DistributedSearch

<!> Solr1.3

What is Distributed Search?

When an index becomes too large to fit on a single system, or when a single query takes too long to execute, an index can be split into multiple shards, and Solr can query and merge results across those shards.

If single queries are currently fast enough and one simply wishes to expand the capacity (queries/sec) of the search system, then standard whole index replication should be used.

This page covers the non-SolrCloud approach to distributed search, valid for all versions 1.3 and later. SolrCloud was introduced in Solr 4.0 and has many advancements that make distributed search easier. Please see the SolrCloud page for more information.

- What is Distributed Search?
- Distributed Searching
- Distributed Searching Limitations
  - Distributed Deadlock
- Distributed Indexing
- Distributed Search Example

Distributed Searching

The presence of the **shards** parameter in a request will cause that request to be distributed across all shards in the list. The syntax of **shards** is host:port /base_url[,host:port/base_url]* A sharded request will go to the standard request handler (not necessarily the original); this can be overridden via **shards.qt** . Since SOLR-3134 it is possible to obtain numFound, maxScore and time per shard in a distributed search query. Use **shards.info=true** to enable this feature. The **shards.tolerant=true** parameter includes error information if available. (SolrCloud can handle this for you in a more transparent way).

Currently, only query requests will be distributed. This includes requests to the standard request handler (and subclasses such as the dismax request handler), and any other handler (org.apache.solr.handler.component.SearchHandler) using standard components that support distributed search.

The current components that support distributed search are

- The Query component that returns documents matching a query
- The Facet component, for facet.query and facet.field requests where facets are sorted by count (the default). Solr 1.4 and later also support sorting by name. Distributed date faceting is supported since SOLR-1709] and Solr4.0. See issue [https://issues.apache.org/jira/browse/SOLR-1709.
- The Highlighting component
- The Stats component
- The Spell Check Component
- The Terms Component
- The Term Vector Component
- The Debug component
- The Grouping component (From ⚠️ Solr3.5 and from version ⚠️ Solr4.0. Currently group.truncate and group.func are the only parameters that aren't supported for distributed searches.)

See also WritingDistributedSearchComponents

Distributed Searching Limitations

- Documents must have a unique key and the unique key must be stored (stored="true" in schema.xml)
- **The unique key field must be unique across all shards.** If docs with duplicate unique keys are encountered, Solr will make an attempt to return valid results, but the behavior may be non-deterministic.
- No distributed idf (see http://wunderwood.org/most_casual_observer/2007/04/progressive_reranking.html) (Also see https://issues.apache.org/jira/browse/SOLR-1632 for some new work on this feature.)
- Doesn't support Join – (see https://issues.apache.org/jira/browse/LUCENE-3759)
- Doesn't support pivot facing – (see https://wiki.apache.org/solr/HierarchicalFaceting#Pivot_Facets))
- The index could change between stages, e.g. a document that matched a query and was subsequently changed may no longer match but will still be retrieved.
- Makes it more inefficient to use a high "start" parameter. For example, if you request start=500000&rows=25 on an index with 500,000+ docs per shard, this will currently result in 500,000 records getting sent over the network from the shard to the coordinating Solr instance. If you had a single-shard index, in contrast, only 25 records would ever get sent over the network. (Granted, setting start this high is not something many people need to do.)

Distributed Deadlock

Each shard may also serve top-level query requests and then make sub-requests to all of the other shards. In this configuration, care should be taken to ensure that the max number of threads serving HTTP requests in the servlet container is greater than the possible number of requests from both top-level clients and other shards (the solr example server is already configured correctly). If this is not the case, a distributed deadlock is possible.

Consider the simplest case of two shards, each with just a single thread to service HTTP requests. Both threads could receive a top-level request concurrently, and make sub-requests to each other. Because there are no more remaining threads to service requests, the servlet containers will block the incoming requests until the other pending requests are finished (but they won't finish since they are waiting for the sub-requests).

Distributed Indexing

It's up to the user to distribute documents across shards. The easiest method to determine what server a document should be indexed at is to use something like **uniqueId.hashCode() % numServers**.

SolrCloud does implement distributed indexing and has various strategies available.

Distributed Search Example

For simple functionality testing, it's easiest to just set up two local Solr servers on different ports.

```
#make a copy
cd solr
cp -r example example7574

#change the port number
perl -pi -e s/8983/7574/g example7574/etc/jetty.xml  example7574/exampledocs/post.sh

#in window 1, start up the server on port 8983
cd example
java -server -jar start.jar

#in window 2, start up the server on port 7574
cd example7574
java -server -jar start.jar

#in window 3, index some example documents to each server
cd example/exampledocs
./post.sh [a-m]*.xml
cd ../../example7574/exampledocs
./post.sh [n-z]*.xml

#now do a distributed search across both servers with your browser or curl
curl 'http://localhost:8983/solr/select?shards=localhost:8983/solr,localhost:7574/solr&indent=true&q=ipod+solr'
```

Again, see SolrCloud page for an example of a more current approach to distributed search.