$18/month***
Rough AWS Equiv:	*~$250 to $1300/month*

“Gotcha” notes

- *Weird routing:* Miami may be faster to Brazil than.. Brazil.
- Different VPS providers have various bandwidth overage policies. Read fine print.
- Large public DNS providers (Google, 8.8.8.8) may share cache across regions, breaking GeoIP for those users. *In practice: not a significant issue.*
- Professional CDN is probably easier/better for pure downloads, but not proprietary service.

Other Options

- Host GeoDNS yourself: PowerDNS, GeoIP back-end, rsync free MaxMind database weekly *(what Wikipedia does)*.
  - But, recurring time-cost not worth it.

Alternative Services

- The same fundamental architecture could probably serve many asynchronous games with GeoIP locality/performance.

Server Automation

Thou Shalt

- Automate *Everything*
- Monitor, Measure and Record Everything
- Alarm Everything

Automation..

- Is *critical* for small teams.
- Reduces administrative errors.
- Is necessary to let you scale quickly and elastically.

Which Automation?

- ***Salt*** vs ***Ansible*** vs ***Puppet*** vs ***Chef***
- Google uses Puppet, Yahoo uses Chef
- Puppet and Chef have more setup complexity to help large environments.
- Salt and Ansible are both simple and lightweight.

Ansible and Salt

- **Ansible** only requires SSH access (cool).
- **Salt** is another connection, but very fast.
- Salt-ssh replicates ssh-only automation.
- Upshot: Preferred the activity of the Salt community, and usage of Python.
- Use what you like?

Salt-Cloud

- Cross-API elastic node management in the cloud.
- Inherently speaks to AWS, RackSpace, DigitalOcean, Linode, others.
- Not hard to configure for other APIs.
- Not as full-featured as individual provider tools, but cross-platform.

Easy to add

- Salt only took a couple of days to learn.
- Fast to integrate new nodes.
- Secure: key-auth, AES on transport.
- Mixed with other tools (rsync), super helpful.
- Easy to script, program, etc.

Implementation Notes

- Differentiate and group server/node naming conventions by maintenance usage.
- Could be by service-type, or geography (DC), whatever is likely to be referenced.
- Shell-style globbing is used. ‘host*’ or ‘host[1-5]’ or ‘host[1,5]’, etc.
- We firewall the salt master. YMMV.

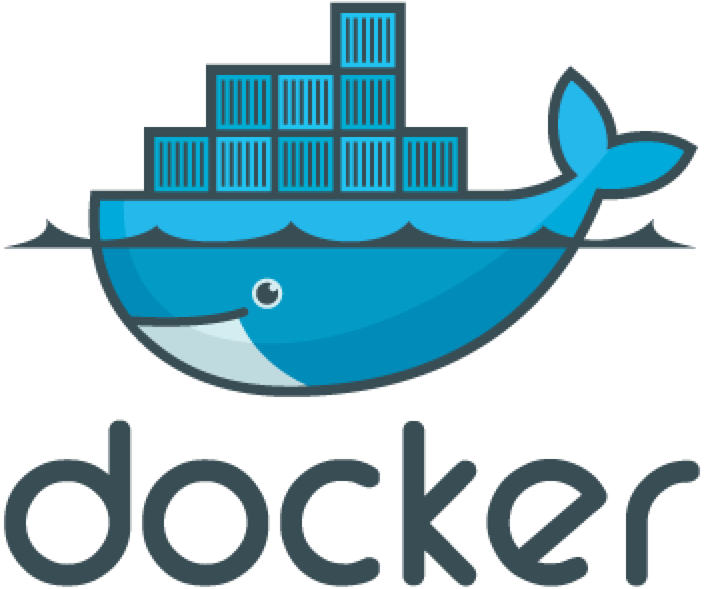
Obvious Example is Obvious

- Update an ipv4 firewall across all “core” servers:

```
salt-cp ‘core*' rules.v4 /etc/iptables

salt ‘core*' cmd.run 'iptables-restore <
/etc/iptables/rules.v4'
```

Automation, Releasing and Reverting



Docker: Dependency Sanity

- Package up your entire server app, binaries, and all dependencies into a single image.
- Distribute and run that package anywhere, identically, without virtualization overhead.
- Roll an updated image for each release.
- Revert with absolute certainty.
- Very elastically friendly. Easy to spin up dev nodes.

Docker: Caveats

- Image size may directly impact distribution time and elastic startup delay.
- A repository can be run locally in a DC
- Or I/O intensive data could be acquired via other channels.
- *64bit Only* at present.

Docker: Try services easily

- There is an existing Docker image for a great many services. Ready to roll.
- Don’t want to configure Graphite, Apache, Statsd, Grafana, to all play together..?
- Install a Docker image in one command.

Automated Client Testing

- Very helpful, but not a panacea.
- Open-source frameworks: *Appium*, *Calabash*, etc.
- Cloud-based devices: *TestDroid*, *Xamarin*, *AWS Device Farm*, $0.17/min.

Monitoring, Analytics

Monitoring, Stress-Testing

- Build a headless test-client!
- Use meaningful player behaviour to test/alarm “server-online” status.
- Re-use the same test-client to implement a server stress-test, prior to launch.

Automated Bug Reporting

- We have crash reporting and a backtrace system on both the clients and server.
- Makes reaction to bugs much faster, more accurate.
- Simplest client-side implementation:
  - Write out critical data on crash.
  - On startup, detect the file very early.
  - Submit via HTTPS on next runtime.

Metrics & Analytics

- Absolutely critical! No excuse to not have *something*.
- If you have *ZERO* time/budget, then:
    - Implement Flurry on the client-side.
    - New Relic for server monitoring.
    - Completely free, to any scale, with good basic data.

Out-Sourced Analytics: Benefits

- Many options: Localytics, Mixpanel, New Relic, etc.
- Near-zero upfront cost (time or money), should “just work”.
- Lots of intelligent defaults for common cases.
- “Free” usage tiers of meaningful scale.

Out-Sourced Analytics: Flipside

- Configured for common-case, not “You”.
- Varying degrees of flexibility.
- Anything custom can be expensive.
- Data resolution can be poor (or expensive).
- High usage loads can become costly.
- Third-party SDK/library may break your app.

For Example..

- Metrics via Mixpanel: 20 million data points per month, $2000/month.
- Vendetta Online server cloud: 260 million data points per month. Cost: a few days of setup, 42kbits of bandwidth, 350MB of disk space.
- BUT, not a 1:1 comparison!

Graphite is Awesome!

- Zero setup time for new metrics.
- Graphing detached from storage (Carbon).
- Many third-party front-ends.
- Easy to tie in log aggregation, data-mining.
- Monitoring/Alarms on time/series data changes.
- Lots of integrated math functions, etc.

Carbon stores Data

- "Whisper" time/series database.
- Fixed size, determined by aggregation type.
- Aggregation chosen by regex match, on metric initialization.
- Data precision can be as high as per-second, useful for server profiling.
- Average/Sum/Last/Min/Max options for aggregation.

Creating a Metric is Trivial

- Any metric can be generated by sending this, to a carbon server:

```
<metric path> <metric value> <metric timestamp>
```

Server Storage Aggregation

- Server data: A reasonable picture of recent activity, 46kB of disk per metric. BUT, there may be many metrics. Roughly 320 per server, or ~15MB:

```
[servers]
pattern = ^servers.*
retentions = 60s:4h,5m:2d,30m:1w,2h:30d,4h:90d,1d:5y
```

Revenue Storage Aggregation

- Revenue data, summed and not averaged, kept at higher accuracy. 112kB per metric:

```
[revenue]
pattern = ^revenue.*
retentions = 5m:14d,30m:4w,2h:90d,4h:180d,1d:5y
```

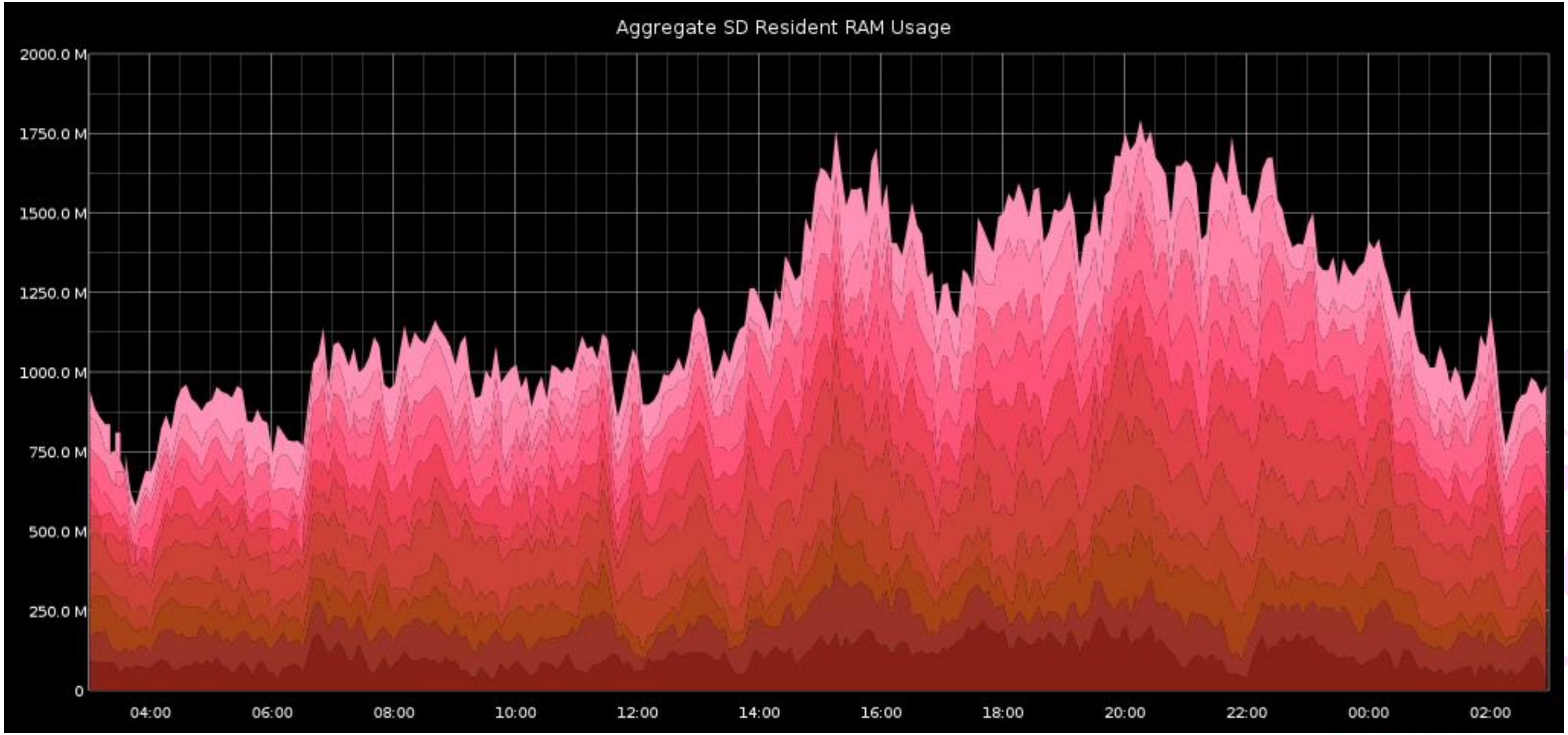
Doing this:

```
echo "servers.vo-sc.`hostname`.sd.totalmemory" `ps -axm |
grep "\.\/bin\/sd" | sed "1 d" | awk '{print $9 * 1024}'
| paste -sd+ - | bc` `date +%s` | nc -q5
redacted.hostname.com 2003;
```

Sends this output:

```
servers.vo-sc.vocll.sd.totalmemory 111009792 1445666511
```

And results in:

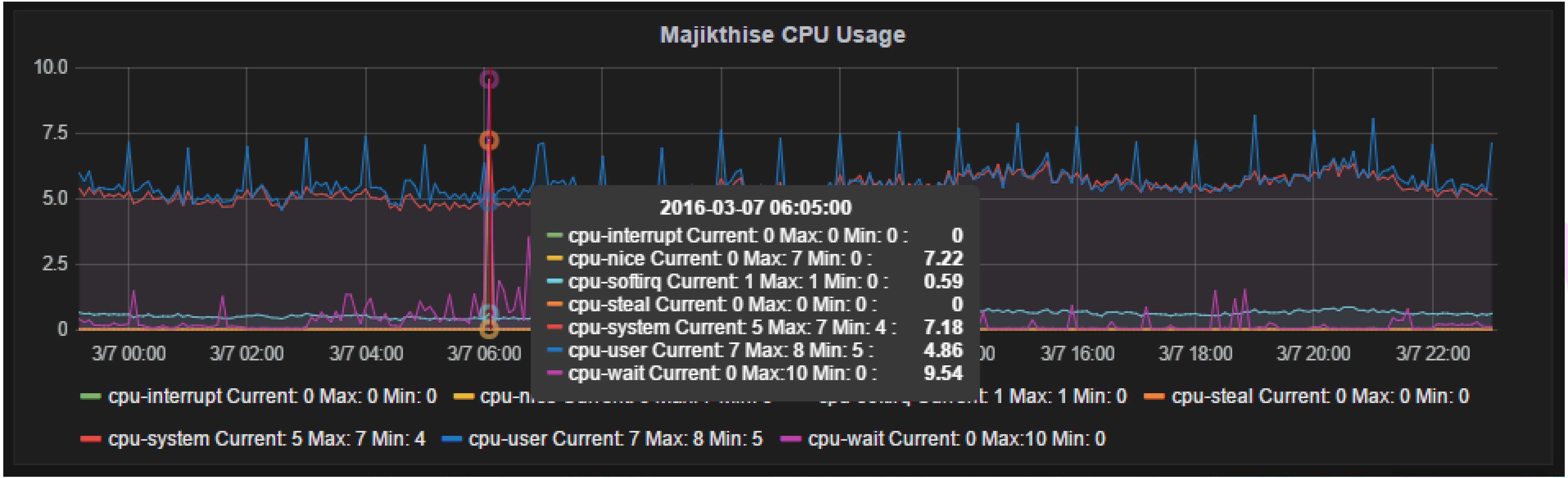


Graphite Renders Carbon data

- Can be hosted separately from Carbon.
- Default front-end, graphing, dashboard.
- Stores graph-config data in sqlite (read-side scale issue for some).
- Far better third-party options, **Grafana** uses d3.js and lets you zoom/scale data in realtime.

Grafana Example

Data is easily zoomed, scaled, measured.



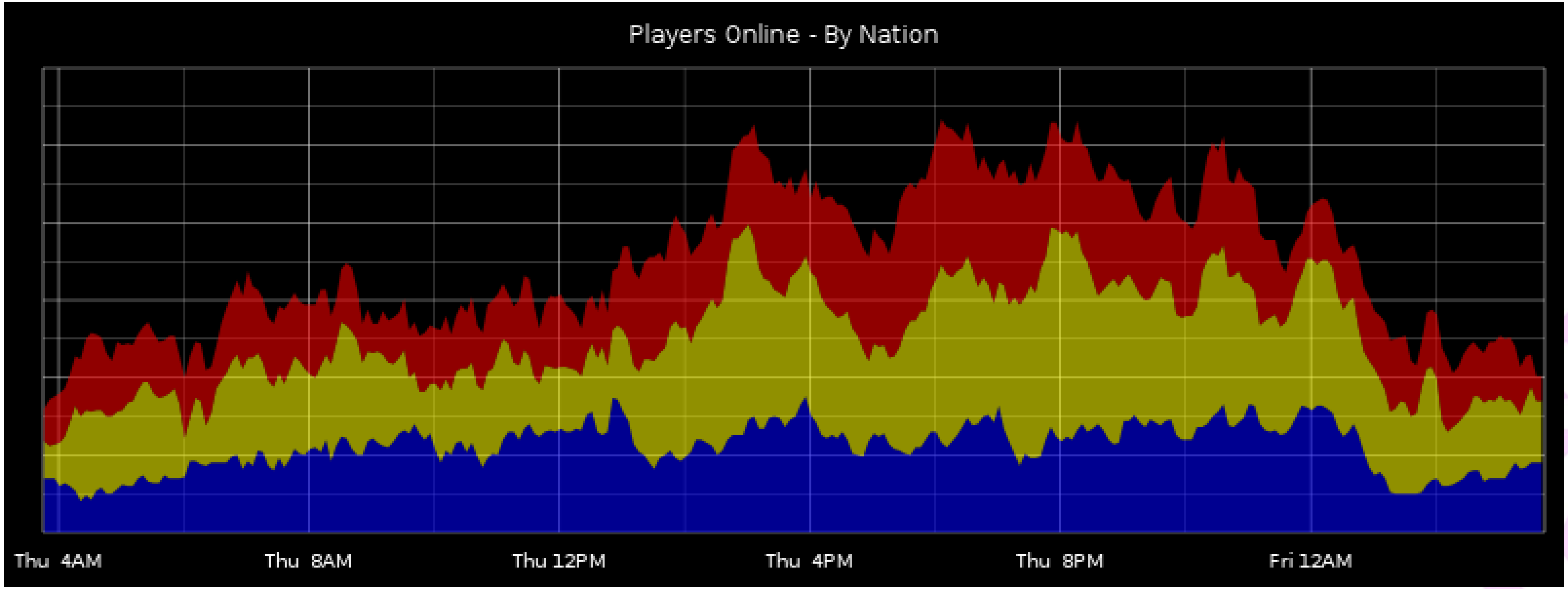
2016-03-07 06:05:00

cpu-interrupt Current: 0 Max: 0 Min: 0 :	0
cpu-nice Current: 0 Max: 7 Min: 0 :	7.22
cpu-softirq Current: 1 Max: 1 Min: 0 :	0.59
cpu-steal Current: 0 Max: 0 Min: 0 :	0
cpu-system Current: 5 Max: 7 Min: 4 :	7.18
cpu-user Current: 7 Max: 8 Min: 5 :	4.86
cpu-wait Current: 0 Max:10 Min: 0 :	9.54

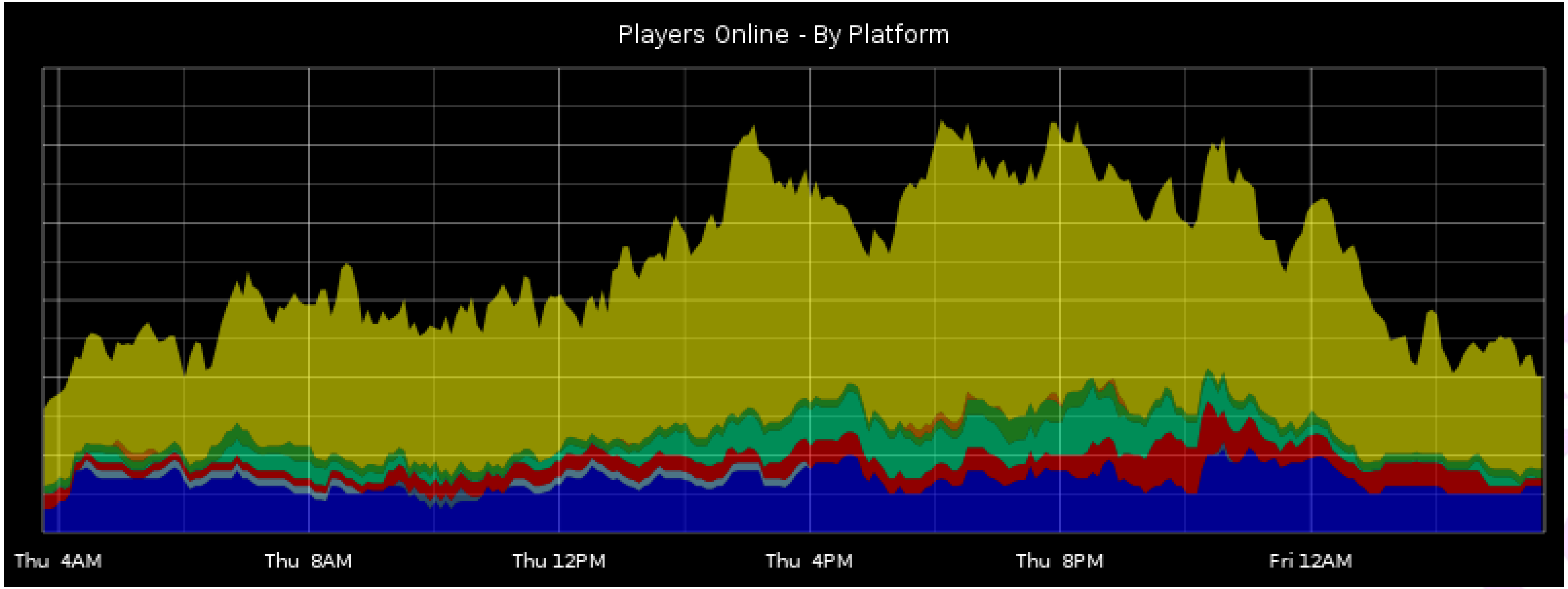
Combine Metrics Arbitrarily

- Cross-combine and mix any metric, or metrics, with any others.
- For instance, even simple dashboards..

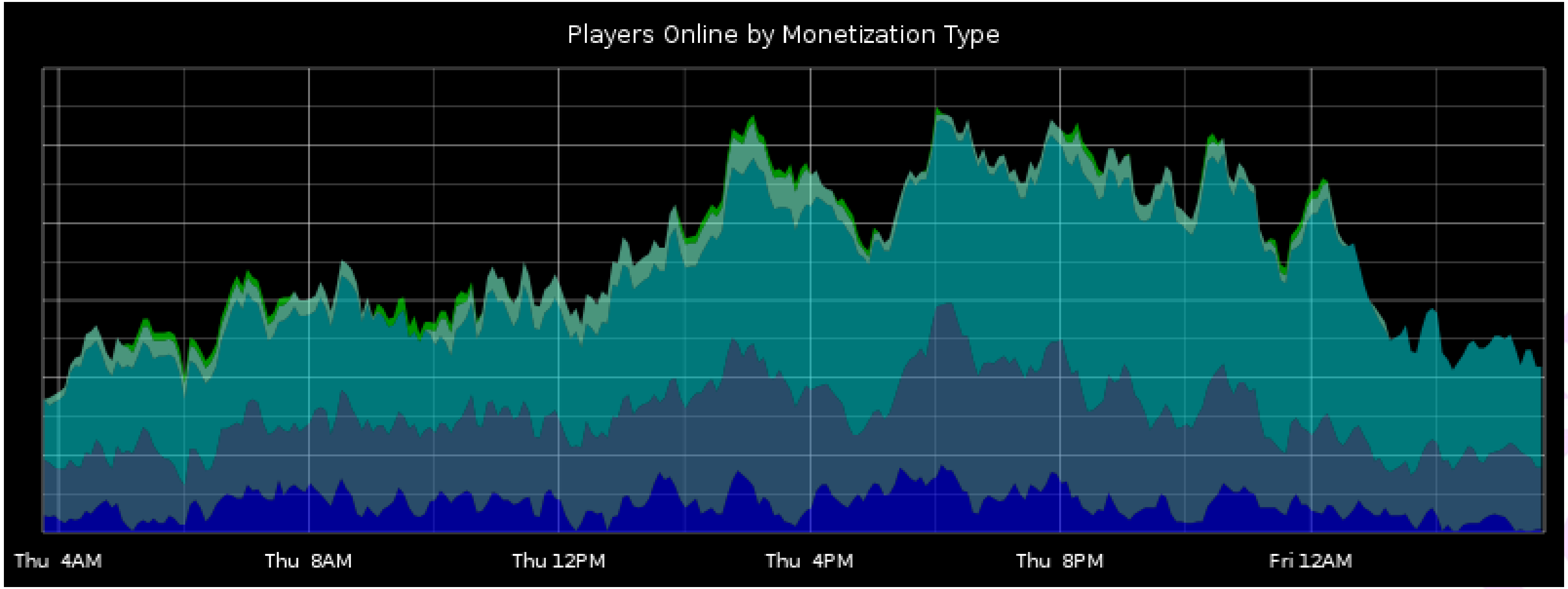
Active Players – By Faction



Active Players – By Platform



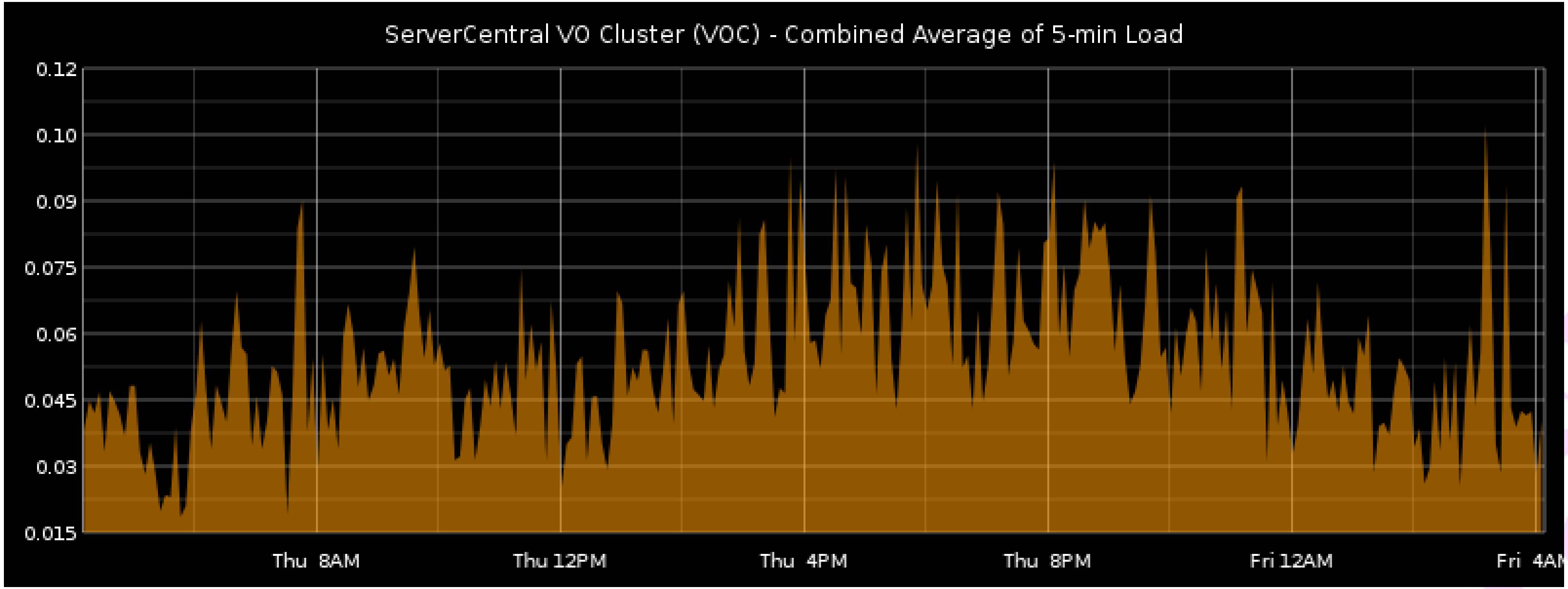
Active Players – By Monetization



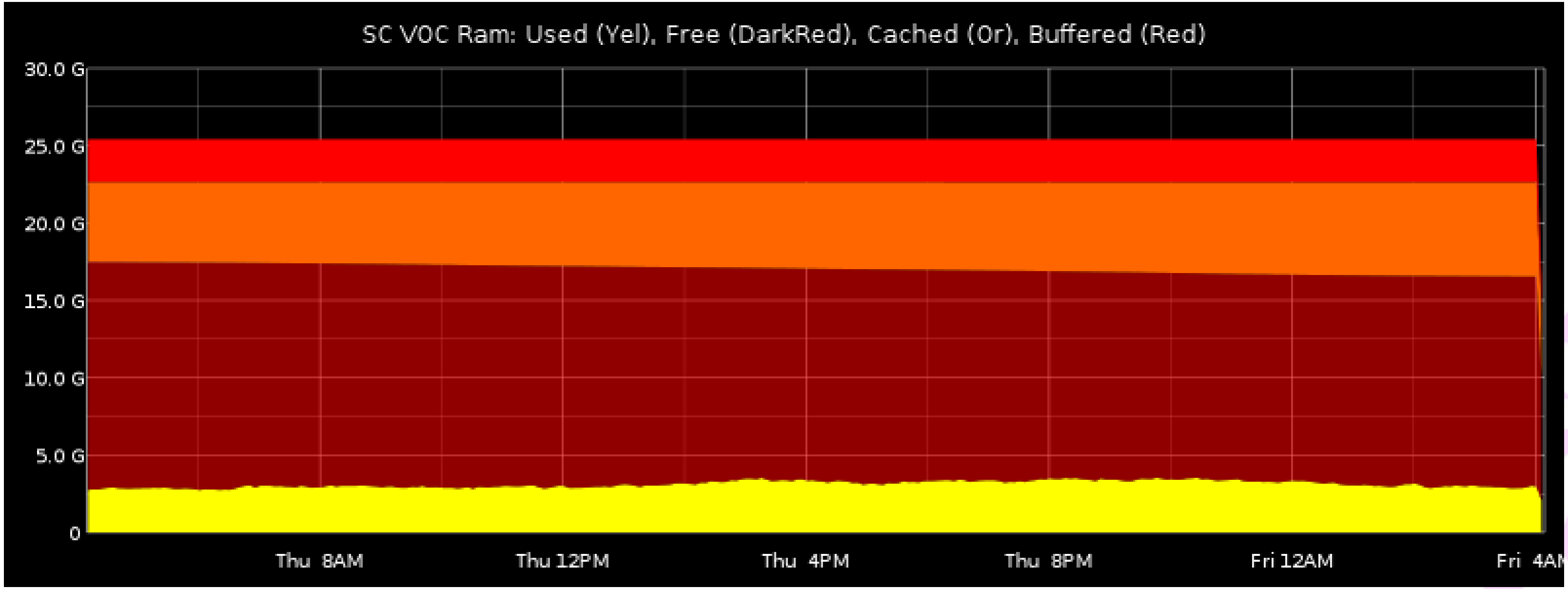
Aggregating Cluster Data

- Rapidly get a picture of combined “server weather” status.

5-min Load, Averaged per DC



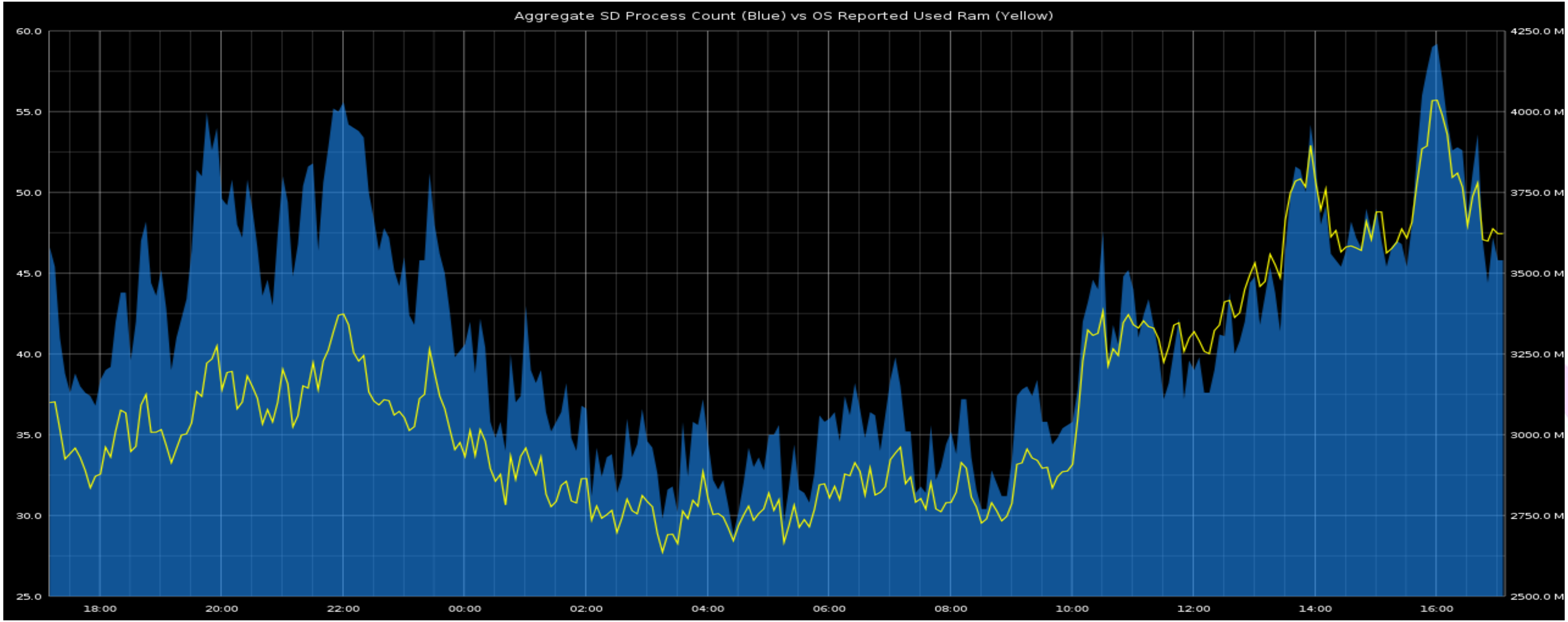
Summed Ram usage across a DC



Correlate Disconnected Data

- Blend data from related, but disconnected inputs to gain insights.
- Separate Y-axes (left vs right) allow unique scales.

Process Count vs RAM footprint



Easily profile rapid changes.

- Server was exhibiting sporadic disk IO latency spikes.

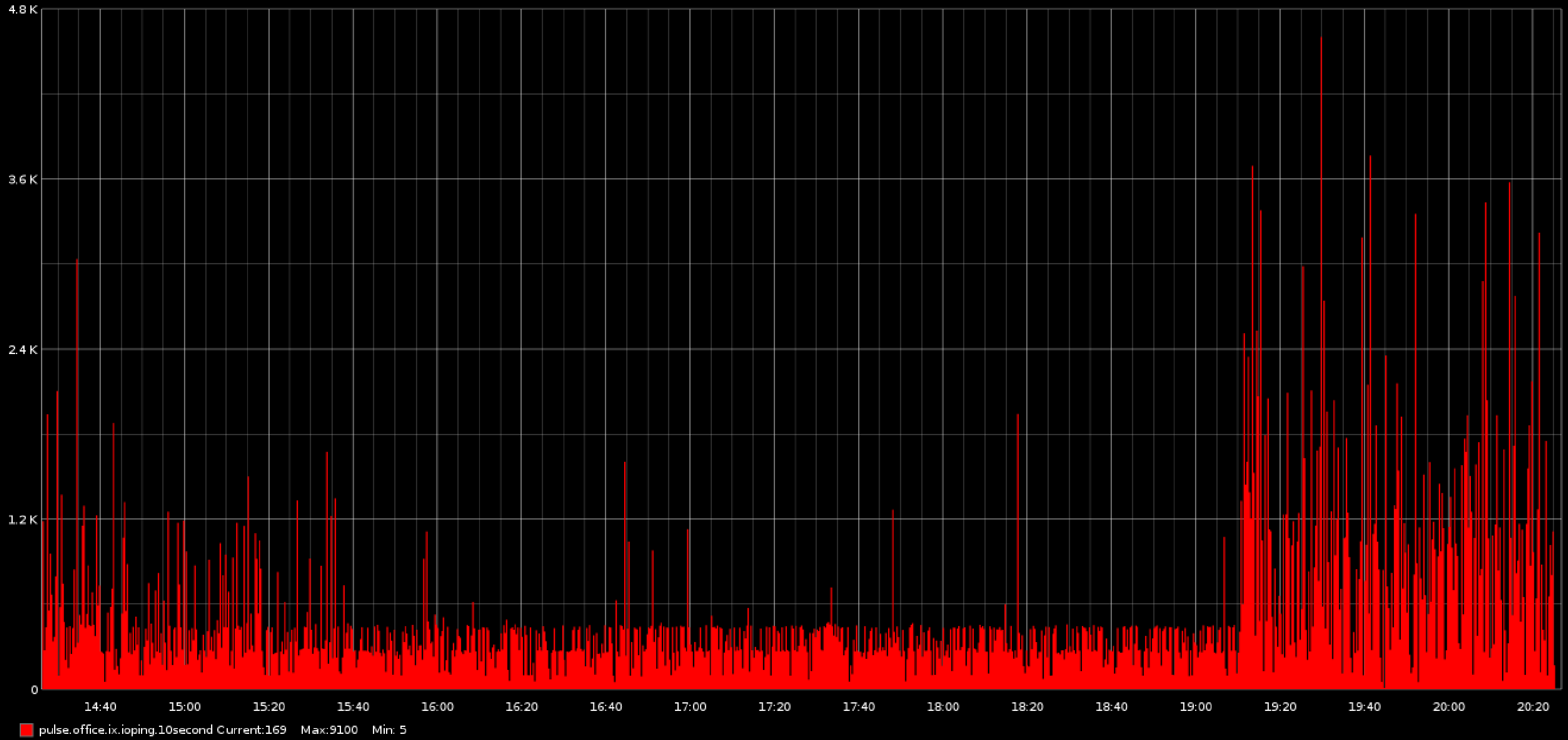
- A simple script, with “ioping” greatly helped analysis.

Simple Script, results in..

```
#!/bin/sh

while [ 1 ]
do
        date=`date +%s`
        echo "pulse.office.`hostname`.ioping.10second.max" `ioping -c 1 -q -p 1 . | \
        awk '{ print $2 }'` "$date" | nc -q 600 redacted.hostname.com 2003 &
        sleep 10
done
```

Ix disk I/O latency (microseconds)



Elastic? Construct on the fly.

- All graphs are simple URLs with parameters (or csv/json/etc).
- Trivial to add and remove elastic nodes.
- Graph by percentage-of-total instead of summed aggregate, etc.

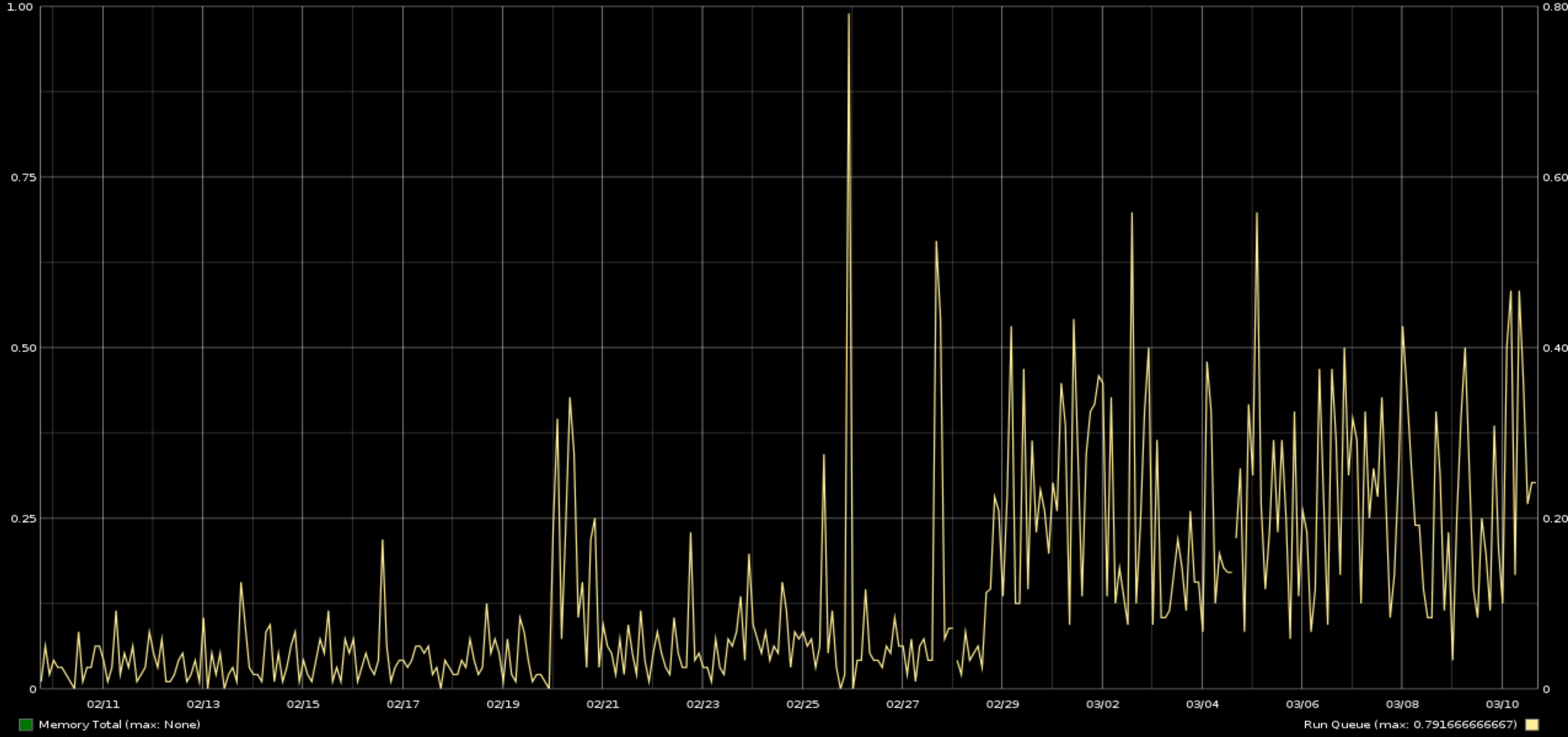
Combined Game Metrics

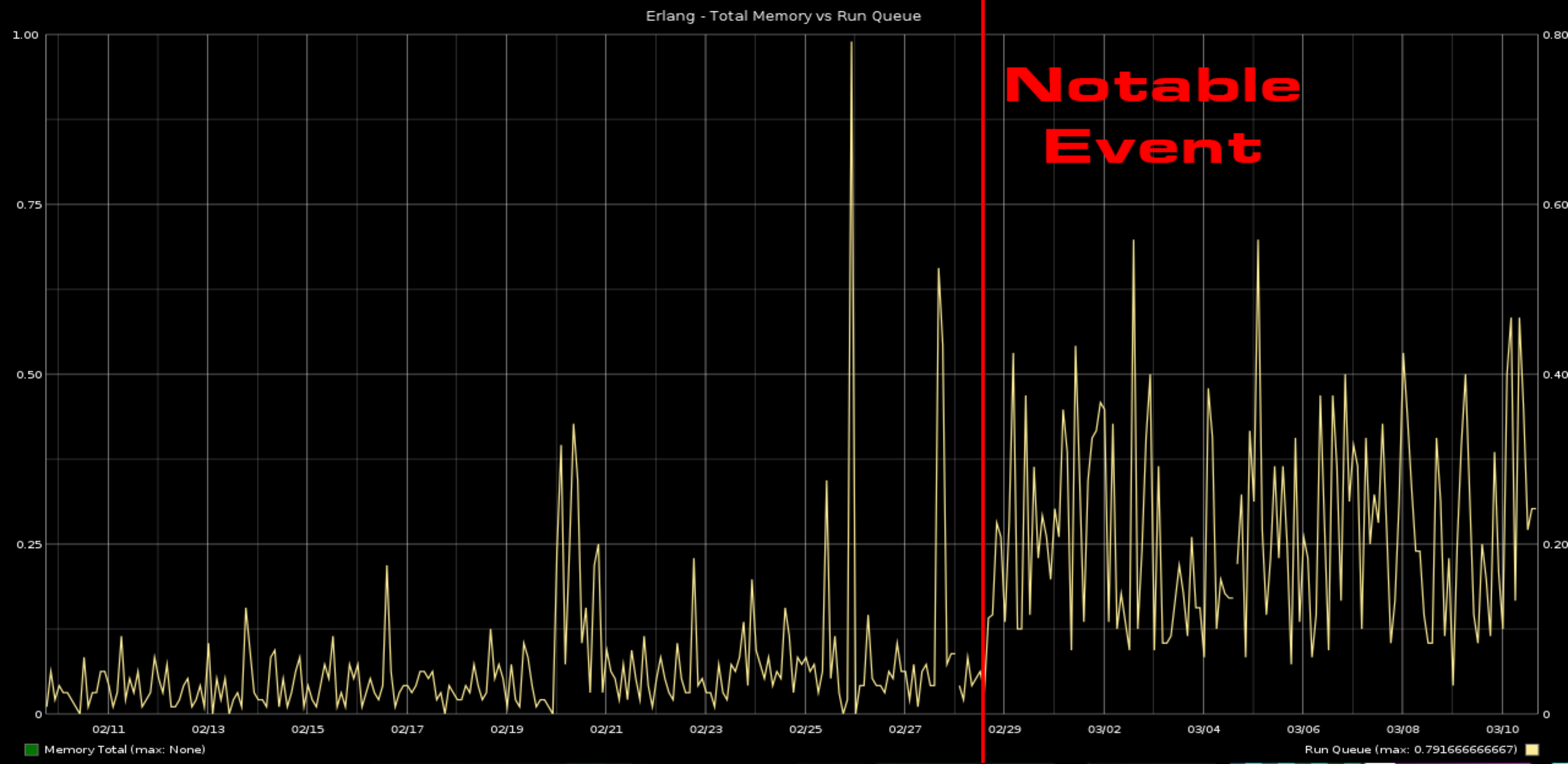
- We record a lot of metrics per player.
- Thus, we can later combine them to get..
  - Monetization of AppleTV vs AndroidTV vs Xbox users.
  - Percentage of VR players who use gamepad vs mouse/keyboard.
  - PvP success of mobile vs PC players.
  - Whatever else we can imagine, and record.

DrawAsInfinite

- Allows a vertical line to be set at a specific time.
- Helps correlate non-numerical events. Ie:
  - See spike in item sales vs an item stat change.
  - Players online with start/end value of a major guild event.
  - Instances of opened support tickets vs recorded latency.

Erlang - Total Memory vs Run Queue





Notable Event

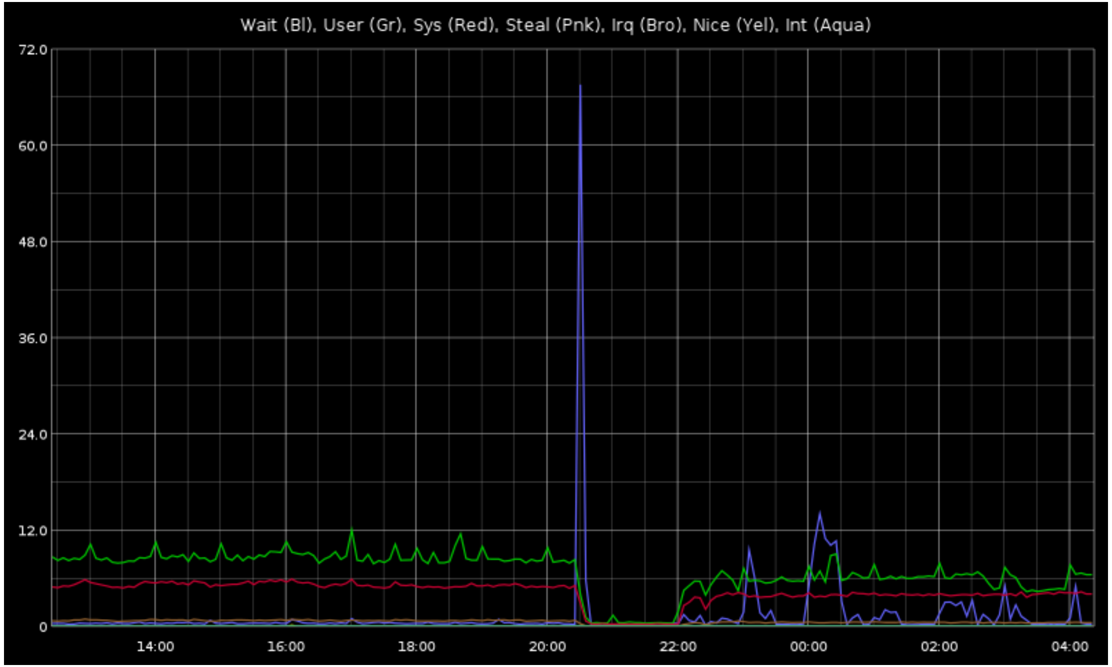
System Monitoring

- Diamond: Effective, "pickled" protocol, but big ram footprint (Python).
- CollectD: Also effective, very fast, low ram overhead.
- Diamond: ~40MB resident, CollectD ~5MB resident.
- Various other options out there, graphite being popular.

CollectD Specifics

- Avoid installing kitchen sink: collectd-core is probably enough.
- Plugin options like "tail" and "exec" offer customization.
- "Aggregation" plugin is convenient for many-core systems, etc.

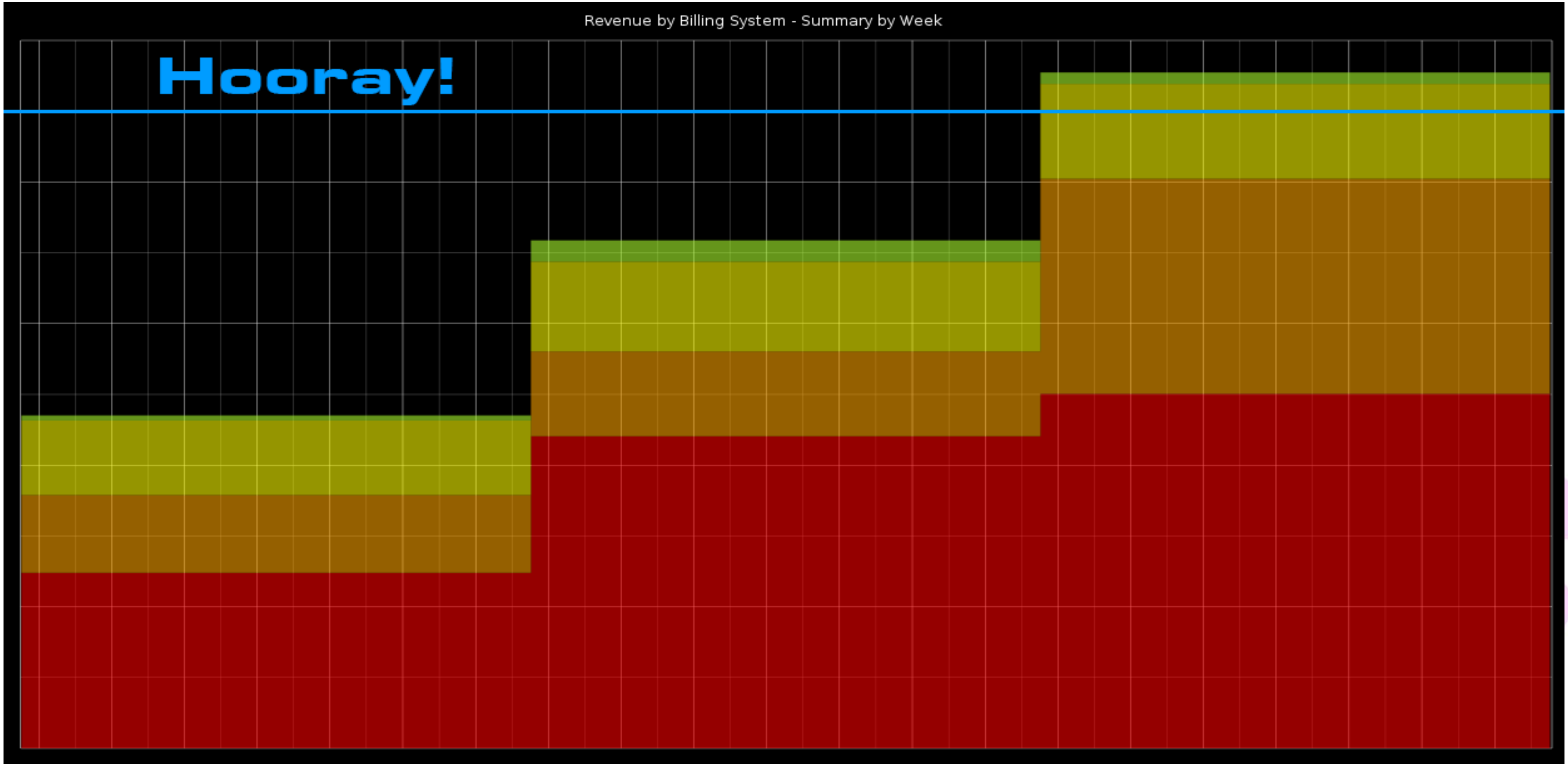
IOwait Example.. Again!



- Remember this?
- Wrong cause was initially suspected.
- Graph drastically narrowed debugging scope.
- Huge time-save.
- Aggregation simplifies output.

Monitoring Metrics

- Deltas and Thresholds can be monitored and alarmed.
- Alarms could notify of any trend, even an increase in revenue:


Revenue by Billing System - Summary by Week
Hooray!

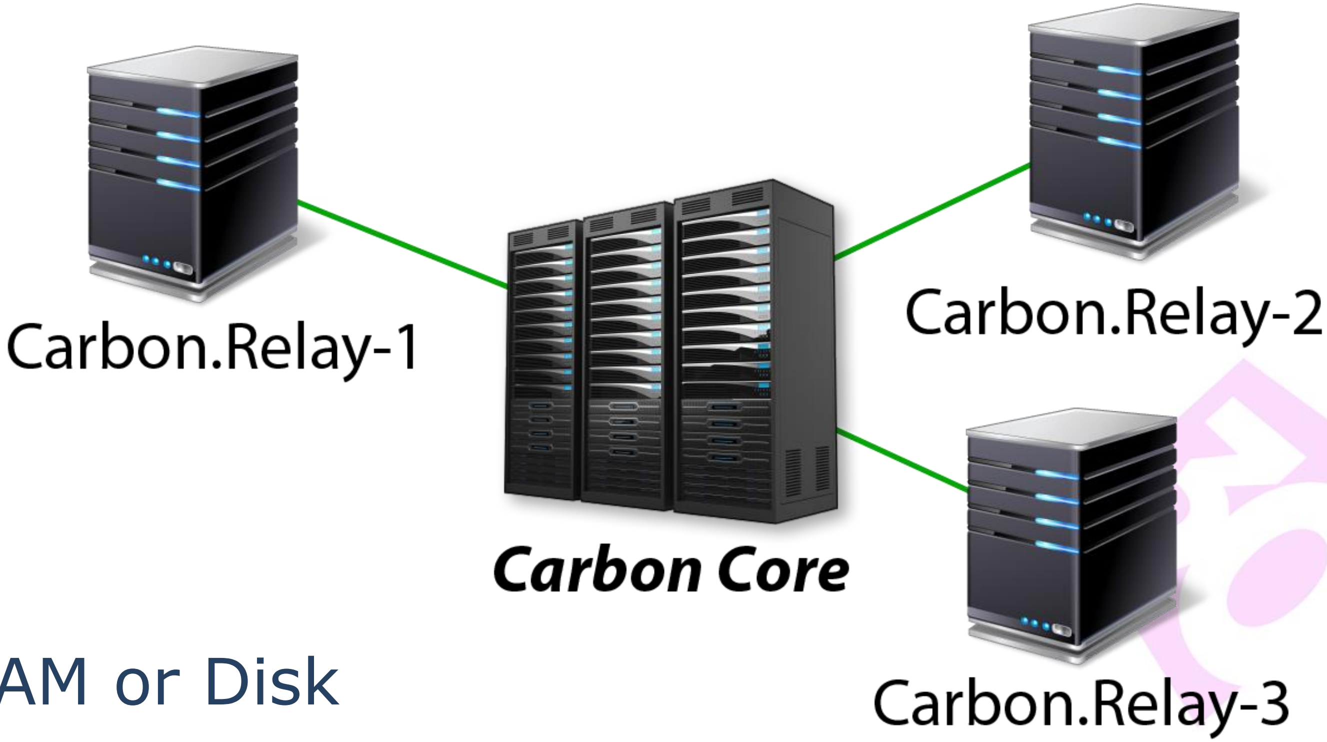
Monitoring / Alarm Solutions

- ***Cabot*** provides means of triggering by delta, etc.
- ***graphite-beacon:*** simple python script for alerts by data queries into carbon.
- ***Possible:*** Skyline *(Etsy)*, AnomalyDetection *(Twitter)*, Anomalizer, modified Monit, etc.
- Fire off email, or hit an SMS gateway, etc.

Recording Client Metrics

- **Graphite can be used end-to-end**, receiving any data from clients just as easily (scale challenges apply).
- **Client performance metrics** are useful: Time from App-start to “Fun”. What takes the most time?
  - Stacked graph of startup, DNS resolution, patch processing, texture loading, etc.

Scaling with *Carbon Relay*



- Aggregates Connections
- Buffers to RAM or Disk

Carbon-Relay Alternatives

- ***BackStop:*** Relays from JSON via HTTP POST to Carbon.
- ***Carbon-c-relay:*** Compiled C, fast with back-end failover.
- ***Carbon-relay-ng:*** Compiled C, upwards of a million metrics / second.
- And still more..

Log Data Tailing

- ***Syslog-ng***, ***Logster***, ***CollectD*** all have integrated Graphite/Carbon support.
- Tail logs, automatically return metric data that appears.
- Add new "metrics" with an additional log-tail/regex.

More Complex Options

- **MANY ways to do things:** *Uber* vs *Etsy* vs *Instagram*, etc.
- ***Statsd*** solutions can send data to Databases, other than Carbon.
- ***Ganglia***, ***ElasticSearch***, other solutions can wire in to augment Graphite, log aggregation, data-diving.

What We Do

- "Server/OS" metrics reported independently by CollectD.
- "Game" metrics aggregated and reported by Erlang subsystem (“estatsd”).
- All fire metrics into carbon-relay for each DC, pickled/aggregated back to core metric server.

Conclusion

Game Server *Performance* drives scalability and costs.

Automate control of your clusters, at any elastic scale.

Server Update and Revert/Rollback should be simple, foolproof, free of dependency issues.

Measure, Monitor and Alarm Everything!

Awareness of recent history shines a spotlight on problems, saving huge debugging time, and downtime.

Questions?

- I will attempt to answer questions, and may just ramble aimlessly.

- Contact me, for electronic rambling!

john@guildsoftware.com

GUILD SOFTWARE