



loggly

Lucene Revolution
Barcelona 2011

Using SolrCloud for real

Whoggly?



- I'm Jon, a happy lucene hacker since 2004
- We do Logging as a Service (SAAS with a single focus)
 - Consolidation, Archiving, Search, Alerting (soon!)
 - You stream your logs to us, we index them, you search
- Full public launch on Feb 2, 2011
- Every customer has their own index
 - ~8k shards, ~7B docs, ~3TB index, 3-5k events/sec, 1-1.5 MB/sec
- Search finds the splinter in the log-jam
 - What happened? When? How often?

Loggly | Shell

https://geekceo.loggly.com/shell/#/_graph 404

loggly From: NOW-1HOUR Until: NOW Inputs: All

```

3.0.30729; .NET4.0C)" RG4AxX8AAAEAAAxAUaUEAAAAT 404
2011-04-26 14:48:51.076 loggly_web 174.129.233.147
207.46.204.180 - - [26/Apr/2011:14:48:50 -0700] "GET /robots.txt HT
" "Mozilla/5.0 (compatible; bingbot/2.0; +http://www.bing.com/bingb
Results from range 2011-04-26T21:20:04.463Z - 2011-04-26T21:48:51.076Z for term
kordless@geekceo> graph 404

```

Events from NOW-1HOUR until NOW

kordless@geekceo>

Loggly | Main Dashboard

https://geekceo.loggly.com

profile | help | logout

kordless@geekceo> search for broken things

dashboard inputs users account monitor

Log History

73,608 events

Inputs

Name	Events In Last Day
default	5,837
gecko_http	0
geekceo_http	0
kedge_http	6
logg_ly	51,038
loggly_web	15,212
port514	0
titus	1,571

Devices

Name/IP	Events In Last Day
72.14.194.33	34,175
72.14.194.17	16,863
Loggly's web server	15,212
Loggly Office	5,837
98.210.51.152	1,577

Daily Usage

Today's usage: 11 MB

3,000 (MB)

■ Dropped ■ Indexed

Time is on our side...

- We're not solving a typical search problem
 - ▶ Endless stream of data (all your logs)
 - ▶ Time is our “score”
 - ▶ Write once, update never
 - ▶ Large number of very small documents (events)
 - ▶ Search is mostly about what just happened
- So, simple index life-cycle with “natural” sharding
 - ▶ For us, a shard is slice of a single customers index, based on start and end time

Why SolrCloud?



- The wheel existed, and keeps getting better
 - Thanks to the community. Big Thanks to Mark & Yonik
- Solr: Multi-core, Plugins, Facets, migration, caching, ...
 - Our Solr instances have hundreds of cores (one per shard)
 - We've made very few changes to core Solr code
- Zookeeper: state of all nodes & shards, so...
 - Automatic cluster config
 - Cluster & Shard changes visible everywhere, almost instantly
 - Any node can service any request (except for indexing)

Cluster Management



- Solr instances register & deregister themselves in ZooKeeper
 - always know what Solr instances are available
- All Solr configs in ZK except for solr.xml
 - all instances use same schema, etc, so simple management
 - solr.xml is “special”, instance specific
- We’ve added our own persistent data
 - Loggly-specific config, and some performance data for Solr
 - Other app configs, “live” status

Index Management

- One Collection (“index”) per customer
 - Sharded by time, multiple shards per customer
- Shards migrated from node to node using replication
 - Minor changes to existing Solr replication code
- Shards merged by us, never automatically
 - We merge to create longer time-slices
 - Doing it manually makes merge load more predictable
- Completely distributed management, no “Master”
 - Simple, Robust, “need to know”

SolrCloud, meet reality

- Day 1 (patch to trunk, 18 months ago), mostly JFW'ed. Since then...
 - ▶ We had to fix a couple of TODO's
 - ▶ We changed shard selection for search
 - ▶ We hit a ZK performance wall
 - ▶ We've pulled (most of) the Solr ZK code into an external jar, and
 - ▶ added some utilities to the ZK controller/client
 - ▶ extended the shard node to include custom data, including "sleep/wake" state and S3 archive state
 - ▶ added non-Solr application configuration to ZK

TODO's

- Very little missing, even 18 months ago...
- *FacetComponent.countFacets()*
 - ▶ `// TODO: facet dates`
 - ▶ We did it. Since been added to trunk (not our code)
- *ZkController.unregister()*
 - ▶ `// TODO : perhaps mark the core down in zk?`
 - ▶ We did it: remove the shard from ZK, walk up the tree removing its parents if they're empty
- One more, but no spoilers...

Shard selection

- *QueryComponent.checkDistributed()* selects shards for each “slice”, based on the Collection of the core being queried.
- We changed some things for version 1...
 - use a “loggly” core, and pass in the collection
 - select the biggest shard when overlaps occur
 - select the “best” copy of a shard on multiple machines
- Now we use plugin variant of admin handler
 - avoids special case core
 - lets us do shard-level short-circuiting for search (not facets)

Performance Whack-a-Mole

- In the last few months...
 - Default SolrCloud ZooKeeper state implementation
 - Not happy with 1000's of shards
 - Default CoreContainer multi-core behaviour
 - Not very happy with 100's (or 1000's) of cores
- Next...
 - Hot-spots in migrations/merging
 - current code is too complex, needs cleanup/dumbing down

SolrCloud – CloudState

- ZkStateReader... (the TODO that bit us, hard!)
 - `// TODO: - possibly: incremental update rather than reread everything?`
 - Yep, EVERYTHING in ZooKeeper, every time we update anything
 - to be fair, we're kind of a pathological case
 - Watchers for every collection, every shard
- The Perfect storm
 - Full read of ZK with 1000's of shards ain't fast
 - We weren't using scheduled updates correctly
 - We're updating frequently, triggering lots of Watcher updates
- **Up to 20 second wait for CloudState**

“Just avoid holding it in that way”

- Watch fewer things
 - Every node doesn't have to know about every shard change
- Incremental updates
 - When a watch triggers, we rebuild only that data
- On-demand Collection updates
 - We read individual Collection info whenever its needed
- Wait for CloudState is now usually $< 10\text{ms}$
 - 100's of CloudState updates / minute

SolrCloud – CoreContainer

- Default multi-core behaviour works really well
 - We've had 1000's of shards on a quad core 16GB box
- But...
 - Context switching like crazy, especially on indexers
 - Lots of RAM, especially when merging & warming
 - Lots of GC, especially when servicing lots of search requests, or handling large result sets, or merging
 - Startup times get very very very long (minutes)

Goodnight Solr...

- We now sleep/wake shards (~50% of shards are asleep)
 - solr.xml & ZK extended to include state
 - sleeping shards are close()'ed, removing from CoreContainer
- Changes to CoreContainer
 - Added LogglyCore – simple wrapper around SolrCore, manages state
 - Replaced “cores” (Map of SolrCores) with “liveCores” and “allCores” (Maps of LogglyCores)
- On startup we start up Jetty first, then open the shards
 - Solr “available” in seconds, shards come online as needed
- Same mechanism used to manage shard archiving

ZK Utilities

- Cleanup
 - ▶ **nuke:** `rm -rf` for the ZooKeeper tree
 - ▶ **purgeShards:** delete all shards for collection X
 - ▶ **restore:** restore shards for collection X from disk
 - ▶ **purgeNode:** delete all references to node Y
 - ▶ **balance:** balance shards across all available nodes
- **upload:** upload a file or directory

Loggly magic

- We've spent lots of time on Shard & Cluster Management
 - Plugins make extending functionality easy
 - ZK makes cluster management easy, for Solr and associated apps
- Minimal changes to Solr itself. Biggest are...
 - OMQ streaming data input
 - Automatic sharding, based on time
 - LogglyCore for Sleep/Wake
- Lots of custom admin API extensions, to tie Solr into overall system

Loggly ZK magic

- We use ZK to store
 - indexing performance data (load balancing indexers)
 - shard state (awake / asleep, archived)
- We use ZK in our other apps, in /cluster
 - we have \sim “live_nodes” for ALL apps
 - one-off’s easy when entire state of the system is available
 - json output + ruby/python + REST = what happened?
- ZK is very robust
 - multiple ZK host failures, no downtime for ZK quorum

Other Stuff We Use

- Amazon's AWS for infrastructure (EC2, S3)
- Syslog-NG for syslog/TLS input services
- 0MQ for event queuing & work distribution
- MongoDB for statistics and API /stat methods
- Node.js for HTTP/HTTPs input services
- Django/Python for middleware/app

loggly

ACK/FIN

One Last Thing... <http://loggly.ly/jobs>