# SolrCloudPlanning

## SolrCloud

## Developer Section

### Tools

There is a very useful zookeeper plugin for eclipse at http://www.massedynamic.org/mediawiki/index.php?title=Eclipse_Plug-in_for_ZooKeeper

### TODO

- when this stull is merged to trunk, integrate the user documentation above into DistributedSearch
  - optionally allow user to query by collection
  - optionally allow user to query by multiple collections (assume schemas are compatible)
- user SolrJ support for getting server list to query from zk
- optionally return partial results if we can't query all of the shards
  - this includes a new code path where there may be no servers up for a shard (i.e. no exception from the LB server is thrown because we never try)
  - seems to require propagating more info into the search handler... need to know what logical shard is missing? if it's just a list of
    - URLs (the shard addresses) then servers that aren't up would just be represented by a blank space: localhost:8983,,localhost:7574
- when using master/slave replication, optionally remove the periodic polling that slaves do and replace with a watch on a znode that can immediately ping/pull an index when the version changes. Seems like low priority since index version polls can be frequent with low overhead.

### High level design goals

These are long term goals for SolrCloud. Many of these features will not be developed in the first versions, but we're designing for the long haul.

#### High Availability and Fault Tolerance

No external load balancer should be required. We should eliminate any single points of failure (i.e. start with a design that will allow us to add this feature at a later point with minimal changes to things like the zookeeper schema)

#### Cluster resizing and rebalancing

To grow the cluster or to rebalance due to hotspots, shards should be resizable. Pieces of existing shards should be able to be split off and assigned to new servers.

#### Clients should not have to know about cluster layout.

A simple client (like a browser) should be able to hit any solr search server in the cluster with a request, and that search server should be able to execute a distributed search against the cluster as a whole, including load balancing and failover, to return the results to the client.

#### Open API

The zookeeper schema should be well defined and public, allowing other software components other than the master node to inspect and change the cluster via zookeeper. A task like creating a new collection eventually be achievable by creating the correct znodes/files in zookeeper, w/o having to talk to any solr servers.

#### Support various levels of custom clusters

Support various splits between how much the user manages and how much solr manages. One could have a set of servers in zookeeper with defined indexes (shards or complete copies) and want to just use the client search capabilities. One may also want a traditional master/slave relationship, even if more advanced options are available.

#### Support user specified partitioning

Partitioning of documents by geographic region, time, user, etc, brings huge performance benefits by allowing only a part of the cluster to be queried. This should be an option even in conjunction with the most advanced modes of operation (automatic document->shard assignment, index rebalancing, no single points of failure, etc) presumably by allowing user-specified hashcodes for documents.

### Entities we want to model or record in zookeeper:

**host**

- The actual physical (or virtual, as it may be) machine
- operating system, RAM, disk
- some sort of metric for what level of load should be placed on this machine

**node**

- a single JVM running one or more solr cores

**collection**

- A collection of documents sharing the same schema

**shard**

- A physical piece of a collection, usually corresponding to a single Lucene index. A shard may or may not have replicas (copies), and may partially overlap with other shards. A shard has a specific address (i.e. replicas are considered different shards).

**slice**

- A logical subset of the collection. In the simplest case, a slice will be represented one more more shards with the same content (replicas).

**core**

- A solr core (is there a better name we can use for this?)
- If a core and a shard have a one-to-one mapping, they could be redundant.

**role**

- is a core a master, a search node, a spell-checker node, etc
- we probably want a generic way to map from role to different configs or config overrides

**network topology**

- switch, rack, data center, etc

# Zookeeper Schema

## Model and State

There seem to logically be two different types of data that we want contained in zookeeper:

**Model** - represents the goal / targets of the cluster and the systems in it.

**State** - represents the actual current state of the cluster and the systems comprising it.

A manager can make well-defined changes to the model, and the servers should respond such that eventually their state matches that of the model.

## Multiple Solr clusters

A Solr cluster should be able to use an existing zookeeper cluster, and multiple solr clusters should be able to coexist on a single zookeeper cluster.

One idea: This seems easiest to achieve with a configuration URL that points to the zookeeper cluster and includes any arbitrary prefix.

## Shard Identification

Two ways of identifying shards are needed.

For complex cluster features in the future, Solr will need to know where to find specific documents. The documents a shard contains can be defined by a range of ids - the ids in this case being hash codes of something else like the unique key field, or user supplied. See the amazon dynamo paper and other descriptions of consistent hashing.

In the most basic case though, we will be dealing with indexes built outside the cluster. In these cases, we won't know what documents are in what shards, but we still need a way to identify the fact that one shard is simply a replica of another shard.

## Layout

This represents brainstorming on what the virtual filesystem (i.e. schema) in zookeeper could look like.

mycluster/hosts

- 192.168.0.10
- *#how to identify hosts? hostname? IP?* Hosts may have more than one IP - do we want to use multiple?

mycluster/configs/collection1_config/v1/

- solrconfig.xml, schema.xml, stopwords.txt, etc

collections/collection1/

- config=collection1_config *#explicitly specified so it allows multiple collections to use the same config*

#Have the collection directly point to the shard URLs? (method #1)

/collections/collection1/shards

- localhost:8983/solr/collection1=shardX,shardY,shardZ
- localhost:7574/solr/collection1=shardX,shardZ

#Or perhaps just specify the nodeid (what would that be? localhost:8983? localhost:8983/solr?) and make it such that "collection1" is always implied because that is the collection name? Perhaps that could turn out to be too limiting though, if someone is trying to do federated search? Perhaps list the nodes belonging to a collection, with the shards each node has? (method #2)

/collections/collection1/nodes/localhost:8983/

- shards/ #the shards that the node localhost:8983 currently has that belong to this collection
- X=/solr/collection1 #the relative URL from the nodeid
- Y=/solr/collection1
- Z=/solr/collection1
- # We should be able to record the version of the index (shard), the last time it was updated, etc.

# Method #3 could move the shards listing for each node to the model of the node itself.

## Cluster Options

### search only

The most degenerate form of a cluster we would support - the user manages everything, tells zk where shards are, solr uses that for completing distributed search requests. Master/slave relationships are not exposed.

There could perhaps be a flag or role that indicates that a node is meant to be hit by user requests (as opposed to sub-requests as a result of a distributed search). This would allow to specialize servers based on function.

In order to further support user partitioning, users should still be able to specify a subset of the shards to query. Perhaps even support optional shard groups, so a user could specify that only shards covering SF or NYC should be queried?

Distributed search is optional - a cluster could simply be a number of servers with the same shard.

### local config

This could be an option in conjunction with any other cluster model - solr need not load it's config from zookeeper. One advantage this has is that it breaks the startup dependency on zookeeper - one could start up a solr server and index data to it w/o zookeeper being up. The registration of this node in zookeeper could be asynchronous - it happens when we do finally connect to zk.

A variant on this could copy config files out of zookeeper to local storage to provide the benefit of disconnected operation.

### search and replication

In addition to "search only" this models the master slave relationship. Certain shards are marked as a master, and slaves will automatically enable replication to pull from the correct master.

Key/doc partitioning is still done by the user (i.e. shards are opaque and solr will not know if documentA belongs in shardX or shardY)

## Example Scenarios

#### Simplest Bootstrap

How to create a new solr cluster + collection? Easiest would be to create one from an existing non-zookeeper server.

Method 1: Environment variable (-Dboostrap_collection=...) that could create files needed for a new collection, including copying the local config files into zookeeper before the core is created, then a zk-enabled core could be started. This would also bootstrap the entire cluster (default name = solr) if it had not been done yet.

java -Dbootstrap_collection=collection_name -DzkHost=... -jar start.jar

This method may be the most user friendly for first time users from the solr example.

Method 2: A URL (a core container command) for moving an existing core to zookeeper (and creating a new collection). The core could be reloaded after the necessary files are moved.

Method 3: A separate utility jar for creating a new cluster / collection. (Downsides - need to set up classpath correctly, etc.)

#### Adding existing shard

User starts up a shard and says it has "shardX" (assume there is not already an existing shardX in the cluster).

## Other Questions

Is there a single master (cluster manager, not solr master) per collection, or a master for all collections in the cluster?

How do we build an index and test it out before adding it to a collection? We want to be part of the collection so we can get the config, but we don't want searchers to use the index yet. Perhaps have a shard state that could indicate this.

Have some sort of command list that every server should execute before certain actions? (could involve hitting URLs, executing system commands, etc)?

## Distributed Search

### Basic Distributed Search

The state of the cluster will be read at startup. Changes to the state will be immediately reflected in the internal representation via zookeeper watches. Once a cluster state has been built, a connection to zookeeper is not needed to serve requests (i.e. it can work when disconnected from zk).

implementation detail: certain information about the internal representation of the cluster should be copied at the start of a request and probably shouldn't change during the request. This probably includes the shards that will be included in the request (we don't want that changing between phases of a request), and the nodes we are querying for those shards. Someone may take a node out of service, or zookeeper may have marked the node as failed, but we can simply continue using the normal request/failover logic for the duration of that distributed request.

Connection refused errors from solr_server->solr_server (or other errors that we believe would not result in an error if executed on a different node) should result in failover behavior (re-request a different shard). It can be a local policy decision to not try that node again for a certain amount of time after so many of these errors. Zookeeper does not need to be updated with this info (but could be in the future).

Node failures should not always fail the distributed search. It should be configurable whether that aborts the search or returns with results from all non-failing nodes along with a list of the nodes that failed. In cases with huge document collections, a missing few percent of total documents may not be worth stopping the user's progress; yet you would want to know about the failure so you could indicate that the results are incomplete.

### Timeouts

Zookeeper ephemeral znodes can be used to determine what servers are available for requests.

Q: if zookeeper dies and comes back up, does it come back with all the ephemeral nodes? If all the ephemeral nodes are deleted, we need to disregard and continue using our last internal model.

solr_server->solr_server requests may result in a timeout after "shard-socket-timeout". If a flag indicating partialResults is set, we should not retry a different shard. If a flag indicating partialResults is not set, we fail the request, or retry a different shard, depending on a new "retryOnTimeout" flag. After a configurable number of timeouts, where other shards did not timeout, we can mark the node as "slow" or "timedout" in zookeeper. A leader could optionally act on that information to remove the node or reallocate resources.

# Resources

http://sourceforge.net/mailarchive/forum.php?forum_name=bailey-developers

http://www.allthingsdistributed.com/2007/10/amazons_dynamo.html

https://svn.apache.org/repos/asf/lucene/solr/branches/cloud/