Apache Cassandra multi-dc essentials

Julien Anguenot (@anguenot)

VP Software Engineering, iland cloud

1	key notions & configuration
2	~~bootstrapping & decommissioning new DCs / nodes~~
3	operations pain points
4	q&a

iland cloud?

- **public / private cloud provider**
- footprint in **U.S., EU and Asia**
- **compliance, advanced security**
- **custom SLA**
- Apache Cassandra users since 1.2

  C* summit 2015 presentation: http://www.slideshare.net/anguenot/leveraging-cassandra-for-realtime-multidatacenter-public-cloud-analytics

  Apache Cassandra spanning 6 datacenters.
- http://www.iland.com

key notions & configuration

What is Apache Cassandra?

- **distributed partitioned row store**
- **physical multi-datacenter native support**
- tailored (features) for multi-datacenter deployment

Why multi-dc deployments?

- multi-datacenter **distributed application**
- **performances**

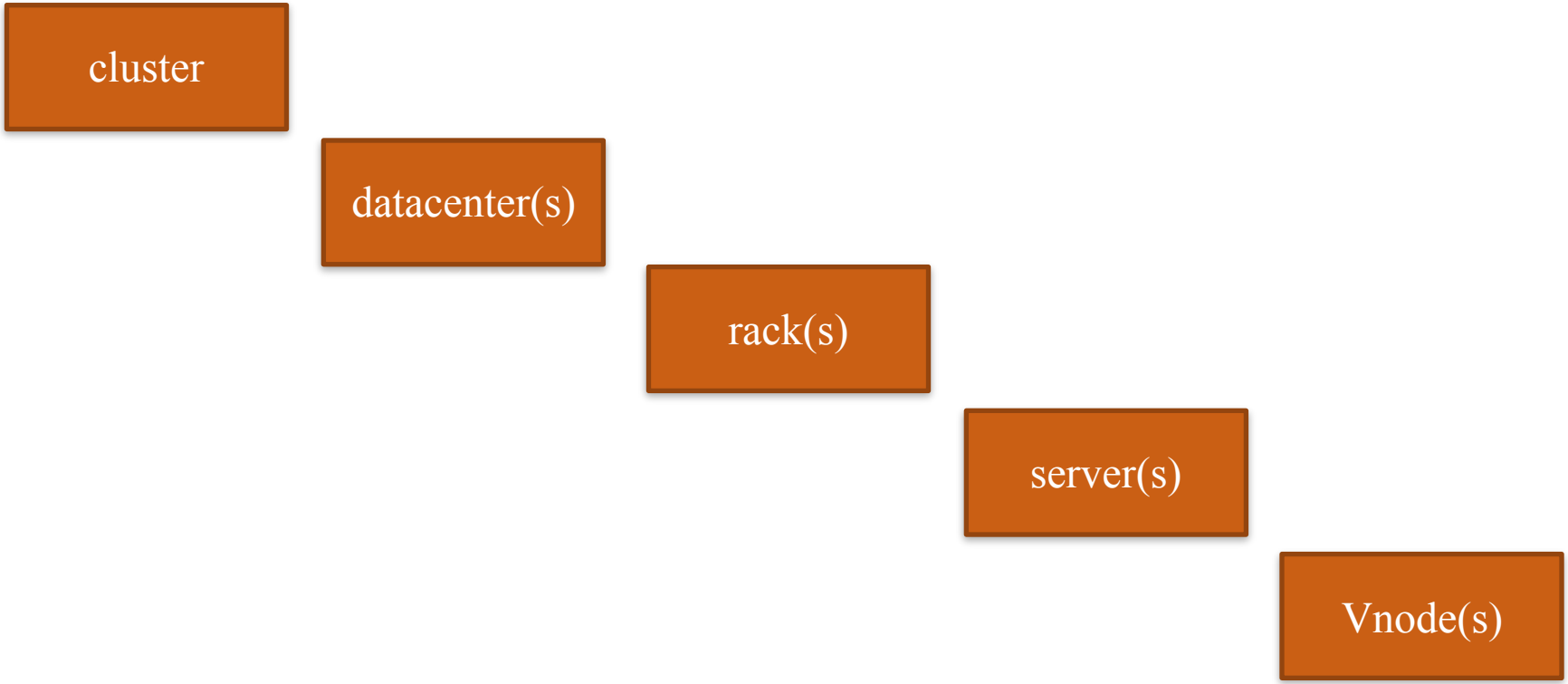
  read / write isolation or geographical distribution
- **disaster recovery** (DR)

  failover and redundancy
- **analytics**

Essentially…

- sequential writes in **commit log** (flat files)
- indexed and written in **memtables** (in-memory: write-back cache)
- serialized to disk in a **SSTable** data file
- writes are **partitioned and replicated** automatically in cluster
- SSTables consolidated though **compaction** to clean **tombstones**
- **repairs** to ensure consistency cluster wide

Cassandra hierarchy of elements



Cassandra cluster

- **the sum total of all the servers in your database throughout all datacenters**
- span physical locations
- defines one or more **keyspaces**
- *no cross-cluster replication*

cassandra.yaml: `cluster_name`

```
# The name of the cluster. This is mainly used to prevent machines in
# one logical cluster from joining another.
cluster_name: ‘my little cluster'
```

Cassandra datacenter

- **grouping of nodes**
- synonymous with **replication group**
- a grouping of nodes configured together for replication purposes
- **each datacenter contains a complete token ring**
- collection of **Cassandra racks**

Cassandra rack

- **collection of servers**
- **at least** one (1) rack per datacenter
- one (1) rack is the most simple and common setup

Cassandra server

- Cassandra (the software) instance installed on a machine
- **AKA node**
- contains virtual nodes (or **Vnodes**). 256 by default

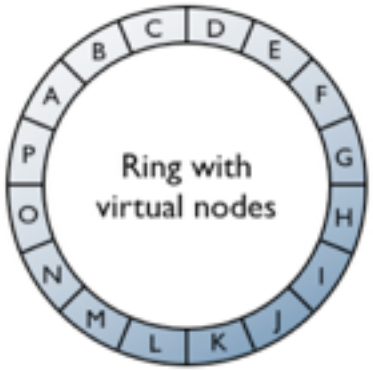
Virtual nodes (Vnodes)

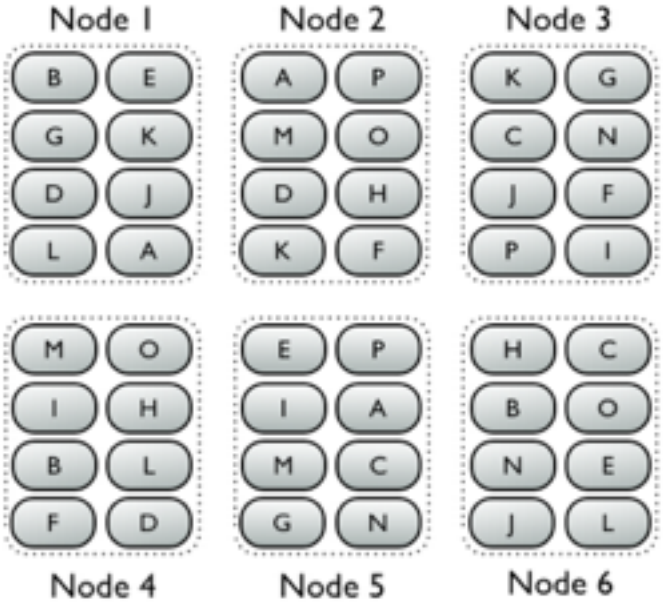
- C* >= 1.2
- data storage layer within a server
- tokens **automatically calculated and assigned randomly** for all Vnodes
- **automatic rebalancing**
- **no manual token generation and assignment**
- default to 256 (num_tokens in cassandra.yaml)

cassandra.yaml: `num_tokens`

```
# This defines the number of tokens randomly assigned to this node on the ring
# The more tokens, relative to other nodes, the larger the proportion of data
# that this node will store. You probably want all nodes to have the same number
# of tokens assuming they have equal hardware capability.
#
# If you leave this unspecified, Cassandra will use the default of 1 token for
legacy compatibility,
# and will use the initial_token as described below.
#
# Specifying initial_token will override this setting on the node's initial start,
# on subsequent starts, this setting will apply even if initial token is set.
#
# If you already have a cluster with 1 token per node, and wish to migrate to
# multiple tokens per node, see http://wiki.apache.org/cassandra/Operations
num_tokens: 256
```

Ring with Vnodes





Partition

- **individual unit of data**
- partitions are replicated across multiple Vnodes
- each **copy** of the **partition** is called a **replica**

Vnodes and consistent hashing

- allows **distribution of data across a cluster**
- **Cassandra assigns a hash value to each partition key**
- each Vnode in the cluster is responsible for a range of data based on the hash value
- **Cassandra places the data on each node according to the value of the partition key and the range that the node is responsible for**

Partitioner (1/2)

- partitions the data across the cluster
- **function for deriving a token representing a row from its partition key**
- **hashing function**
- each row of data is then distributed across the cluster by the value of the token

Partitioner (2/2)

- **Murmur3Partitioner** (default C* >= 1.2)
  uniformly distributes data across the cluster based on MurmurHash hash values
- **RandomPartitioner** (default C* < 1.2)
  uniformly distributes data across the cluster based on MD5 hash values
- **ByteOrderedPartitioner** (BBB)
  keeps an ordered distribution of data lexically by key bytes

cassandra.yaml: `partitioner`

```
# The partitioner is responsible for distributing groups of rows (by
# partition key) across nodes in the cluster.  You should leave this
# alone for new clusters.  The partitioner can NOT be changed without
# reloading all data, so when upgrading you should set this to the
# same partitioner you were already using.
#
# Besides Murmur3Partitioner, partitioners included for backwards
# compatibility include RandomPartitioner, ByteOrderedPartitioner, and
# OrderPreservingPartitioner.
#
partitioner: org.apache.cassandra.dht.Murmur3Partitioner
```

Partitioner example (1/4)

name	age	car	gender
jim	36	camaro	M
carol	37	bmw	F
johnny	12		M
suzy	10		F

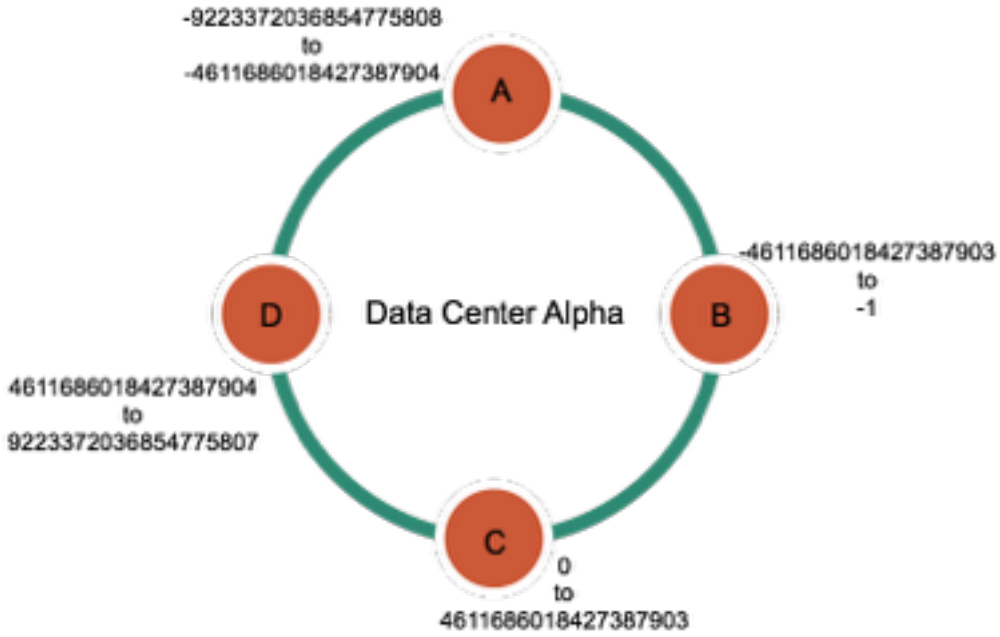
Credits to Datastax. Extract from documentation.

Partitioner example (2/4)

Partition key	Murmur3 hash value
jim	-2245462676723223822
carol	7723358927203680754
johnny	-6723372854036780875
suzy	1168604627387940318

Credits to Datastax. Extract from documentation.

Partitioner example (3/4)



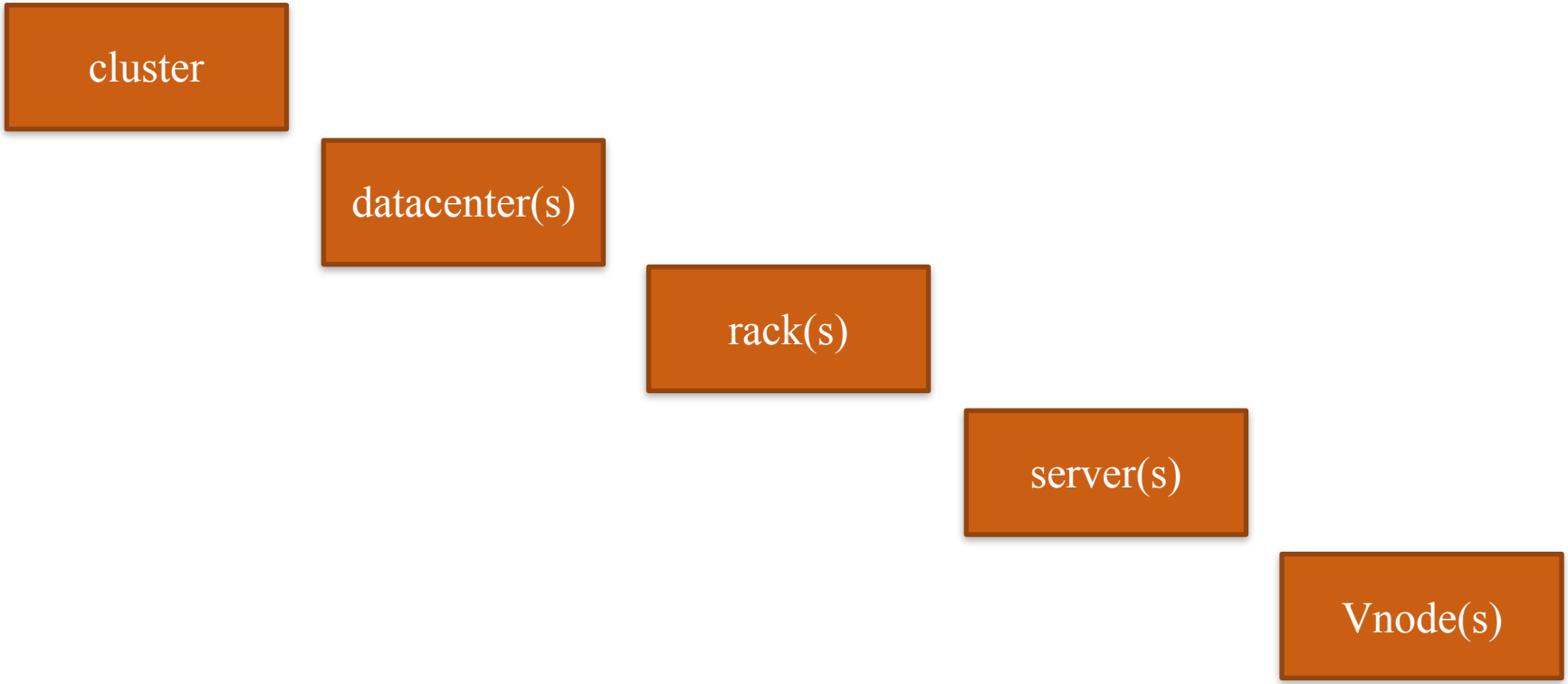
Credits to Datastax. Extract from documentation.

Partitioner example (4/4)

Node	Start range	End range	Partition key	Hash value
A	-9223372036854775808	-4611686018427387904	johnny	-6723372854036780875
B	-4611686018427387903	-1	jim	-2245462676723223822
C	0	4611686018427387903	suzy	1168604627387940318
D	4611686018427387904	9223372036854775807	carol	7723358927203680754

Credits to Datastax. Extract from documentation.

Cassandra hierarchy of elements (recap)



Cassandra Keyspace (KS)

- **namespace container that defines how data is replicated on nodes**
- cluster defines KS
- contains **tables**
- **defines the replica placement strategy and the number of replicas**

Data replication

- process of storing **copies** (**replicas**) on multiple nodes
- KS has a **replication factor** (**RF**) and **replica placement strategy**
- max (RF) = max(number of nodes) in one (1) data center
- **data replication is defined per datacenter**

Replica placement strategy

there are two (2) available **replication strategies**:

1. SimpleStrategy (single DC)
2. NetworkTopologyStrategy (recommended cause easier to expand)

choose strategy depending on **failure scenarios** and application needs for **consistency level**

Consistency Level (CL)

- how many nodes must ACK operation at **client level**?
- **tunable** consistency at **client level**
- ANY
- ONE
- ALL
- QUORUM / LOCAL_QUORUM (DC only)
- **SERIAL** and conditional updates (*IF DOES NOT EXIST*)

local_quorum examples

- nodes=3, RF=3 - can tolerate 1 replica being down
- nodes=5, RF=3 - can tolerate 2 replica being down
- etc.

snitch (1/3)

- determines which data centers & racks nodes belong to
- **informs Cassandra about the network topology**
- effective routing
- **replication strategy places the replicas based on snitch**

snitch (2/3)

- **SimpleSnitch**
  single DC only
- **GossipingPropertySnitch**
  cassandra-rackdc.properties
- **PropertyFileSnitch**
  cassandra-topology.properties
- **RackInferringSnitch**
  determined by rack and data center, which are 3rd and 2nd octet of each node's IP respectively

Snitch files (examples)

cassandra-rackdc.properties

These properties are used with GossipingPropertyFileSnitch and will indicate the rack and dc for this individual node only

dc=west-dc
rack=rack1

cassandra-topology.properties

 # These properties are used with PropertyFileSnitch and will be identical on every nodes.
 # Cassandra Node IP=Data Center:Rack

192.168.1.100=east-dc:rack1
192.168.1.101=east-dc:rack1
192.168.1.102=east-dc:rack1

192.168.2.100=west-dc:rack1
192.168.2.101=west-dc:rack1
192.168.2.102=west-dc:rack1

snitch (3/3)

- more deployment specific snitches for EC2, Google, Cloudstack etc.

cassandra.yaml: `endpoint_snitch`

```
# You can use a custom Snitch by setting this to the full class name
# of the snitch, which will be assumed to be on your classpath.
endpoint_snitch: RackInferringSnitch
```

seed node

- bootstrapping the gossip process for new nodes joining the cluster
- use the same list of seed nodes for all nodes in a cluster
- include at least one (1) node of each datacenter in seeds list

cassandra.yaml: `seed_provider`

```
# any class that implements the SeedProvider interface and has a
# constructor that takes a Map<String, String> of parameters will do.
seed_provider:
    # Addresses of hosts that are deemed contact points.
    # Cassandra nodes use this list of hosts to find each other and learn
    # the topology of the ring.  You must change this if you are running
    # multiple nodes!
    - class_name: org.apache.cassandra.locator.SimpleSeedProvider
      parameters:
          # seeds is actually a comma-delimited list of addresses.
          # Ex: "<ip1>,<ip2>,<ip3>"
          - seeds: “10.239.206.80,10.239.206.81,10.244.206.80,10.244.206.81"
```

Gossip

- **peer-to-peer communication protocol**
- discover and share location and state information about the other nodes in a Cassandra cluster
- **persisted by each node**
- nodes exchange state messages on regular basis

operations pain points

operations pain points

- bootstrapping new nodes / new DCs
- repairs
- hints
- tombstones
- compactions
- indexes
- sick nodes

bootstrapping new nodes / new DCs

- **slow** if dense nodes
- don’t expect to “just add a new node” to accommodate load as it comes.
- use C* >= 2.2 (*nodetool bootstrap resume*)
- **pressure on network**
- first seed node: *nodetool rebuild -- <dc>*
  stream timeouts / throughput params

Throttles all outbound streaming file transfers on this node to the
given total throughput in Mbps. This is necessary because Cassandra does
mostly sequential IO when streaming data during bootstrap or repair, which
can lead to saturating the network connection and degrading rpc performance.
When unset, the default is 200 Mbps or 25 MB/s.
stream_throughput_outbound_megabits_per_sec: 200

Throttles all streaming file transfer between the datacenters,
this setting allows users to throttle inter dc stream throughput in addition
to throttling all network stream traffic as configured with
stream_throughput_outbound_megabits_per_sec
When unset, the default is 200 Mbps or 25 MB/s
inter_dc_stream_throughput_outbound_megabits_per_sec: 200

Set socket timeout for streaming operation.
The stream session is failed if no data/ack is received by any of the participants
within that period, which means this should also be sufficient to stream a large
sstable or rebuild table indexes.
Default value is 86400000ms, which means stale streams timeout after 24 hours.
A value of zero means stream sockets should never time out.
 # streaming_socket_timeout_in_ms: 86400000

repairs

- Anti-Entropy: QUORUM & ALL replicas compared for CF and discrepancies fixed.
- must run before `gc_grace_period` (10 days by default)
- cluster pressure
- network pressure (same as bootstrapping)
- GC fun...
- plan extra cluster capability for repairs

repairs: what to do then?

- you must define a repair strategy from the beginning
  - custom tooling
  - DSE and OpsCenter
  - Spotify Cassandra Reaper:
    https://github.com/spotify/cassandra-reaper
    https://github.com/adejanovski/cassandra-reaper
- do not necessary repair everything all the time (know your data)

hints

- if node down: spool and redelivery
- slow and broken until 3.0: must truncate manually as some are left off
- < 3.0: SSTables (which means compactions)
- >= 3.0 flat files with compression
- >= 3.0 **disablehintsfordc** / **enablehintsfordc** to selectively disable or enable hinted handoffs for a data center.
- increase hint delivery threads along with # of DCs

cassandra.yaml: `hints`

```
max_hints_delivery_threads: 2

# Directory where Cassandra should store hints.
# If not set, the default directory is $CASSANDRA_HOME/data/hints.
# hints_directory: /var/lib/cassandra/hints

# Compression to apply to the hint files. If omitted, hints files
# will be written uncompressed. LZ4, Snappy, and Deflate compressors
# are supported.
#hints_compression:
#   - class_name: LZ4Compressor
#     parameters:
#         -
```

compactions

- **process of merging SSTables to single files**
- IO heavy: GC / CPU / eat disk space
- removes tombstones
- **high write throughout on a single table from every nodes of every DC might eat your CPU w/ compactions: choose compaction strategy wisely!**
- increment # of compactors

cassandra.yaml: `concurrent_compactors`

```
[…]

concurrent_compactors: 2

[…]
```

tombstones

- monitor for tombstones warnings
- maintenance ops issue or application level issue?

cassandra.yaml: `tombstone_warn_threshold`

When executing a scan, within or across a partition, we need to keep the
tombstones seen in memory so we can return them to the coordinator, which
will use them to make sure other replicas also know about the deleted rows.
With workloads that generate a lot of tombstones, this can cause performance
problems and even exaust the server heap.
(http://www.datastax.com/dev/blog/cassandra-anti-patterns-queues-and-queue-like-datasets)
Adjust the thresholds here if you understand the dangers and want to
scan more tombstones anyway. These thresholds may also be adjusted at runtime
using the StorageService mbean.
tombstone_warn_threshold: 1000
tombstone_failure_threshold: 100000

indexes

- secondary indexes? SASI?
- if looking for search:
  - use DSE and its integrated search
  - check https://github.com/vroyer/elassandra
  - use another service to do the job outside C* and move / sync data

sick nodes

- what is the issue?
- remove “sick nodes” from the ring when it happens

must reads

- Datastax Apache Cassandra documentation
  - http://docs.datastax.com/en//cassandra/3.0/cassandra/cassandraAbout.html
- Al's Cassandra 2.1 tuning guide
  - https://tobert.github.io/pages/als-cassandra-21-tuning-guide.html
- cassandra-user mailing list
  - http://www.planetcassandra.org/apache-cassandra-mailing-lists/

Q&A

@anguenot

http://www.slideshare.net/anguenot/

iland