Measuring Distributed Databases across the Globe

Matt Davis
Site Reliability Engineer, OpenX
SCaLE x13, 2015

what is a measure…?

Measure

- *Formal rule* that helps assess relationships
- *Quantity* of a substance
- *Unit of time* defining a collection of beats or events
- *Dimensions and capacity* of a given thing

In distributed systems we try to perfect the **rules** by which we store, process, and deliver mass **quantities** of data. We are solving puzzles, estimating **capacity**, and maintaining *structure* all at once, while workloads and use cases evolve over **time**.

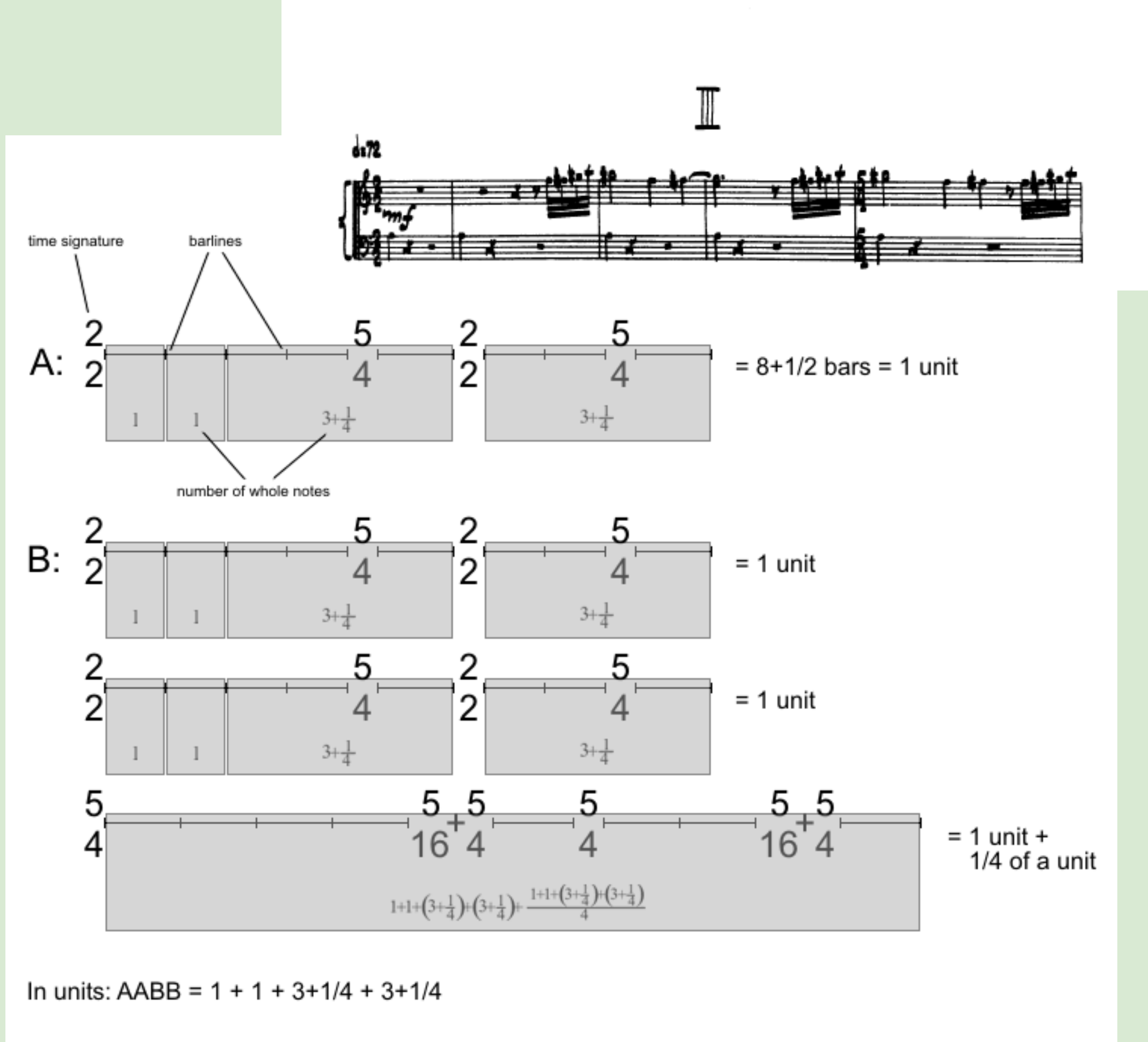
John Cage (1912-1992)

Sonatas and Interludes for Prepared Piano

“micro/macro-cosmic” method placed importance on **rhythmic structure** over harmony and melody

in computer science these rhythms and waveforms are in evidence all the time

*become familiar with the **structure of rhythmic patterns** in data feedback, it will give important clues to how your distributed ecosystem is behaving!*



aaah, measure is evidence of structure!

Like a musical improviser learning scales and beats and time signatures, the system operator must become aware of inherent real-time relationships.

Well-placed ***measures*** *help the admin internalize how data flows through the system, illuminating the* ***structures*** *of both architectural and operational rhythms.*

Ad exchange (including real-time bidding), publisher monetization (SSP), and ad server all combine to enable over a billion daily ad impressions across the US, Europe and Asia.

OpenX

In terms of our distributed data, this means...

- Combined gateways measure over 400,000 connections/sec at peak
- Over 6PB across all US Hadoop clusters
- 5000+ physical devices between 5 datacenters
- Reporting data totaling over 133TB
- Over 40 billion unique keys between five differently sized and variously connected Riak Enterprise and CS clusters with hundreds of nodes spread between Asia, Europe, and US datacenters.

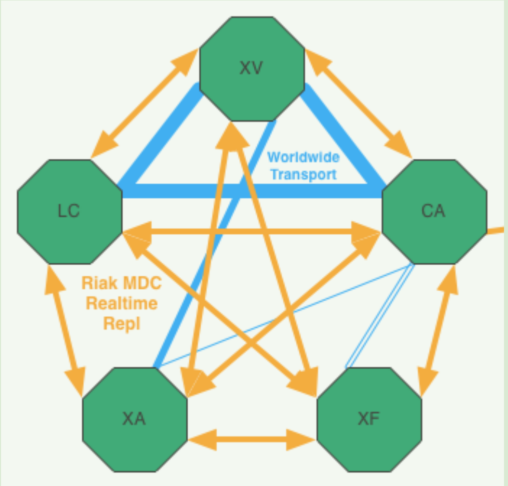
cloude

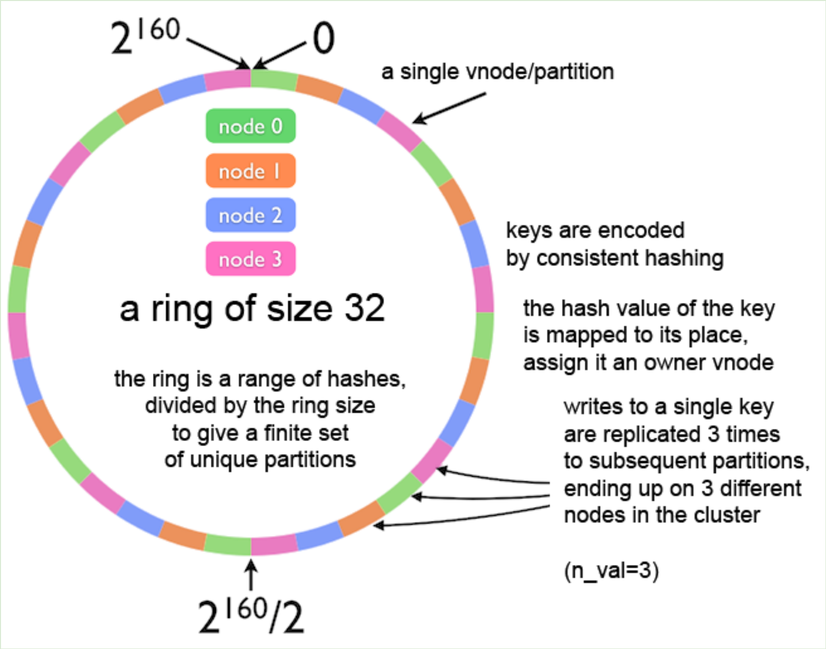
Erlang

VERTICA

Distributed Data at OpenX

***Riak** is a highly available, distributed key/value store.*





globally connected riak clusters provide realtime stores to front-end services

mandala

मण्डल

Maṇḍala

circle

a spiritual and ritual symbol representing the Universe

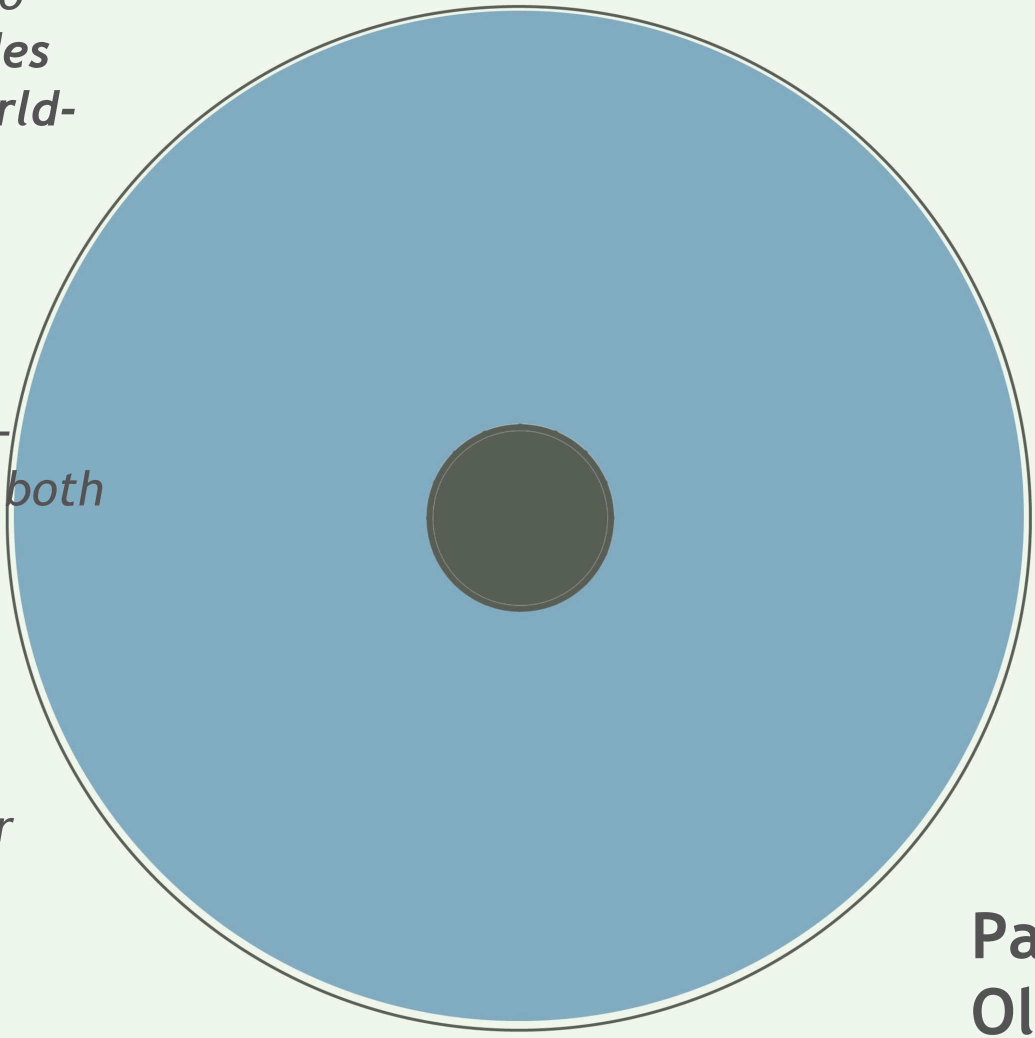
Mandalas often exhibit radial balance; elements are arranged so that no one part seems heavier than any other part.

the well distributed system is also balanced, where all parts are matched and behaving as one.

*Like the Cage sonata, relationships of **client** to **code** to **data** to **partitions** to **nodes** to **clusters** to **world-wide mesh** are complex and overlapping.*

micro- and macro-cosmic, requiring both attention and awareness...

...we listen to our machines



attention

+

awareness

Pauline Oliveros (1932)

Deep Listening

Monitoring:

*the art of staying **attentive** and being **aware***

- ❖ **Instrumentation** of OS & application statistics
- ❖ **Visualization** of OS and hardware health
- ❖ **Aggregation** of stats and logs, OS & application

and here's the bonus!

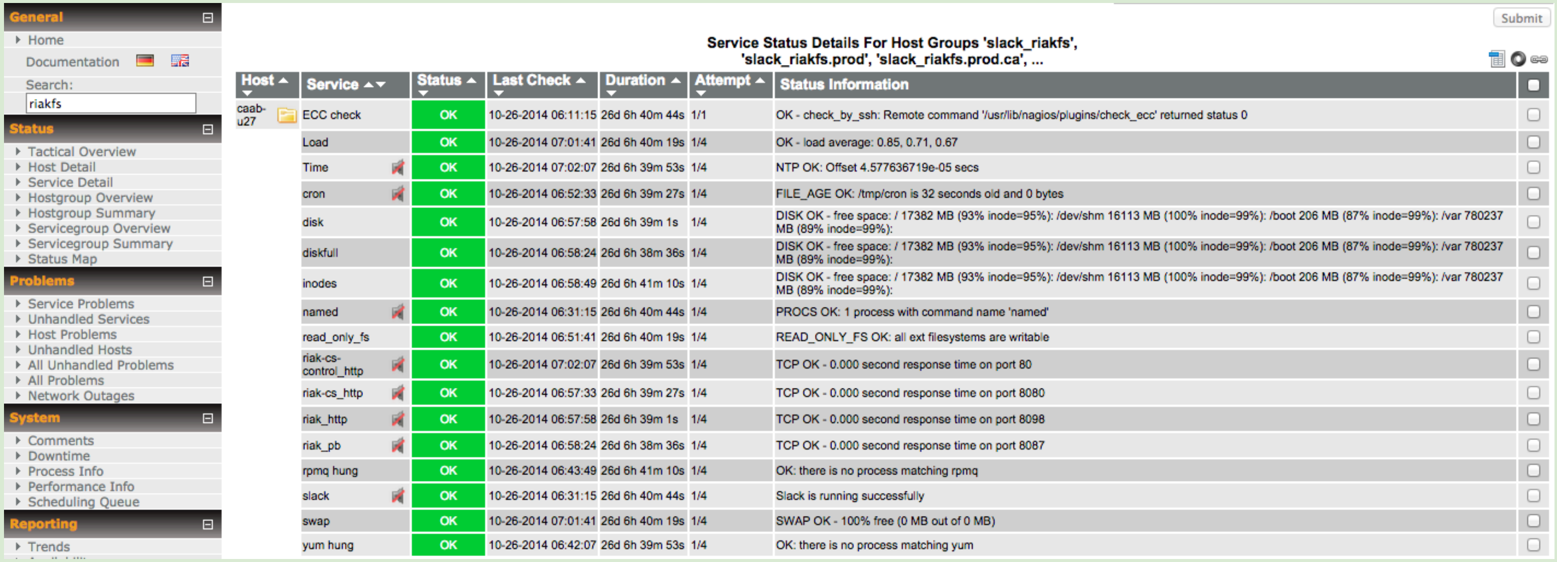
*all contribute to **intelligently documented procedures***

essential for NOC and oncall operations

Instrumentation:
Icinga

The name Icinga is a Zulu word meaning "it looks for", "it browses" or "it examines" and is pronounced with a click consonant. It is a fork of the popular Nagios system.

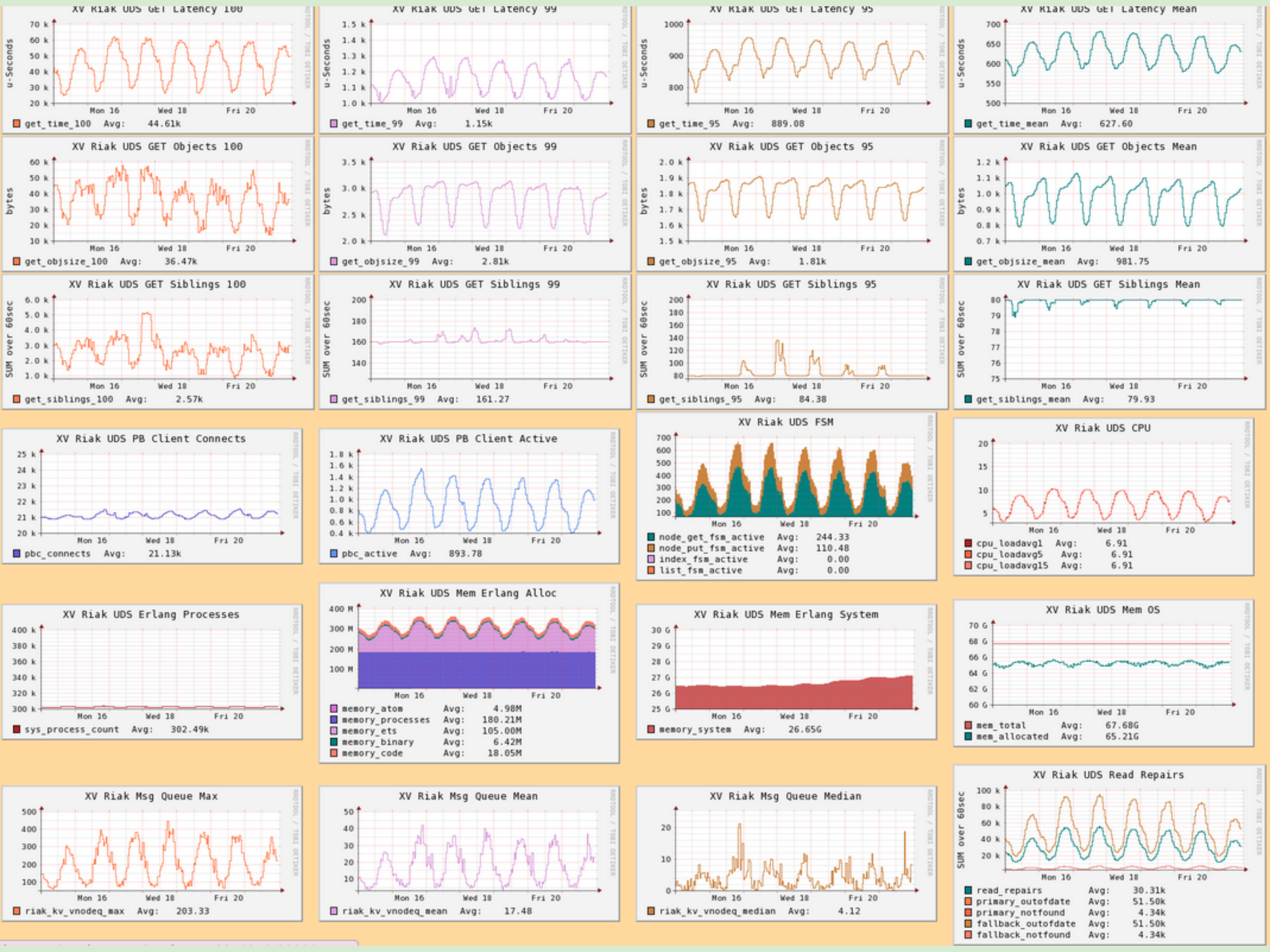
- ➔ *system resource monitoring*
- ➔ *application endpoint health*
- ➔ *alert history and histograms*



Host	Service	Status	Last Check	Duration	Attempt	Status Information
caab-u27	ECC check	OK	10-26-2014 06:11:15	26d 6h 40m 44s	1/1	OK - check_by_ssh: Remote command '/usr/lib/nagios/plugins/check_ecc' returned status 0
	Load	OK	10-26-2014 07:01:41	26d 6h 40m 19s	1/4	OK - load average: 0.85, 0.71, 0.67
	Time	OK	10-26-2014 07:02:07	26d 6h 39m 53s	1/4	NTP OK: Offset 4.577636719e-05 secs
	cron	OK	10-26-2014 06:52:33	26d 6h 39m 27s	1/4	FILE_AGE OK: /tmp/cron is 32 seconds old and 0 bytes
	disk	OK	10-26-2014 06:57:58	26d 6h 39m 1s	1/4	DISK OK - free space: / 17382 MB (93% inode=95%): /dev/shm 16113 MB (100% inode=99%): /boot 206 MB (87% inode=99%): /var 780237 MB (89% inode=99%):
	diskfull	OK	10-26-2014 06:58:24	26d 6h 38m 36s	1/4	DISK OK - free space: / 17382 MB (93% inode=95%): /dev/shm 16113 MB (100% inode=99%): /boot 206 MB (87% inode=99%): /var 780237 MB (89% inode=99%):
	inodes	OK	10-26-2014 06:58:49	26d 6h 41m 10s	1/4	DISK OK - free space: / 17382 MB (93% inode=95%): /dev/shm 16113 MB (100% inode=99%): /boot 206 MB (87% inode=99%): /var 780237 MB (89% inode=99%):
	named	OK	10-26-2014 06:31:15	26d 6h 40m 44s	1/4	PROCS OK: 1 process with command name 'named'
	read_only_fs	OK	10-26-2014 06:51:41	26d 6h 40m 19s	1/4	READ_ONLY_FS OK: all ext filesystems are writable
	riak-cs-control_http	OK	10-26-2014 07:02:07	26d 6h 39m 53s	1/4	TCP OK - 0.000 second response time on port 80
	riak-cs_http	OK	10-26-2014 06:57:33	26d 6h 39m 27s	1/4	TCP OK - 0.000 second response time on port 8080
	riak_http	OK	10-26-2014 06:57:58	26d 6h 39m 1s	1/4	TCP OK - 0.000 second response time on port 8098
	riak_pb	OK	10-26-2014 06:58:24	26d 6h 38m 36s	1/4	TCP OK - 0.000 second response time on port 8087
	rpmq hung	OK	10-26-2014 06:43:49	26d 6h 41m 10s	1/4	OK: there is no process matching rpmq
	slack	OK	10-26-2014 06:31:15	26d 6h 40m 44s	1/4	Slack is running successfully
	swap	OK	10-26-2014 07:01:41	26d 6h 40m 19s	1/4	SWAP OK - 100% free (0 MB out of 0 MB)
	yum hung	OK	10-26-2014 06:42:07	26d 6h 39m 53s	1/4	OK: there is no process matching yum

Instrumentation:
MonDemand

- High performance instrumentation library
- Most used with **erlang** and **java**
- An enabled application emits LWES events to the mondemand server, which can write to several backends for graphing and aggregation (e.g.: rrd, riemann, graphite, opentsdb, quorra)


XV Riak UDS GET Latency 100
get_time_100 Avg: 44.61k
XV Riak UDS GET Latency 99
get_time_99 Avg: 1.15k
XV Riak UDS GET Latency 95
get_time_95 Avg: 889.08
XV Riak UDS GET Latency Mean
get_time_mean Avg: 627.60
XV Riak UDS GET Objects 100
get_objsize_100 Avg: 36.47k
XV Riak UDS GET Objects 99
get_objsize_99 Avg: 2.81k
XV Riak UDS GET Objects 95
get_objsize_95 Avg: 1.81k
XV Riak UDS GET Objects Mean
get_objsize_mean Avg: 981.75
XV Riak UDS GET Siblings 100
get_siblings_100 Avg: 2.57k
XV Riak UDS GET Siblings 99
get_siblings_99 Avg: 161.27
XV Riak UDS GET Siblings 95
get_siblings_95 Avg: 84.38
XV Riak UDS GET Siblings Mean
get_siblings_mean Avg: 79.93
XV Riak UDS PB Client Connects
pbc_connects Avg: 21.13k
XV Riak UDS PB Client Active
pbc_active Avg: 893.78
XV Riak UDS FSM
node_get_fsm_active Avg: 244.33
node_put_fsm_active Avg: 110.48
index_fsm_active Avg: 0.00
list_fsm_active Avg: 0.00
XV Riak UDS CPU
cpu_loadavg1 Avg: 6.91
cpu_loadavg5 Avg: 6.91
cpu_loadavg15 Avg: 6.91
XV Riak UDS Erlang Processes
sys_process_count Avg: 302.49k
XV Riak UDS Mem Erlang Alloc
memory_atom Avg: 4.98M
memory_processes Avg: 180.21M
memory_ets Avg: 105.00M
memory_binary Avg: 6.42M
memory_code Avg: 18.05M
XV Riak UDS Mem Erlang System
memory_system Avg: 26.65G
XV Riak UDS Mem OS
mem_total Avg: 67.68G
mem_allocated Avg: 65.21G
XV Riak Msg Queue Max
riak_kv_vnodeq_max Avg: 203.33
XV Riak Msg Queue Mean
riak_kv_vnodeq_mean Avg: 17.48
XV Riak Msg Queue Median
riak_kv_vnodeq_median Avg: 4.12
XV Riak UDS Read Repairs
read_repairs Avg: 30.31k
primary_outofdate Avg: 51.50k
primary_notfound Avg: 4.34k
fallback_outofdate Avg: 51.50k
fallback_notfound Avg: 4.34k

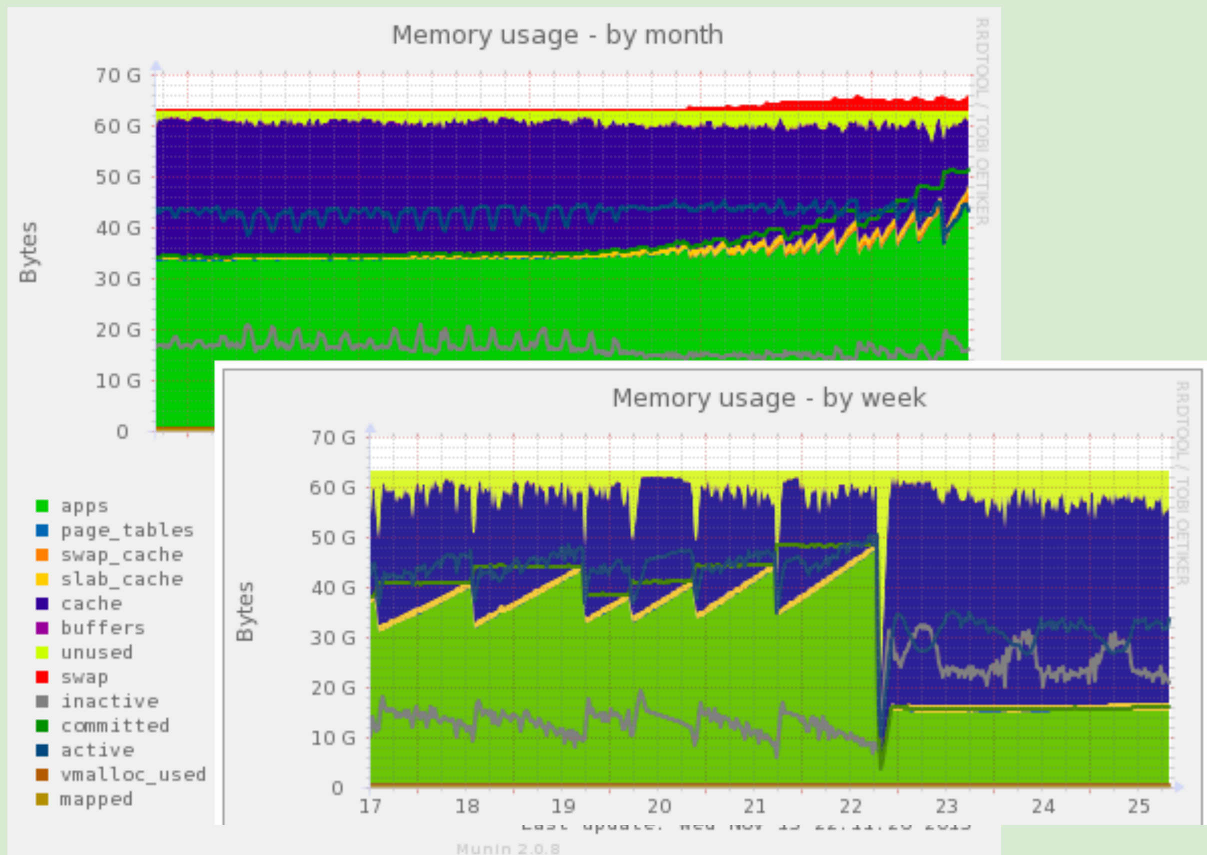
Good *Instrumentation* gives way to great *visibility*.

Visualization:
Munin

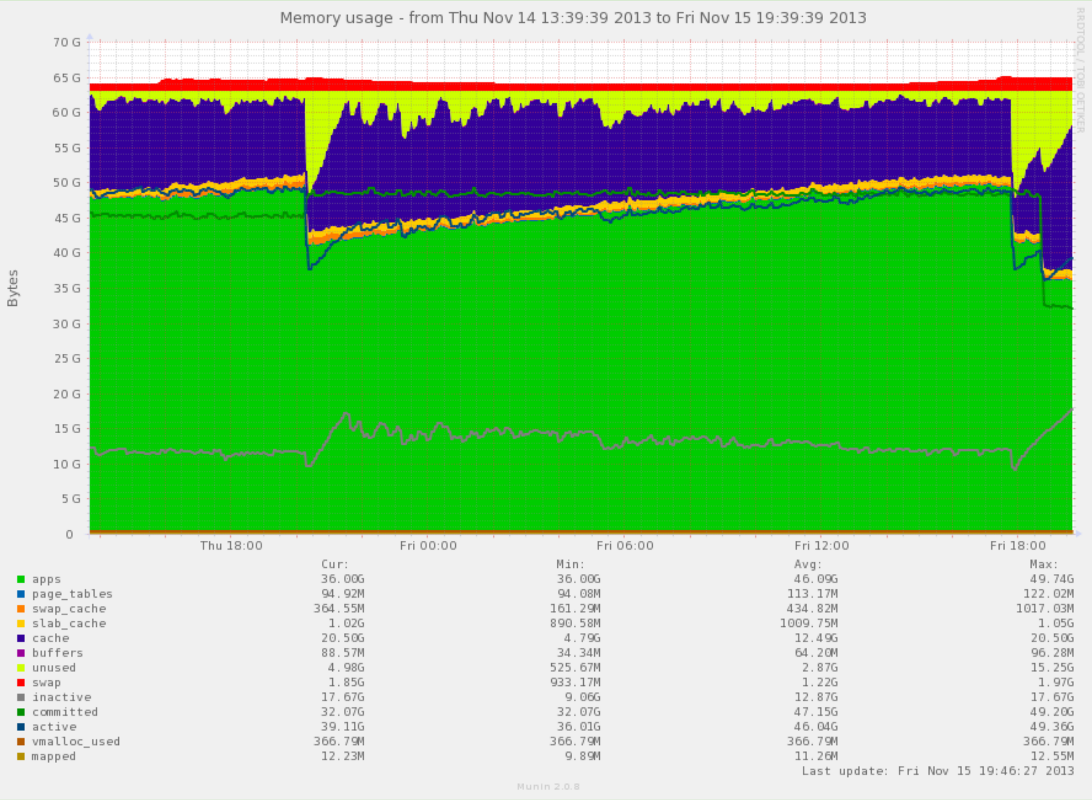
In Norse mythology Hugin and Munin are the ravens of the god king Odin. They flew all over Midgard for him, seeing and remembering, and later telling him. "Munin" means "memory".

Long-term rhythmic patterns in memory usage gives clues about what's going on with bitcask.

This pattern shows the perfect storm: erlang's history-based memory allocation, keys expiring while they're being merged, but without ample time to complete before running out of heap and getting in the way of garbage collection.



Through working with Basho, gathering data, having reliable graphing systems and log retention, we were able to pinpoint the issue and facilitate improvements for v1.4



Because of the key density compounded with expiration, the merge worker basically never goes idle, never gets its heap size reduced, runs out of memory, erlang allocates more than is physically available, and the beam process is killed by linux with an OOM message. A long startup ensues due partially to corrupt hint files.

Aggregation:
rsyslog + SumoLogic

*The <u>**R**</u>ocket-fast <u>**SYS**</u>tem for <u>**LOG**</u> processing: high-performance and modular, accepts a wide range of inputs including syslog facilities and simple file tailing, provides caching.*

***SumoLogic** enables Ops teams to perform rapid root cause analysis of critical infrastructure;*

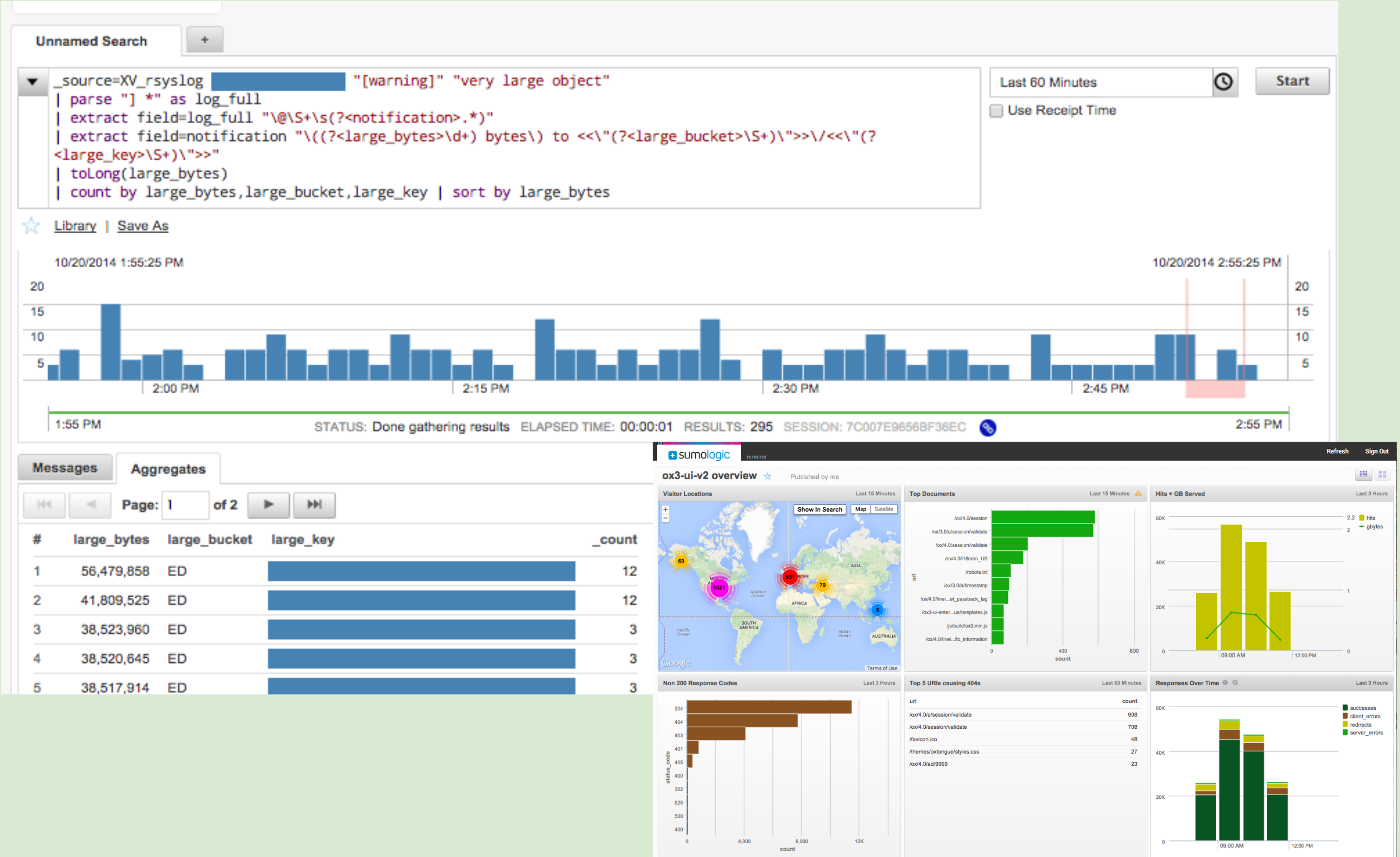
Dev teams to quickly analyze and troubleshoot production application issues;

and Security teams to uncover security incidents buried in terabytes of log data.

Some examples we use:

- Direct-to-syslog services like erlang's lager (e.g. Riak)
- File-tailing with rsyslog (e.g. namenodes, tasktrackers, kafka/storm status)
- Linux system events (sysinfo, /var/log/messages)

The search language is java-regex and fairly robust, meeting the requirements of most log parsing. There are also built-in libraries aid in creating fields from standard log formats (e.g. apache WC3, nginx, mysql).



things we can only see in logs... illuminated!

homogeneity

it must be as easy to replace nodes
as it is to let them fail

how do we “manage” these “configurations”?

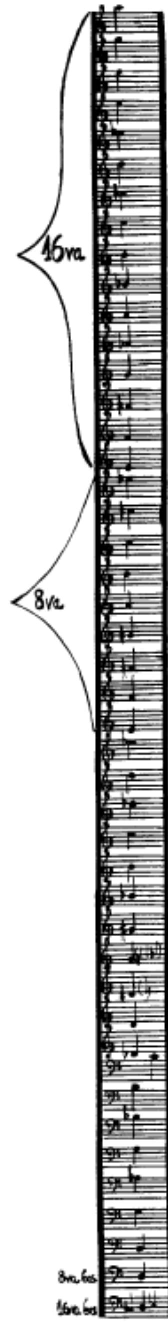
how do we guarantee high availability and avoid manual processes and human error?

...we build architectural structure

Sonatas and Interludes for Prepared Piano:
Table of Preparations (excerpt)

“[mutes of various materials are placed between the strings of the keys used, thus effecting transformations of the piano sounds with respect to all their characteristics.]”

- John Cage



MATERIAL	STRINGS LEFT TO RIGHT	DISTANCE FROM DAMPER (INCHES)	MATERIAL	STRINGS LEFT TO RIGHT	DISTANCE FROM DAMPER (INCHES)	MATERIAL	STRINGS LEFT TO RIGHT	DISTANCE FROM DAMPER (INCHES)	TONE
			SCREW	2-3	1 1/2*				A
			MED. BOLT	2-3	1 7/8*				G
			SCREW	2-3	1 5/8*				F
			SCREW	2-3	1 15/16*				E
			SCREW	2-3	1 3/4*				E♭
			SM. BOLT	2-3	2*				D
			SCREW	2-3	1 9/16*				C♯
			FURNITURE BOLT	2-3	2 3/16*				C
			SCREW	2-3	2 1/2*				B
			SCREW	2-3	1 7/8*				B♭
			MED. BOLT	2-3	2 5/8*				A
			SCREW	2-3	2 1/4*				A♭
			SCREW	2-3	3 3/4*				G
			SCREW	2-3	2 5/16*				F♯
SCREW	1-2	3/4*	FURN. BOLT + 2 NUTS	2-3	2 1/8*	SCREW + 2 NUTS	2-3	3 1/4*	F
			SCREW	2-3	1 15/16*				E
			FURNITURE BOLT	2-3	1 7/8				E♭
			SCREW	2-3	1 15/16				C♯
			SCREW	2-3	1 1/16				C
(DAMPER TO BRIDGE = 4 5/16; ADJUST ACCORDINGLY)			MED. BOLT	2-3	3 3/4				B
			SCREW	2-3	4 5/16				A
RUBBER	1-2-3	4 1/2	FURNITURE BOLT	2-3	1 1/4				G♯
			SCREW	2-3	1 3/4				F♯
			SCREW	2-3	2 5/16				F
RUBBER	1-2-3	5 3/4							E
RUBBER	1-2-3	6 1/2	FURN. BOLT + NUT	2-3	6 7/8				E♭
			FURNITURE BOLT	2-3	2 3/16				D
RUBBER	1-2-3	3 5/8							D♭
			BOLT	2-3	7 7/8				C
			BOLT	2-3	2				B
SCREW	1-2	10	SCREW	2-3	1	RUBBER	1-2-3	8 1/4	B♭
(PLASTIC (See G))	1-2-3	2 5/16				RUBBER	1-2-3	4 1/2	G♯
PLASTIC (OVER 1 - UNDER 2-3)	1-2-3	2 7/8				RUBBER	1-2-3	10 1/8	G
(PLASTIC (See D))	1-2-3	4 1/4				RUBBER	1-2-3	5 7/16	D♯
PLASTIC (OVER 1 - UNDER 2-3)	1-2-3	4 5/8				RUBBER	1-2-3	9 3/4	D
BOLT	1-2	15 1/2	BOLT	2-3	1 1/16	RUBBER	1-2-3	14 1/8	D♭
BOLT	1-2	14 1/2	BOLT	2-3	7/8	RUBBER	1-2-3	6 1/2	C
BOLT	1-2	14 3/4	BOLT	2-3	9/16	RUBBER	1-2-3	14	B
RUBBER	1-2-3	9 1/2	MED. BOLT	2-3	10 1/8				B♭
SCREW	1-2	5 7/8	LG. BOLT	2-3	5 7/8	SCREW + NUTS	1-2	1	A
BOLT	1-2	7 1/8	MED. BOLT	2-3	2 1/4	RUBBER	1-2-3	4 1/8	A♭
LONG BOLT	1-2	8 3/4	LG BOLT	2-3	3 1/4				G
			BOLT	2-3	1 1/16				D
SCREW + RUBBER	1-2	4 5/16							D
ERASER (OVER D UNDER C♯+E♭) AN. RUBBER CO. #365	1	6 3/4							D

*MEASURE FROM BRIDGE.

Salt Stack

Structure Management and Orchestration

Orchestration is a compositional art in itself: understanding the components, the way they interact, their ranges and capacities, the way processes and jobs and textures and sonorities are layered and synchronized, pipelined in and harmonized.

Instrumentation provides the raw materials, and giving responsibility to an instrument means expecting a continuously reliable result, and that's the goal with large-scale distributed clusters: guaranteeing the micro-level *be* the macrocosm, in the same way, every time.

Configure intelligently, repeatably, and elegantly.

Give the data every chance to be **awesome** by making management *easy*.

Salt provides a **structure** to ensure configurations are consistent and repeatable, upholding a homogenous approach to cluster management

identical hardware provides a predictable balance of resources!

and easily replaceable hardware upholds the "let-it-die" mantra of distributed systems

this example shows the layout of a riak-cs system, providing configuration management and service orchestration

```
#
# riakfs states
#

include:
  - element.network.rsyslog
  - .packages
  - .files.config
  - .services
  - .stanchion

{% for kernel_param, value in salt['pillar.get']('riakfs-etc:sysctl', {}).items() %}
{{ kernel_param }}:
  sysctl.present:
    - value: {{ value }}
{% endfor %}

salt-call state.highstate:
  cron.present:
    - user: root
    - minute: random

# disable swap
swapoff_cmd:
  cmd.run:
    - name: /sbin/swapoff -a ; /bin/sed -i '/swap/d' /etc/fstab

NORMAL > +0 ~0 -0 > init.sls          < sls < utf-8[unix] < 3% :  1:  1
"init.sls" 27L, 468C
```

```
|├── files
|│   ├── config.sls
|│   ├── etc
|│   │   ├── haproxy
|│   │   │   └── haproxy.cfg.jinja2
|│   │   ├── nginx
|│   │   │   └── conf.d
|│   │   │       ├── riak-cs-control.conf
|│   │   │       └── riakfs.htpasswd
|│   │   ├── riak
|│   │   │   ├── app.config_riak-ee_1.4.jinja2
|│   │   │   └── vm.args_riak-ee_1.4.jinja2
|│   │   ├── riak-cs
|│   │   │   ├── app.config_riak-cs_1.4.jinja2
|│   │   │   ├── app.config_riak-cs_1.5.jinja2
|│   │   │   └── vm.args_riak-cs_1.5.jinja2
|│   │   ├── riak-cs-control
|│   │   │   ├── app.config_riak-cs-control_1.0.jinja2
|│   │   │   └── vm.args_riak-cs-control_1.0.jinja2
|│   │   ├── rsyslog.d
|│   │   │   └── riakfs.conf
|│   │   ├── salt
|│   │   │   ├── grains_not_stanchion
|│   │   │   └── grains_stanchion
|│   │   ├── security
|│   │   │   └── limits.d
|│   │   │       └── 99-riakfs.conf
|│   │   ├── stanchion
|│   │   │   ├── app.config_stanchion.jinja2
|│   │   │   └── vm.args_stanchion.jinja2
|│   │   └── sysconfig
|│   │       ├── iptables_block_stanchion
|│   │       └── iptables_stanchion
|│   └── usr
|│       ├── lib64
|│       │   └── riak
|│       │       └── lib
|│       │           └── basho-patches
|│       │               └── schedmon_i.beam
|│       └── local
|│           └── bin
|│               └── s3curl.pl
|├── init.sls
|├── packages.sls
|├── services.sls
|└── stanchion.sls
```

the rhythmic structure of the data is supported by the architectural structure of solid configuration management

When Elephants Attack

or

The Curse of Hosting Multiple Distributed Systems

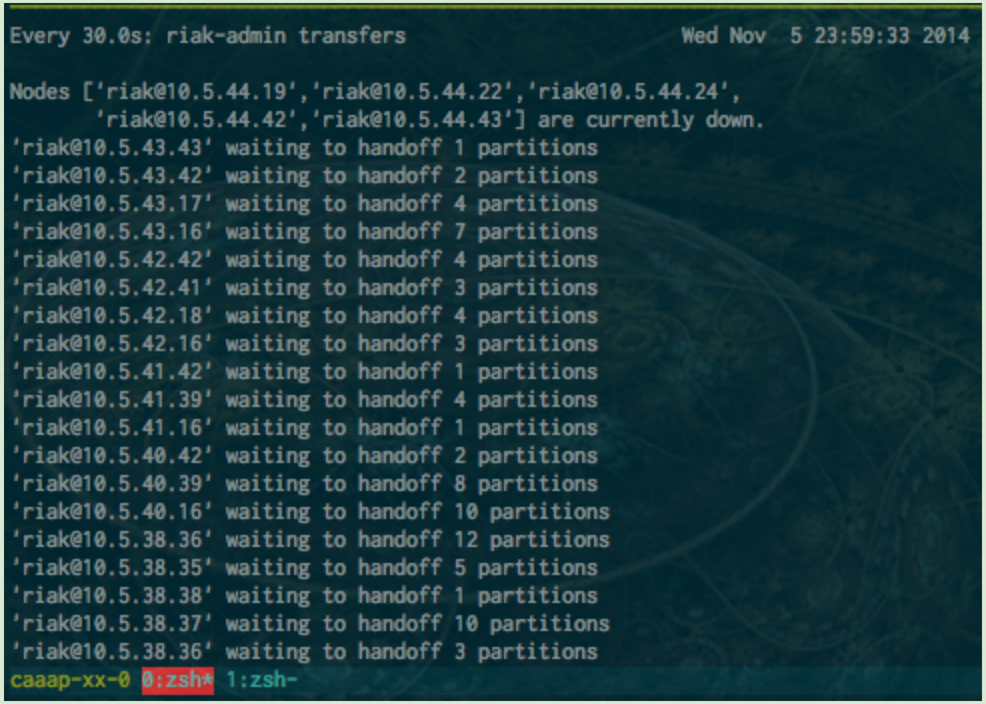
Resolution by correlation:
What are those spikes?

It should always look like this…

Then began looking like this…

and thanks to well architected configuration management,
we know all things must be equal…?

After many weeks of troubleshooting nodes, making sure all hardware was operating correctly, actually finding some about-to-fail nodes and replacing them, I happen to catch the following display exactly when a spike in latency occurred...



then one day...

… and comparing with Cacti, our network monitoring tool,
plus sumologic reporting errors from the frontend,
i found the culprit: hadoop network saturation

- ★ ***Instrumentation*** at the right places allowed collection of important data points
- ★ ***Visualization*** of these data points showed the stark contrasts seen in data rhythms
- ★ Log ***Aggregation*** illuminated front-end errors
- ★ Configuration management guaranteed a ***homogenized*** distributed cluster for ruling out misconfiguration, and allowed for painless re-deployment of nodes to address issues

By observation of **Measure** and **Internalization** of data rhythms, the root cause was finally uncovered.

lesson learned: don’t let your elephant beat up your ninjas

Q & A

thanks!