CollectionDistribution

This document describes the ssh/rsync based replication available since Solr1.1 **This mechanism only works on systems that support removing open hard links**.

⚠️ These Scripts were superseded by the ReplicationHandler Java implementation of index replication that works over HTTP and was introduced in Solr1.4 , and are no longer actively maintained. ⚠️

For More details, please see...

- SolrCollectionDistributionScripts
- SolrCollectionDistributionStatusStats
- SolrCollectionDistributionOperationsOutline

Introduction

Solr distribution is similar in concept to database replication. All collection changes come to one master Solr server. All production queries are done against query slaves. Query slaves receive all their collection changes indirectly as new versions of a collection which they pull from the master. These collection downloads are polled for on a cron'd basis.

A collection is a directory of many files. Collections are distributed to the slaves as snapshots of these files. Each snapshot is made up of hard links to the files so copying of the actual files is not necessary when snapshots are created. Lucene only *significantly* rewrites files following an optimization command. Generally, a file once written, will change very little if at all. This makes the underlying transport of rsync very useful. Files that have already been transfered and have not changed do not need to be re-transferred with the new edition of a collection.

- Introduction
- Terminology
- The Snapshot and Distribution Process
- Snapshot Directories
- Solr Distribution Scripts
- Solr Distribution related Cron Jobs
  - snapcleaner
  - snappuller and snapinstaller
- Commit and Optimization
- Distribution and Optimization

Terminology

Term	Definition
Collection	A Lucene collection is a directory of files. These comprise the indexed and returnable data of a Solr search repository.
Distribution	The copying of a collection from the master to all slaves. Distribution of a collection from the master to all slaves takes advantage of Lucene's index file structure. (This same feature enables fast incremental indexing in Lucene.)
Inserts and Deletes	As inserts and deletes occur in the collection the directory remains unchanged. Documents are always inserted into newly created files. Documents that are deleted are not removed from the files. They are flagged in the file, **deletable**, and are not removed from the files until the collection is **optimized**.

Master & Slave	The Solr distribution system uses the master/slave model. The master is the service which receives all updates initially and keeps everything organized. Solr uses a single update master server coupled with multiple query slave servers. All changes (such as inserts, updates, deletes, etc.) are made against the single master server. Changes made on the master are distributed to all the slave servers which service all query requests from the clients.
Update	An update is a single change request against a single Solr instance. It may be a request to delete a document, add a new document, change a document, delete all documents matching a query, etc. Updates are handled synchronously within an individual Solr instance.
Optimization	A Lucene collection must be optimized periodically to maintain query performance. Optimization is run on the master server only. An optimized index will give you a performance gain at query time of *at least* 10%. This gain may be more on an index that has become fragmented over a period of time with many updates and no optimizations. Optimizations require a **much** longer time than does the distribution of an optimized collection to all slaves. During optimization, a collection is compacted and all segements are merged. New secondary segment(s) are created to contain documents inserted into the collection after it has been optimized.
Segments	The number of files in a collection.
mergeFactor	Controls the number of files (segments). For example, when mergeFactor is set to 2, existing documents remain in the main segment, while all updates are written to a single secondary segment.
Snapshot	The snapshot directory contains hard links to the data files. Snapshots are distributed from the master server when the slaves pull them, "smartcopying" the snapshot directory that contains the hard links to the most recent collection data files.

The Snapshot and Distribution Process

These are the sequential steps that Solr runs:

1. The **snapshooter** command takes snapshots of the collection on the master. It runs when invoked by Solr after it has done a commit or an optimize.
2. The **snappuller** command runs on the query slaves to pull the newest snapshot from the master. This is done via rsync in daemon mode running on the master for better performance and lower CPU utilization over rsync using a remote shell program as the transport.
3. The **snapinstaller** runs on the slave after a snapshot has been pulled from the master. This signals the local Solr server to open a new index reader, then auto-warming of the cache(s) begins (in the new reader), while other requests continue to be served by the original index reader. Once auto-warming is complete, Solr retires the old reader and directs all new queries to the newly cache-warmed reader.
4. All distribution activity is logged and written back to the master to be viewable on the distribution page of its GUI.
5. Old versions of the index are removed from the master and slave servers by a cron'd **snapcleaner**.

If you are building an index from scratch, distribution is the final step of the process. For detailed steps, see CollectionRebuilding

Manual copying of index files is not recommended; however, running distribution commands manually (i.e., not relying on crond to run them) is perfectly fine.

Snapshot Directories

Snapshots directories are in the format, snapshot.*yyyymmddHHMMSS*.

All the files in the index directory are hard links to the latest snapshot. This technique has these advantages:

- Can keep multiple snapshots on each host without the need to keep multiple copies of index files that have not changed.
- File copying from master to slave is very fast.
- Taking a snapshot is very fast as well.

Solr Distribution Scripts

- For the Solr distribution scripts, the name of the index directory is defined either by the environment variable **data_dir** in the configuration file **solr/conf/scripts.conf** or the command line argument **-d**. It should match the value used by the Solr server which is defined in **solr/conf/solrconfig.xml**.
- All Solr collection distribution scripts are bundled within a Solr release and reside in the directory **solr/src/scripts**. It is suggested that the be installed in a **solr/bin/** directory.
- Collection distribution scripts create and prepare for distribution a snapshot of a search collection after each **commit** and **optimize** request if the *postCommit* and *postOptimize* event listener is configured in **solr/conf/solrconfig.xml** to execute **snapshooter**.
- The **snapshooter** script creates a directory *snapshot.<ts>*, where <ts> is a timestamp in the format, yyyymmddHHMMSS. It contains hard links to the data files.
- Snapshots are distributed from the master server when the slaves pull them, "smartcopying" the snapshot directory that contains the hard links to the most recent collection data files.
- For usage arguments and syntax see SolrCollectionDistributionScripts

Name	Description
snapshooter	Creates a snapshot of a collection. **Snapshooter** is normally configured to run on the master Solr server when a **commit** or **optimize** happens. **Snapshooter** can also be run manually, but one must make sure that the index is in a consistent state, which can only be done by pausing indexing and issuing a commit.

snappuller	A shell script that runs as a cron job on a slave Solr server. The script looks for new snapshots on the master Solr server and pulls them.
snappuller-enable	Creates the file, **solr/logs/snappuller-enabled**, whose presence enables **snappuller**.
snapinstaller	**Snapinstaller** installs the latest snapshot (determined by the timestamp) into the place, using hard links (similar to the process of taking a snapshot). Then **solr/logs/snapshot.current** is written and scp (secure copied) back to the master Solr server. **Snapinstaller** then triggers the Solr server to open a new **Searcher**.
snapcleaner	Runs as a cron job to remove snapshots more than a configurable number of days old or all snapshots except for the most recent *n* number of snapshots. Also can be run manually.
rsyncd-start	Starts the rsyncd daemon on the master Solr server which handles collection distribution requests from the slaves.
rsyncd daemon	Efficiently synchronizes a collectionbetween master and slavesby copying only the files that actually changed. In addition, rsync can optionally compress data before transmitting it.
rsyncd-stop	Stops the rsyncd daemon on the master Solr server. The stop script then makes sure that the daemon has in fact exited by trying to connect to it for up to 300 seconds. The stop script exits with *error code 2* if it fails to stop the rsyncd daemon.
rsyncd-enable	Creates the file, **solr/logs/rsyncd-enabled**, whose presence allows the rsyncd daemon to run, allowing replication to occur.
rsyncd-disable	Removes the file, **solr/logs/rsyncd-enabled**, whose absence prevents the rsyncd daemon from running, preventing replication.

Solr Distribution related Cron Jobs

The distribution process is automated via the use of cron jobs. The cron jobs should run under the user id that the Solr server is running under.

snapcleaner

snapcleaner should be run out of cron at the regular basis to clean up old snapshots. This should be done on both the master and slave Solr servers. For example, the following cron job runs everyday at midnight and cleans up snapshots 8 days and older:

```
0 0 * * * <solr.solr.home>/solr/bin/snapcleaner -D 7
```

Additional cleanup can always be performed on-demand by running **snapcleaner** manually.

snappuller and snapinstaller

On the slave Solr servers, **snappuller** should be run out of cron regularily to get the latest index from the master Solr server. It is a good idea to also run **snapinstaller** with **snappuller** back-to-back in the same crontab entry to install the latest index once it has been copied over to the slave Solr server. For example, the following cron job runs every 5 minutes to keep the slave Solr server in sync with the master Solr server:

```
0,5,10,15,20,25,30,35,40,45,50,55 * * * * <solr.solr.home>/solr/bin/snappuller;<solr.solr.home>/solr/bin
/snapinstaller
```

NB: Modern cron allows this to be shortened to `*/5 * * * * ...`.

Commit and Optimization

On a very large index, adding even a few documents then running an optimize means rewriting the complete index. This consumes a lot of disk I/O and impacts query performace. Optimizing a very large index may even involve copying the index twice the current code for merging one index into another calls optimize at the beginning *and* the end. If some docs have been deleted, the first optimize call will rewrite the index even before the second index is merged.

Optimization is an I/O intensive process, as the entire index is read and re-written in optimized form. Anecdotal data shows that optimizations on modest server hardware can take around 5 minutes per GB, although this obviously varies considerably with index fragmentation and hardware bottlenecks. We do not know what happens to query performance on a collection that has not been optimized for a long time. We *do* know that it will get worse as the collection becomes more fragmented, but how much worse is very dependent on the manner of updates and commits to the collection.

We are presuming optimizations should be run once following large *batch-like* updates to the collection and/or once a day.

Distribution and Optimization

An optimization on the master takes several minutes. Distribution of a newly optimized collection takes only a little over a minute. During optimization the machine is under load and does not process queries very well. Given a schedule of updates being driven a few times an hour to the slaves, we cannot run an optimize with every committed snapshot. We do recommend that an optimize be run on the master at least once a day.

Copying an optimized collection means that the **entire** collection will need to be transferred during the next snappull. This is a large expense, but not nearly as huge as running the optimize everywhere. In a three-slave one-master configuration, distributing a newly-optimized collection takes approximately 80 seconds *total*. Rolling the change across a tier would require approximately ten minutes per machine (or machine group). If this optimize were rolled across the query tier, and if each collection being optimized were disabled and not receiving queries, a rollout would take at least twenty minutes and potentially an hour and a half. Additionally, the files would need to be synchronized so that the *following* rsynch snappull would not think that the independently optimized files were different in any way. This would also leave the door open to independent corruption of collections instead of each being a perfect copy of the master.

Optimizing on the master allows for a straight-forward optimization operation. No query slaves need to be taken out of service. The optimized collection can be distributed in the background as queries are being normally serviced. The optimization can occur at any time convenient to the application providing collection updates.