# Solr Tuning

Solr has two main disciplines which can be tuned for performance and has different characteristics and considerations while addressing performance:

1. Indexing data
2. Searching for data

## Some of the recommendations for Solr Indexing Tuning:

**Speed**1: a fast disk subsystem is essential if search and indexing performance is to be maintained as the number of documents in the repository grows.

1. Index files (e.g. /alf\_data/solr4/index ) should be on the fastest disk subsystem. The index volumes can be under heavy read/write load; a slower disk write times will adversely affect indexing times. So, Striped SSDs are recommended.
2. The index content files can be stored on less expensive, slower disks. When indexing a document, Solr4 asks the repository for a copy of the text extracted from the document. Transformations are particularly expensive operations for the repository, so to reduce the load, Solr4 caches a copy of the extracted text, compressed, in a separate directory (solr4/content).

**Memory:**

**ramBufferSizeMB2:**

Once accumulated document updates exceed this much memory space (defined in megabytes), then the pending updates are flushed. This can also create new segments or trigger a merge. Using this setting is generally preferable to maxBufferedDocs. This is defaulted to 100MB. Based on the available memory, increase this to higher number such as 200 or 300 (MB) in solrconfig.xml

**maxBufferedDocs:**

Sets the number of document updates to buffer in memory before they are flushed as a new segment. This may also trigger a merge. The default Solr configuration sets to flush by RAM usage. This is defaulted to 1000.

**mergeFactor3:**

The mergeFactor roughly determines the number of segments. The mergeFactor value tells Lucene how many segments of equal size to build before merging them into a single segment.

High value merge factor (e.g., 25):

* Pro: Generally improves indexing speed
* Con: Less frequent merges, resulting in a collection with more index files which may slow searching

Low value merge factor (e.g., 2):

* Pro: Smaller number of index files, which speeds up searching.
* Con: More segment merges slow down indexing.

For indexing tune the merge factor to 25 in solrconfig.xml.

**Garbage collector4:**

Solr operations are memory intensive so tuning the Garbage collector is an important step to achieve good performance. Monitor closely the JVM health of both Solr and Alfresco (GC, Heap usage).

For Solr4 (Alfresco 5.x) add next options to its JVM startup options5:

-XX:+UseConcMarkSweepGC -XX:+UseParNewGC

Other options recommended by Alfresco, Disable explicit GC and include

-XX:+UseG1GC

**Disk Cache:**

For index updates, Solr relies on fast bulk reads and writes. One way to satisfy these requirements is to ensure that a large disk cache is available. Ideally have enough memory available in the OS disk cache so that the important parts of your index, or ideally your entire index, will fit into the cache.

For example: Let’s assume a Solr index size of 8GB. If OS operations, Solr’s Java heap, and all other running programs require 4GB of memory, then an ideal memory size for that server is at least 12GB.

**Core Pool Size:**

Search and Indexing can be processor–intensive operations. The core pool size dictates how many threads are used for performing the indexing of nodes.

Set the Core Pool Size to half the number of CPUs available

In the solrcore.properties file in solr4/workspace-SpacesStore/conf set the value of the property alfresco.corePoolSize to half the number of CPUs available to the system that Solr is running on.

**Analyze your Indexing process:**

During the indexing process, plug in a profiler tool (YourKit) to check the repository health during the indexing. Sometimes, during indexing, the repository layer executes heavy and IO/CPU/Memory intensive operations like transformation of content to text in order to send it to Solr for indexing. This can become a bottleneck when for example the transformations are not working properly or the GC cycles are taking a lot of time.

**Batch count:**

This property indicates the number of updates that should be made to this core before a commit is executed.

Increase your index batch counts to get more results on your indexing webscript on the repository side. In each core solrcore.properties, raise the batch count to 2000 or more alfresco.batch.count=2000

**Disable Archiving store indexing:**

Disable full text Indexing on archive:SpacesStore. In Solr, this is done by adding the property alfresco.index.transformContent=false. Alfresco never searches for content inside files that are deleted/archived. This saves on disk space, memory on Solr, Cpu during Indexing and overall resources.

**Enable JodConverter instead of Libreoffice (only Enterprise)**

Make sure that only one transformation subsystem is enabled. Update the following in alfresco-global.properties

ooo.enabled=false

jodconverter.enabled=true

**Disable SSL:**

By default, the communication between Solr and Alfresco is encrypted, if you don’t need this encryption it’s a good idea to disable this in order to reduce complexity that can contribute to increased performance.

On the Alfresco server, edit the alfresco-global.properties and set:

* solr.secureComms=none
* On the alfresco webapp deployment descriptor web.xml, comment out the security constraint.
* For every Solr core that you have configured set alfresco.secureComms=none on the solcore.properties file.
* On the alfresco Solr deployment descriptor web.xml or solr.xml under Catalina/conf/localhost/solr.xml, comment out the security constraint.

## Some of the recommendations for Solr Search Tuning:

**Speed**1: a fast disk subsystem is essential if search and indexing performance is to be maintained as the number of documents in the repository grows. Striped SSDs are recommended.

**CPU:** Get the fastest CPU as search is CPU intensive rather than RAM intensive.

**mergeFactor3:**

The mergeFactor roughly determines the number of segments. The mergeFactor value tells Lucene how many segments of equal size to build before merging them into a single segment.

High value merge factor (e.g., 25):

* Pro: Generally improves indexing speed
* Con: Less frequent merges, resulting in a collection with more index files which may slow searching

Low value merge factor (e.g., 2):

* Pro: Smaller number of index files, which speeds up searching.
* Con: More segment merges slow down indexing.

For searching, tune the merge factor to 2 in solrconfig.xml.

**Solr Caches:**

Solr uses caches to improve query performance. We can monitor Solr caches effectiveness by monitoring cache values shown in the Solr reports:

https://<host>:8443/solr4/#/alfresco/plugins/cache

https:// <host>:8443/solr4/#/archive/plugins/cache

Each set of cache statistics has a number of different metrics. To determine the effectiveness of a cache, the most interesting figures are:

The **cumulative hit ratio** (cumulative\_hitratio) - The percentage of queries that were satisfied by the cache (a number between 0 and 1, where 1 is ideal).

The **cumulative number of inserts** (cumulative\_inserts) - The number of entries added to the cache over its lifetime.

The **cumulative number of evictions** (cumulative\_evictions) - The number of entries removed from the cache over its lifetime.

The ultimate measure of a cache's performance is its hit ratio.

If you see a high number of evictions relative to inserts, try increasing the size of that cache and monitor the effect on its hit ratio. It might be that entries are being evicted too quickly for your levels of search activity.

If a cache has a high hit ratio but very few evictions, it might be too large. Try reducing the cache size and see if there's any corresponding change in the hit ratio.

Don't be discouraged if your hit ratio remains low for certain caches. If your queries are generally non-repetitive then no amount of cache sizing is going to get that number up, and you might as well opt for a small cache size.

Solr caches are tuned and configured in solrcore.properties. Increase query caches and tune it to optimal number.

**Query optimization:**

* Avoid path search queries, those are known to be slow.
* Avoid using sort, you can sort your results on the client side using js or any client side framework of your choice.
* Avoid \* search, avoid ALL search

**Separate search and indexing tiers:**

If there are two separate solr server farms, then dedicate one to the indexing and the other to search. This will increase your global performance.

**Optimize ACL policy:**

Re-use permissions and use inherit option which is enabled by default. Don’t setup specific permissions for users or groups at a folder level.

## Monitoring Solr

**Logging search requests:**

If you want to have a look at the queries that Alfresco is running against Solr when you execute searches, or just click around, in Alfresco Share then enable debug logging as follows in

log4j.properties (located in <alfrescoinstalldir>/tomcat/webapps/alfresco/WEB‐INF/classes):

log4j.logger.org.alfresco.repo.search.impl.solr.SolrQueryHTTPClient=debug

**Logging indexing requests:**

To see what Solr is doing during indexing enable the following logging in log4j‐solr.properties (located in <alfrescoinstalldir>/solr4):

log4j.logger.org.alfresco.solr.tracker.AclTracker=debug

log4j.logger.org.alfresco.solr.tracker.ContentTracker=debug

log4j.logger.org.alfresco.solr.tracker.MetadataTracker=debug

log4j.logger.org.alfresco.solr.tracker.ModelTracker=debug

**References:**

1. <https://myalfresco.force.com/support/articles/en_US/White_Paper/Solr-Tuning> (Solr4 Basic Best Practices.pdf)
2. <https://lucene.apache.org/solr/guide/6_6/indexconfig-in-solrconfig.html#IndexConfiginSolrConfig-ramBufferSizeMB>
3. <https://wiki.apache.org/solr/SolrPerformanceFactors#mergeFactor>
4. <https://javapapers.com/java/types-of-java-garbage-collectors/>
5. <https://issues.alfresco.com/jira/browse/MNT-12938>