LucidWorks™

Introduction to SolrCloud
ApacheCon, April 7, 2014

Timothy Potter

Search | Discover | Analyze

SolrCloud tag cloud

created using Wordle.net

My SolrCloud Experience

- Currently, working on scaling up to a 200+ node deployment at LucidWorks
- Operated 36 node cluster in AWS for Dachis Group (1.5 years ago, 18 shards ~900M docs)
- Contributed several tests and patches to the code base
- Built a Fabric/boto framework for deploying and managing a cluster in EC2
- Co-author of Solr In Action; wrote CH 13 which covers SolrCloud

What is SolrCloud?

LucidWorks

Subset of *optional* features in Solr to enable and simplify horizontal scaling a search index using **sharding** and **replication**.

Goals

scalability, performance, high-availability, simplicity, and elasticity

Caveat

Still evolving ... try to stay up-to-date with recent releases

Terminology

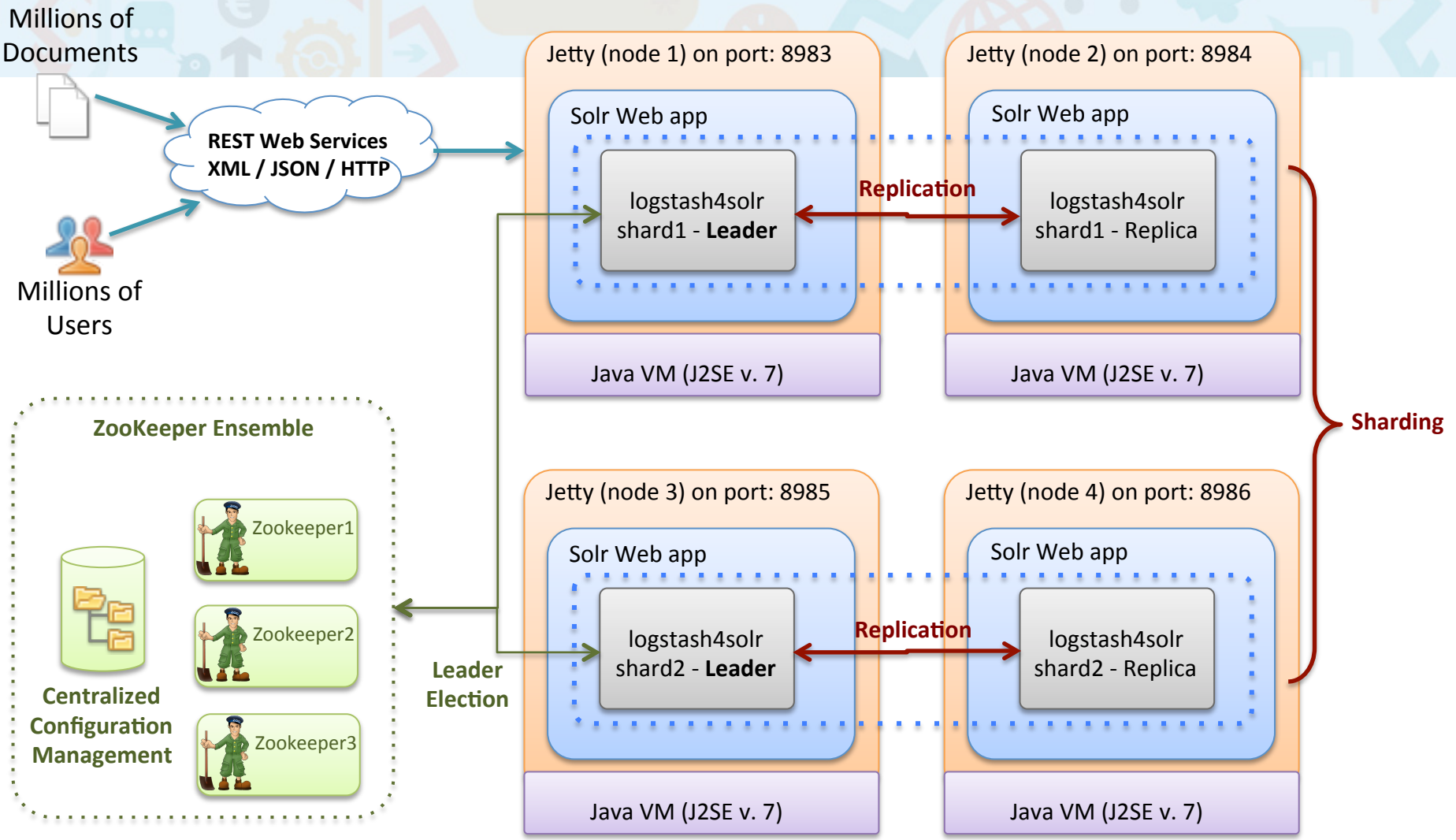
LucidWorks

- **ZooKeeper**: Distributed coordination service that provides centralized configuration, cluster state management, and leader election
- **Node**: JVM process bound to a specific port on a machine; hosts the LWS core application
- **Collection**: Search index distributed across multiple nodes; each collection has a name, shard count, and replication factor
- **Replication Factor**: Number of copies of a document in a collection
- **Shard**: Logical slice of a collection; each shard has a name, hash range, leader, and replication factor. Documents are assigned to one and only one shard per collection using a hash-based document routing strategy.
- **Replica**: LWS index that hosts a copy of a shard in a collection; behind the scenes, each replica is implemented as a Solr core
- **Leader**: Replica in a shard that assumes special duties needed to support distributed indexing in Solr; each shard has one and only one leader at any time and leaders are elected using ZooKeeper

SolrCloud High-level Architecture

LucidWorks

logstash4solr collection is distributed into 2 shards across 4 nodes with replication factor 2



Collection == Distributed Index

LucidWorks

A collection is a **distributed index** defined by:

- **named configuration** stored in ZooKeeper
- **number of shards**: documents are distributed across N partitions of the index
- **document routing strategy**: how documents get assigned to shards
- **replication factor**: how many copies of each document in the collection

Collections API:

```
curl "http://localhost:8983/solr/admin/collections?
   action=CREATE&name=logstash4solr&replicationFactor=2&
       numShards=2&collection.configName=logs"
```

Demo

LucidWorks

1. Start-up bootstrap node with embedded ZooKeeper
2. Add another shard
3. Add some replicas
4. Index some docs
5. Distributed queries
6. Knock-over a node, see cluster stay operational

ZooKeeper

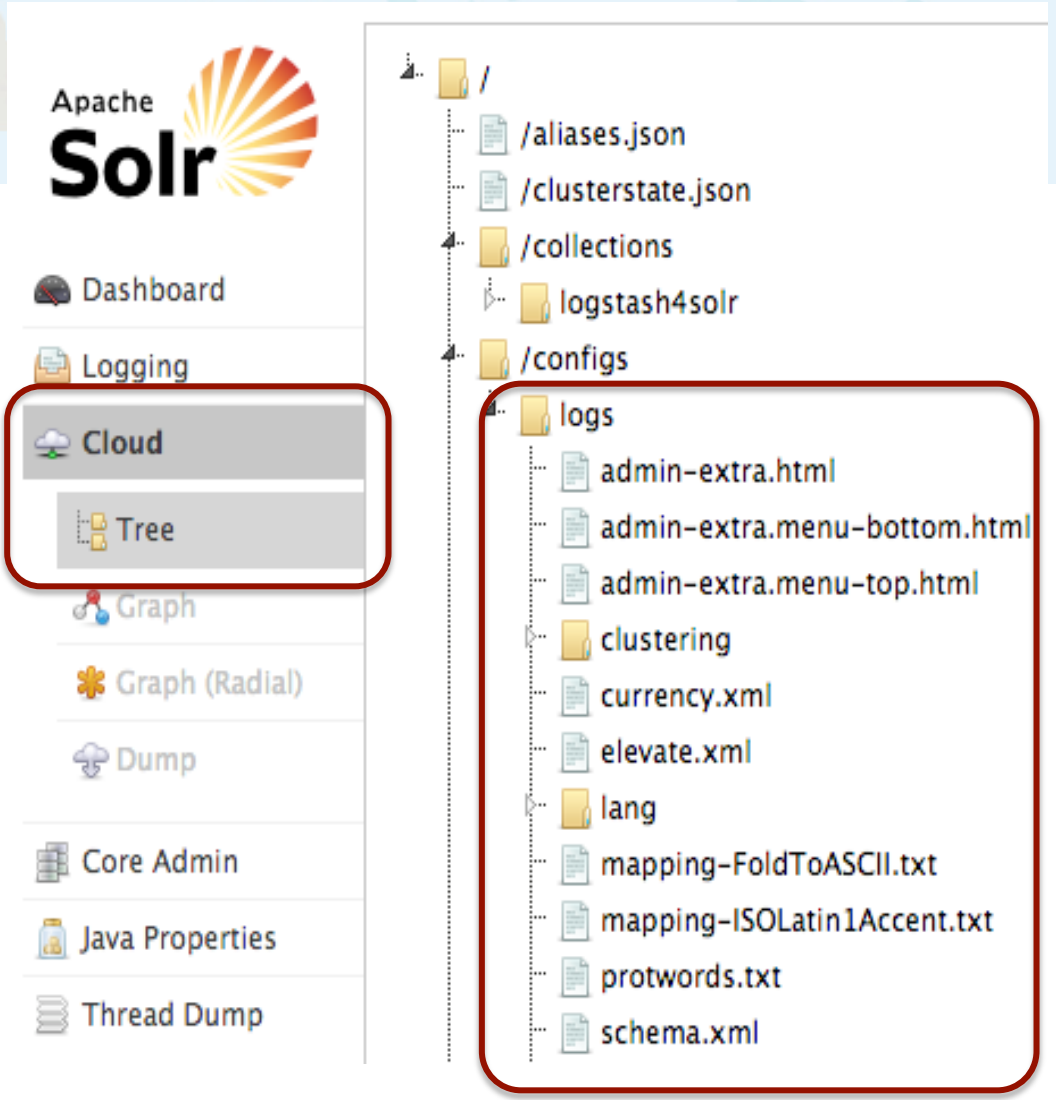
LucidWorks

- Is a very good thing ... clusters are a zoo!
- Centralized configuration management
- Cluster state management
- Leader election (shard leader and overseer)
- Overseer distributed work queue
- Live Nodes
  - Ephemeral znodes used to signal a server is gone
- Needs 3 nodes for quorum in production

ZooKeeper: Centralized Configuration

LucidWorks

- Store config files in ZooKeeper
- Solr nodes pull config during core initialization
- Config sets can be "shared" across collections
- Changes are uploaded to ZK and then collections should be reloaded

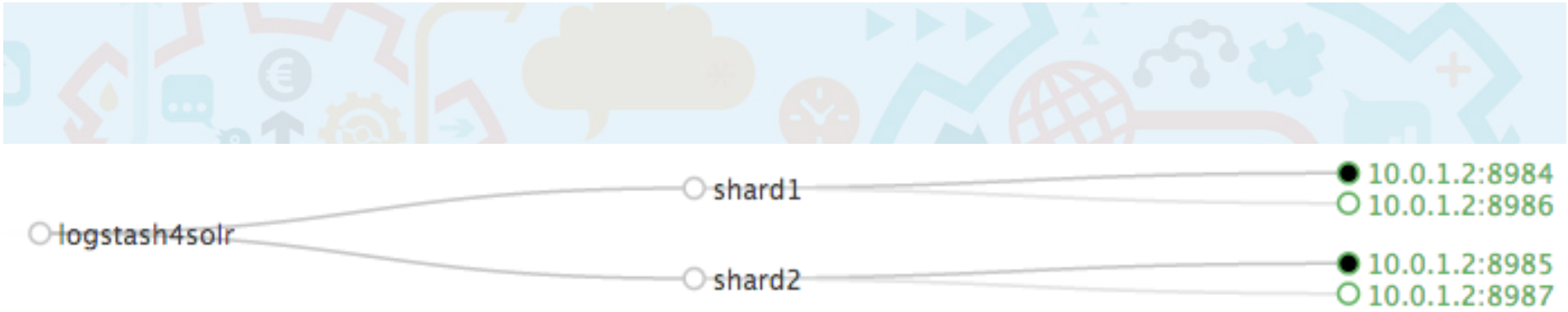


ZooKeeper: State management

LucidWorks

- Keep track of live nodes **/live_nodes** znode
  - ephemeral nodes
  - ZooKeeper client timeout
- Collection metadata and replica state in **/clusterstate.json**
  - Every core has watchers for /live_nodes and /clusterstate.json
- Leader election
  - ZooKeeper sequence number on ephemeral znodes

Sharding



- Collection has a fixed number of shards
  - existing shards can be split
- When to shard?
  - Large number of docs
  - Large document sizes
  - Parallelization during indexing and queries
  - Data partitioning (custom hashing)

Document Routing

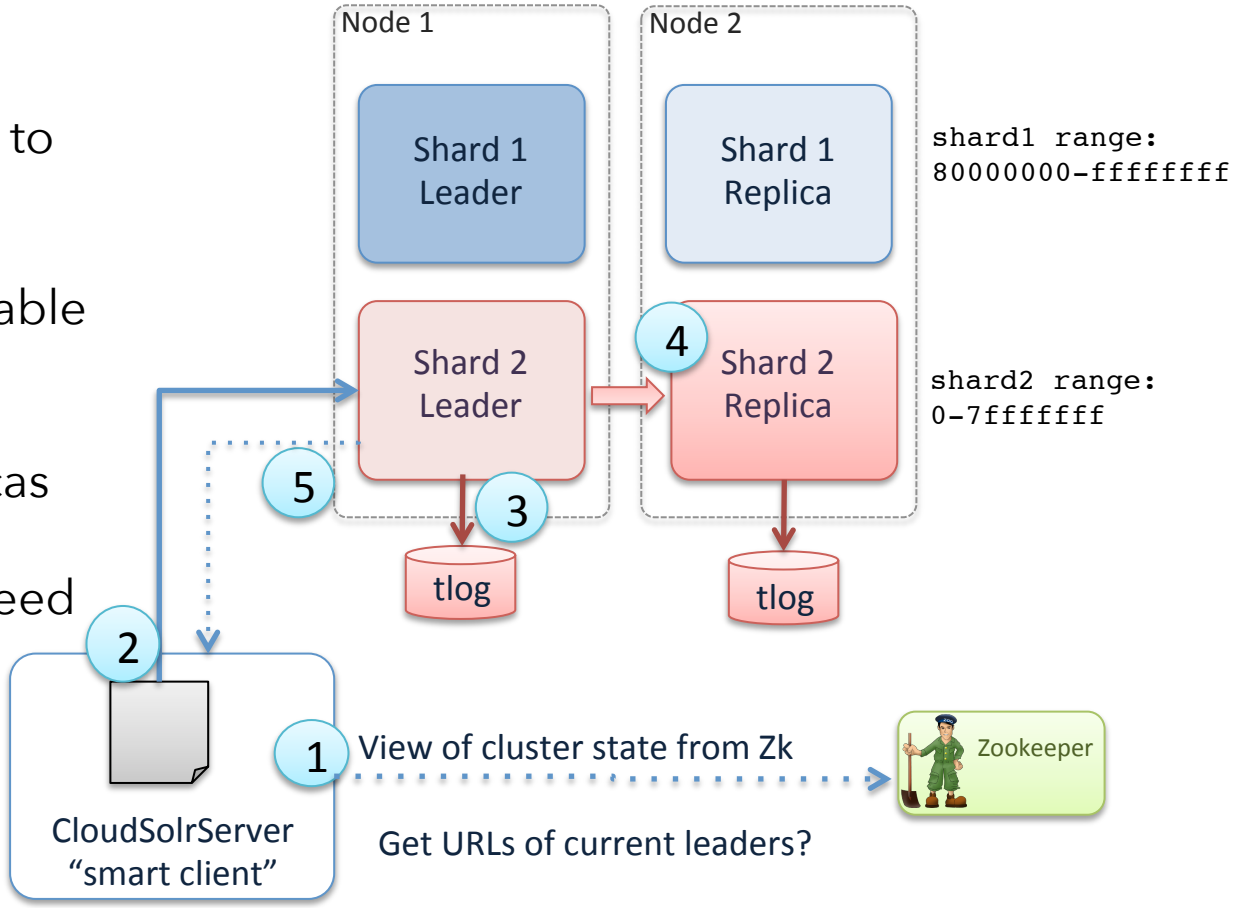
LucidWorks

- Each shard covers a hash-range
- Default: Hash ID into 32-bit integer, map to range
- Custom-hashing (example in a few slides)
- Tri-level: app!user!doc
- Implicit: no hash-range set for shards

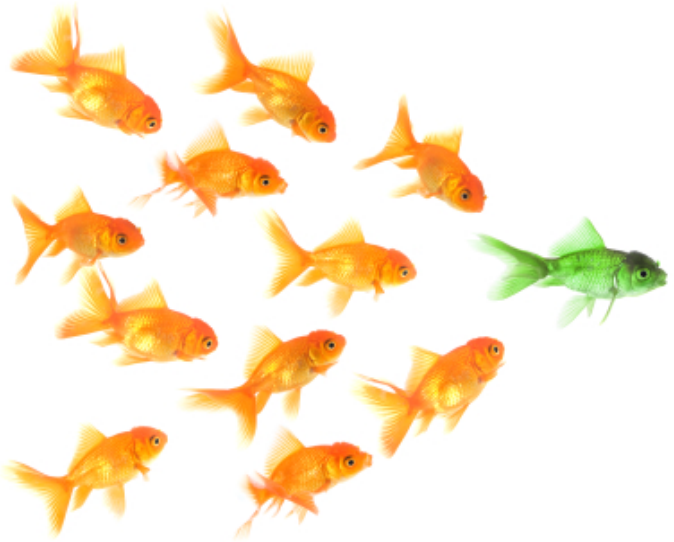
Distributed Indexing

LucidWorks

1. Get cluster state from ZK
2. Route document directly to leader (hash on doc ID)
3. Persist document on durable storage (tlog)
4. Forward to healthy replicas
5. Acknowledge write succeed to client



Shard Leader



- Additional responsibilities during indexing only! Not a master node
- Leader is a replica (handles queries)
- Accepts update requests for the shard
- Increments the _version_ on the new or updated doc
- Sends updates (in parallel) to all replicas

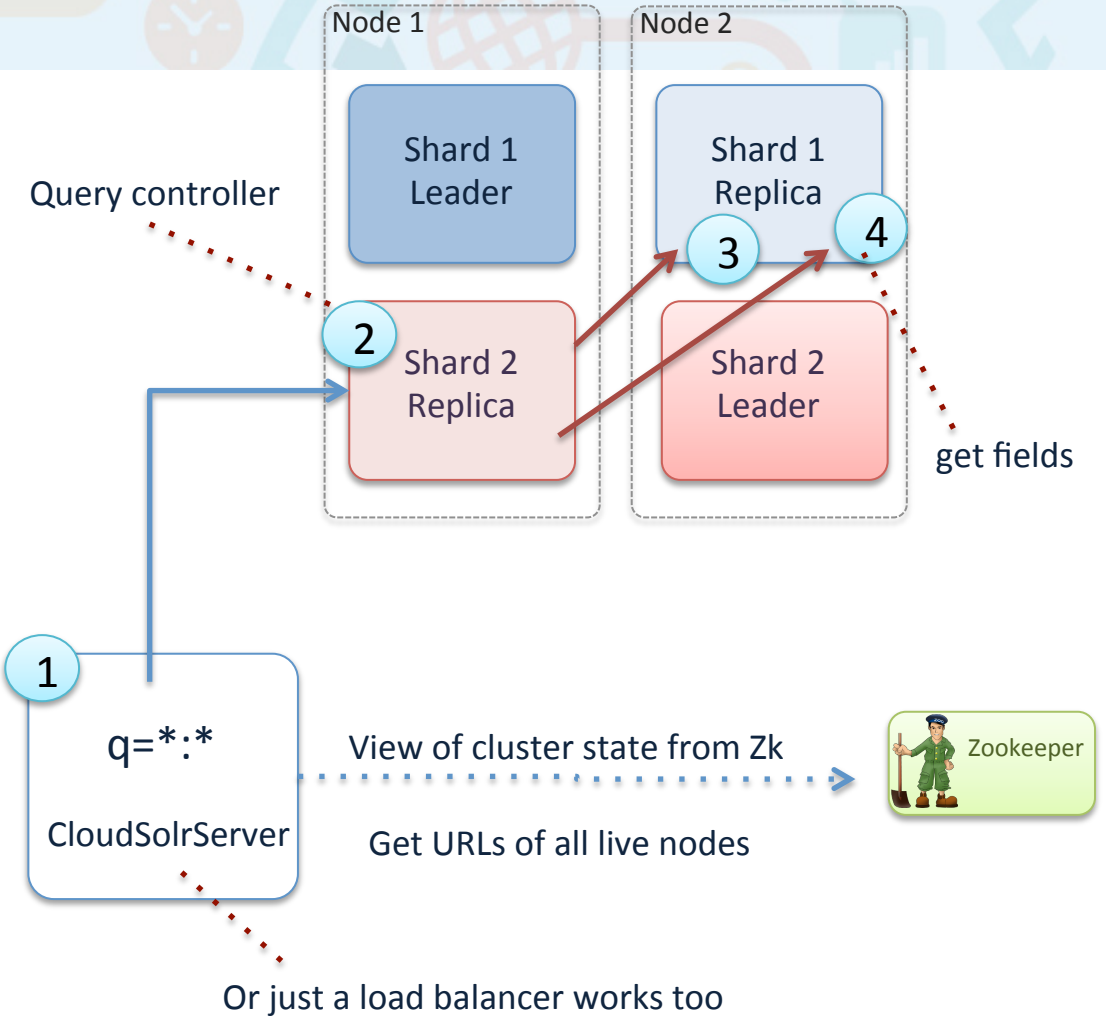
Replication

LucidWorks

- Why replicate?
  - High-availability
  - Load balancing
- How does it work in SolrCloud?
  - Near-real-time, not master-slave
  - Leader forwards to replicas in parallel, waits for response
  - Error handling during indexing is tricky

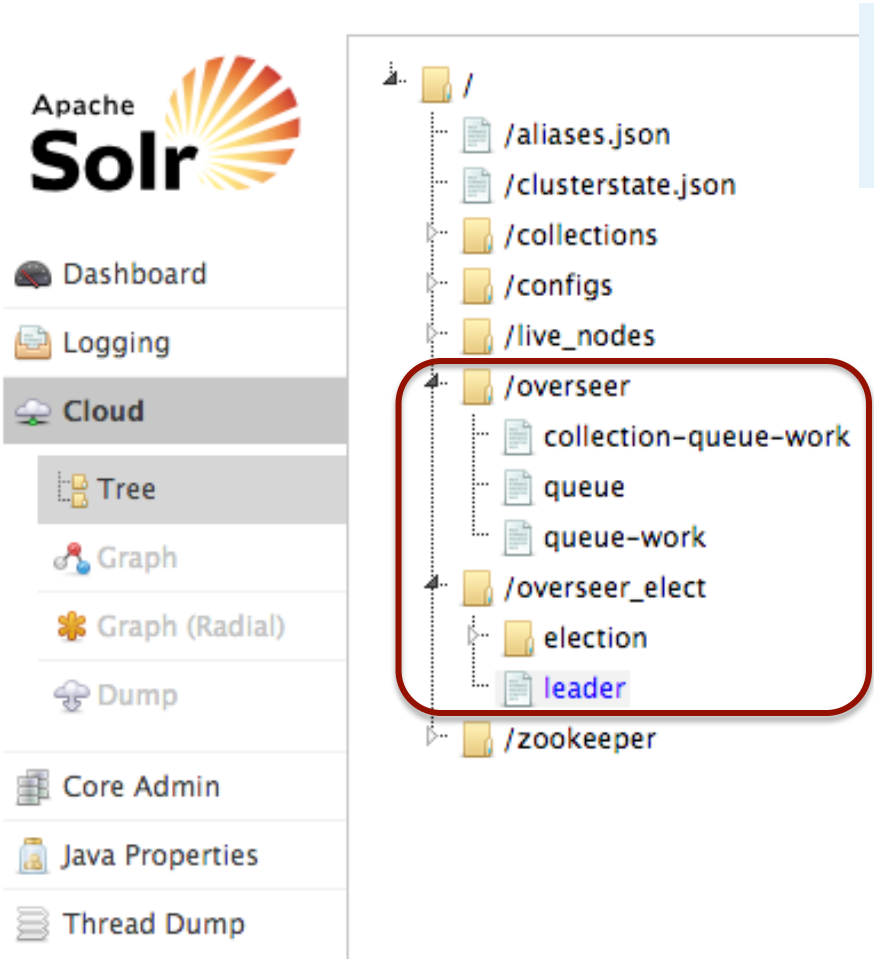
Distributed Queries

1. Query client can be ZK aware or just query thru a load balancer
2. Client can send query to any node in the cluster
3. Controller node distributes the query to a replica for each shard to identify documents matching query
4. Controller node sorts the results from step 3 and issues a second query for all fields for a page of results



Overseer

- What does it do?
  - Persists collection state change events to ZooKeeper
  - Controller for Collection API commands
  - Ordered updates
  - One per cluster (for all collections); elected using leader election
- How does it work?
  - Asynchronous (pub/sub messaging)
  - ZooKeeper as distributed queue recipe
  - Automated failover to a healthy node
  - Can be assigned to a dedicated node (SOLR-5476)



SolrCloud and CAP

LucidWorks

- A distributed system should be: **C**onsistent, **A**vailable, and **P**artition tolerant
  - CAP says pick 2 of the 3! (slightly more nuanced than that in reality)
- SolrCloud favors **consistency** over write-availability (CP)
  - All replicas in a shard have the same data
  - Active replica sets concept (writes accepted so long as a shard has at least one active replica available)
- No tools to detect or fix consistency issues in Solr
  - Reads go to one replica; no concept of quorum
  - Writes must fail if consistency cannot be guaranteed (SOLR-5468)

Scalability / Stability Highlights

LucidWorks

- All nodes in cluster perform indexing and execute queries; no master node
- Distributed indexing: No SPoF, high throughput via direct updates to leaders, automated failover to new leader
- Distributed queries: Add replicas to scale-out qps; parallelize complex query computations; fault-tolerance
- Indexing / queries continue so long as there is 1 healthy replica per shard

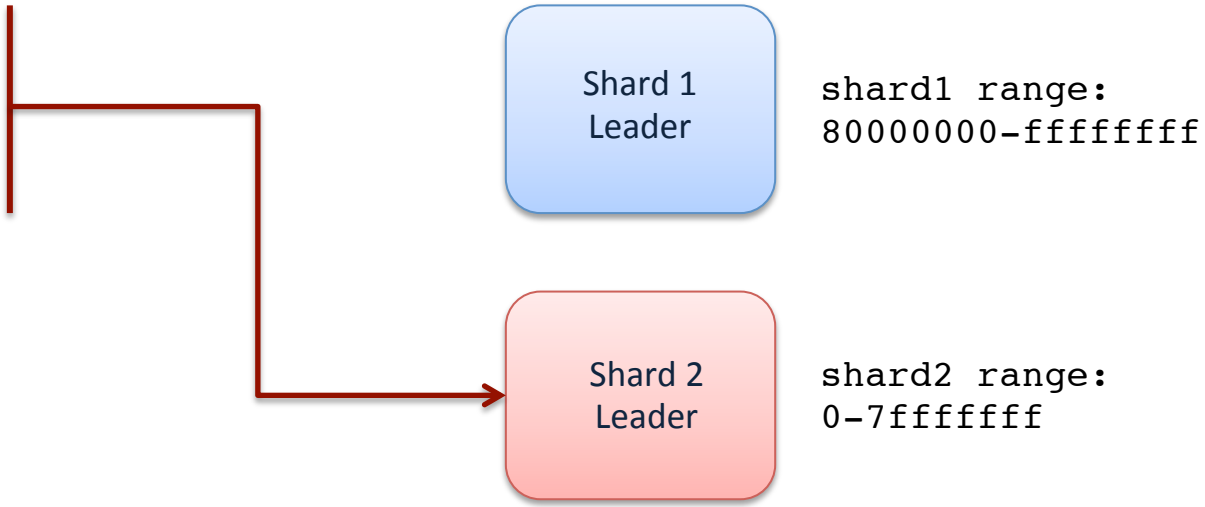
Custom Hashing

LucidWorks

- Route documents to specific shards based on a shard key component in the document ID
  - Send all log messages from the same system to the same shard
- Direct queries to specific shards

```
{
  "id" : "httpd!2",
  "level_s" : "ERROR",
  "lang_s" : "en",
   ...
},
```

Hash:
shardKey!docID



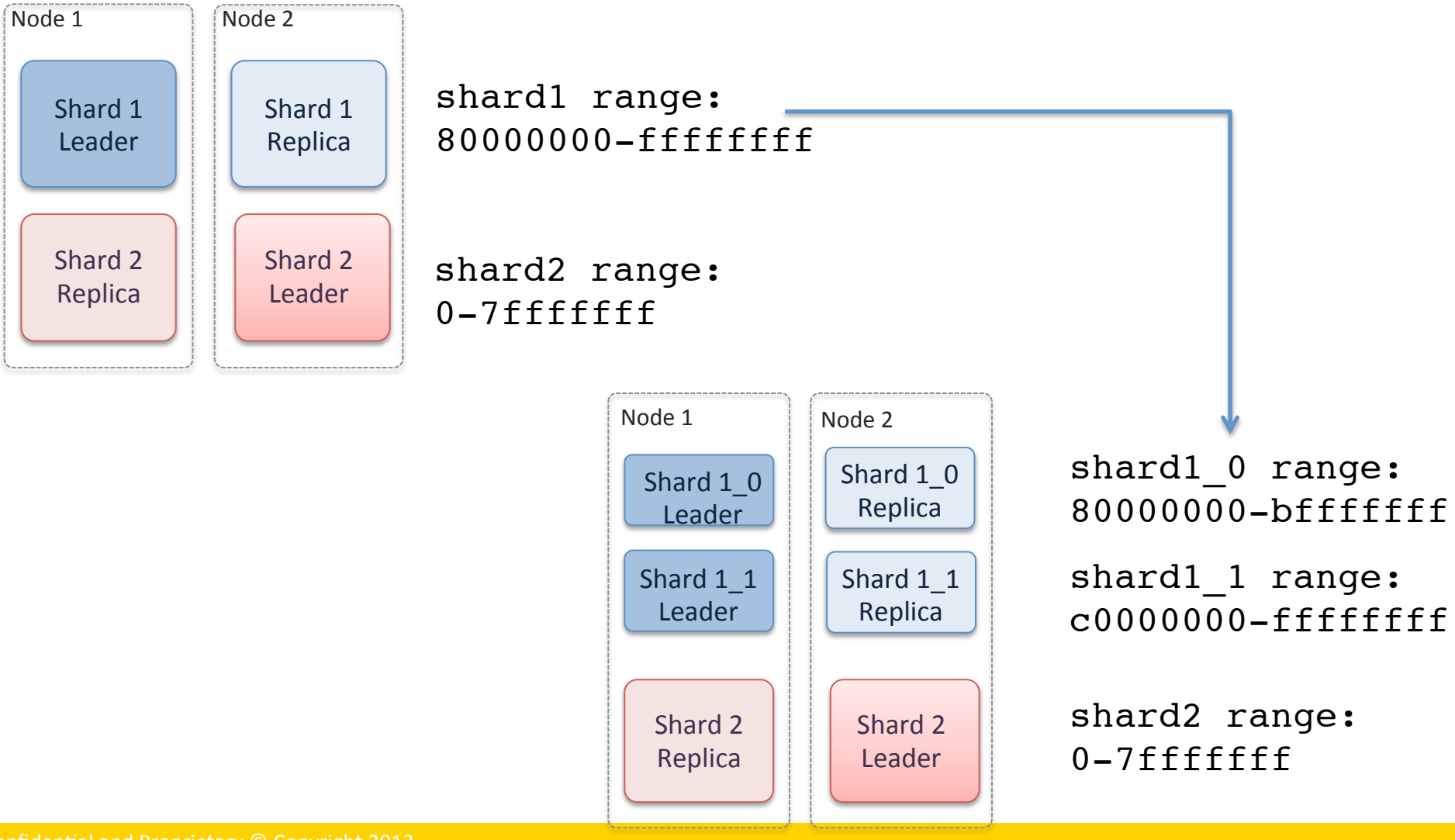
Custom Hashing Highlights

LucidWorks

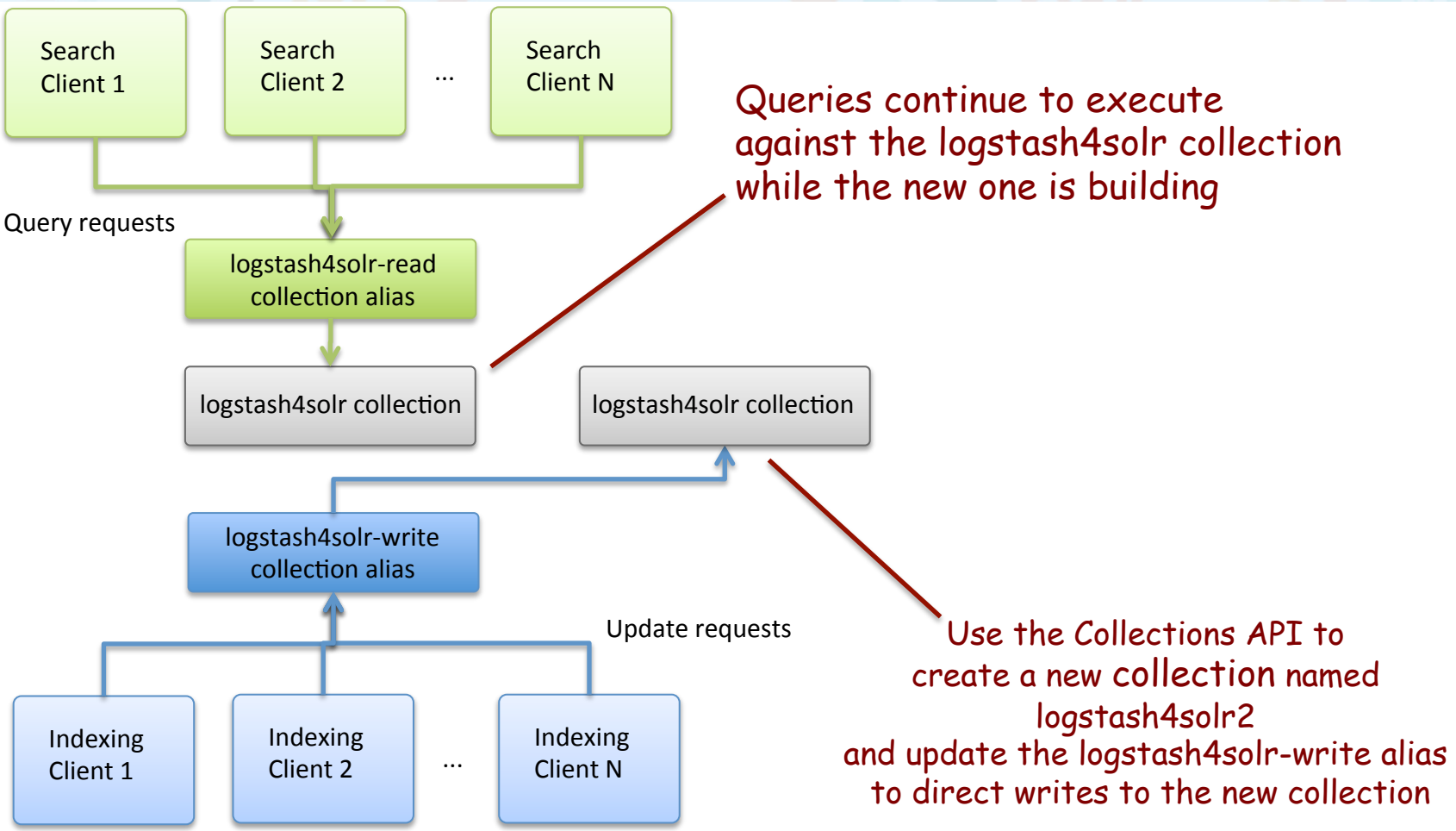
- Co-locate documents having a common property in the same shard
  - e.g. docs having IDs httpd!21 and httpd!33 will be in the same shard
- Scale-up the replicas for specific shards to address high query and/or indexing volume from specific apps
- Not as much control over the distribution of keys
  - httpd, mysql, and collectd all in same shard
- Can split unbalanced shards when using custom hashing

Shard Splitting

- Split range in half



Collection Aliases



Other Features / Highlights

LucidWorks

- **Near-Real-Time Search**: Documents are visible within a second or so after being indexed
- **Partial Document Update**: Just update the fields you need to change on existing documents
- **Optimistic Locking**: Ensure updates are applied to the correct version of a document
- **Transaction log**: Better recoverability; peer-sync between nodes after hiccups
- **HTTPS**
- Use **HDFS** for storing indexes
- Use **MapReduce** for building index (SOLR-1301)

What's Next?

LucidWorks

- Constantly hardening existing features
  - More Chaos monkey tests to cover tricky areas in the code
  - See Mark Miller's ApacheCon talk: http://sched.co/1bsUCOQ
- Large-scale performance testing; 1000's of collections, 100's of Solr nodes, billions of documents
- Splitting collection state into separate znodes (SOLR-5473)
- Collection management UI (SOLR-4388)
- Cluster deployment / management tools
  - My talk tomorrow: http://sched.co/1bsKUMn
- Ease of use!
  - Please contribute to the mailing list, wiki, JIRA

Wrap-up

LucidWorks

- LucidWorks: http://www.lucidworks.com
- SiLK: http://www.lucidworks.com/lucidworks-silk/
- Solr In Action: http://www.manning.com/grainger/
- Connect: @thelabdude / tim.potter@lucidworks.com

Questions?