

I frequently get asked how many documents Alfresco Search Services can index and why is it 50 million. To shed some light on the subject I will begin with 50 million documents per instance.

The origin of 50 million documents came about while Alfresco was performing the 1 Billion Document Benchmark. This exercise used a range of files with different size documents. During the benchmark the limit was evaluated at 50 million as the optimal size, based on the data and AWS instances we were running at the time.

There are many factors to consider when sizing an index. This ranges from hardware to content type. At Alfresco we have many customers with different requirements, and this makes it hard to answer the question. The maximum number of documents on a single instance is subjective to the content indexed.

An example would be a publication company, who has a requirement to index articles, journals and books versus an insurance company who are indexing claims. The content indexed by publication tends to have a larger corpus than an insurance claim. In fact, some financial customers are only interested in indexing the metadata. As a result, the type of data indexed directly influences the size of the index

It's also worth noting that the number of custom models affects the size of the index. This is due to the fields of each model that gets indexed. As you can see these are some of the factors that need to be taken into consideration. Not knowing these details makes it hard to estimate the size of the index.

The 50M figure has always been used as a cautious guideline, like you see in an elevator where the recommended max load of 8 persons or 600 KG. We know that if one of the factors change, for instance if we saw 5 sumo fighters enter the elevator, we may consider taking the next one.



MAXIMUM LOAD
8 PERSONS
OR 600 KG

So how does one size the index and what should be taken into consideration?

We will first look at memory as it is the fastest way to retrieve the information. Life would have been so much easier if we are able to store everything in memory but that is not possible.

There are many posts and blogs on what the sweet spot is and this ranges from 6-32GB. The max figure is based on the JVM and it boils down to the size of GC. Anything greater than 32GB and you will start to notice a performance degradation, at this stage the garbage collection will start having issues clearing the objects.

One final note on memory, is to leave enough for the operating system. A quarter of your physical memory should be enough for the heap space running the index, the rest should be left free for the operating system cache. If there are more applications running on the same instance, then adjust accordingly. Ideally, we want the operating system to memory map as much of the index.

The disk size is slightly more elaborate, and it is dependent on how Lucene works. We won't go in depth, but we will cover some of the basic concepts that explains how the disk is used. Lucene in Action provides a formula that can be used to estimate the index size.

$$\text{"1/3 x indexed + 1 x stored + 2 x term vectors"}$$

Taking this formula, we can apply it to Alfresco Search Services and estimate what is the projected index size. [There is a useful excel spreadsheet](#) by Lucid works based on the above formula, by entering the number of fields and average size of documents in Alfresco we can estimate the size of the index. I would add that this excel is a helpful tool for estimating but has its limitations and issues.


Out of the box Alfresco has 893 documents with 205 fields, 10 stored fields and an average document size of 5KB. When we enter the values into the spreadsheet, we can expect to see the index size of 4.1MB which is near the actual index size.

Keep in mind that the number of documents, document type, size and custom models will affect the results. Each field indexed may contain different size corpus which impacts the index size. For example the fingerprint along with cm:content fields are known to take up considerable space on the index.

In another instance we have indexed some of the content from the Project Gutenberg along with a few metadata files. When we updated the average document size with 2MB and the total documents to 16000, we were able to project the index size.

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q
1	Lucene/Solr Disk Usage Estimator																
2																	
3	Disk Storage Estimate (MB):	4793.711177	Disk Space Needed for Optimize (common case):		9587.42235	Optimize Worst Case*:		14381.1335									
4																	
5																	
6	Assumptions																
7	Total # of Documents	16,029															
8	Avg. Document Size (KB)	2000															
9	Indexed Field Compression Factor	0.33333															
10	# of Indexed Fields	205															
11	# of Stored Fields	10															
12	# of Fields with Term Vectors	0															
13	Transient (MB):	4															
14																	
15																	
16																	
17																	
18																	
19																	
20																	
21																	

Excel index estimation based on 16,000 documents



Dashboard

Logging

Core Admin

Java Properties

Thread Dump

alfresco

Overview

Analysis

Dataimport

Documents

Files

Ping

Plugins / Stats

Query

Replication

Schema

Segments info

Statistics

Last Modified: about 21 hours ago

Num Docs: 16097

Max Doc: 16097

Heap Memory Usage: ~1

Deleted Docs: 0

Version: 5723

Segment Count: 1

Optimized: ✓

Current: ✓

Replication (Master)

	Version	Gen	Size
Master (Searching)	1572959700935	279	4 GB
Master (Replicable)	1572959700935	279	-

Instance

CWD: /opt/alfresco-search-services/solr/server

Instance: /opt/alfresco-search-services/solrhome/alfresco

Data: /opt/alfresco-search-services/data/alfresco

Index: /opt/alfresco-search-services/data/alfresco/index

Impl: org.apache.solr.core.NRTCachingDirectoryFactory

Healthcheck

Ping request handler is not configured with a healthcheck file.

Documentation

Issue Tracker

IRC Channel

Community forum

Solr Query Syntax

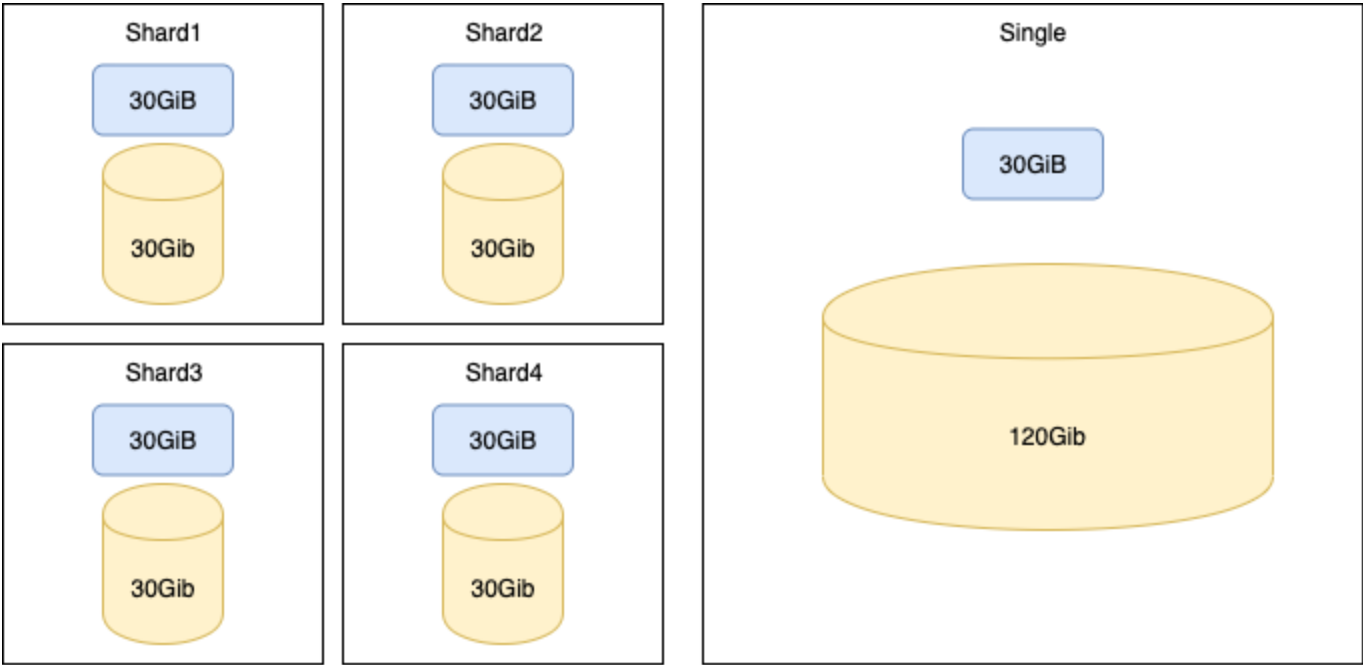
Solr admin console

The total number of nodes can be obtained by querying Alfresco DB, with the following SQL:

```
select * from (select count( * ) N_Alfresco from alf_node where store_id = (select id from alf_store where protocol = 'workspace' and identifier = 'SpacesStore')) as N1 ,  
  
(select count( * ) N_Archive from alf_node where store_id = (select id from alf_store where protocol = 'archive' and identifier = 'SpacesStore')) as N2 ;
```

Taking an example of 100M documents which required 120GB memory. We would recommend breaking this down into smaller pieces. Putting the entire data into memory is not advisable and for an optimal performance we would advise splitting it into 4 shards each holding up to 30GB.

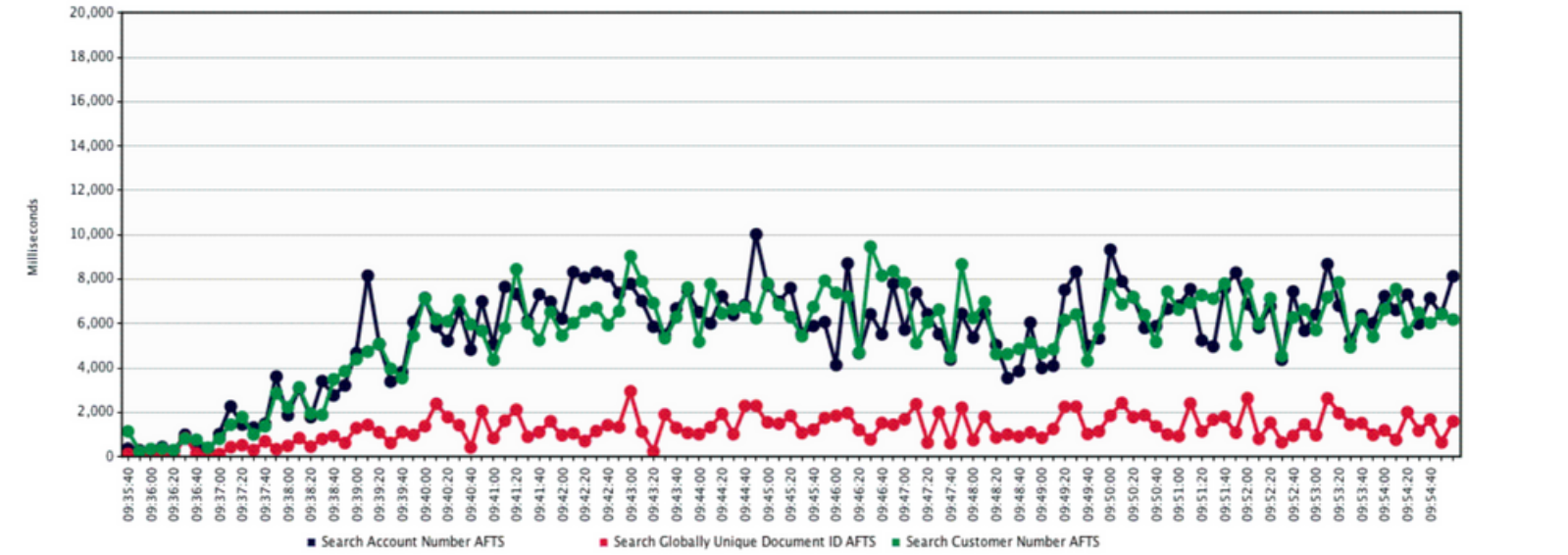
There is another option of storing data on disk. This has an impact on the query performance as it's no longer served from memory. Accessing the disk takes longer but will still perform well, up to a point. If we take the same example, we would store the 120GB on disk and make use of the 30GB memory, the performance should still be more than adequate and sharding should not be required.



m4.4xlarge Memory 64Gib \$0.80 per Hour
\$0.125 per GB-month of provisioned storage

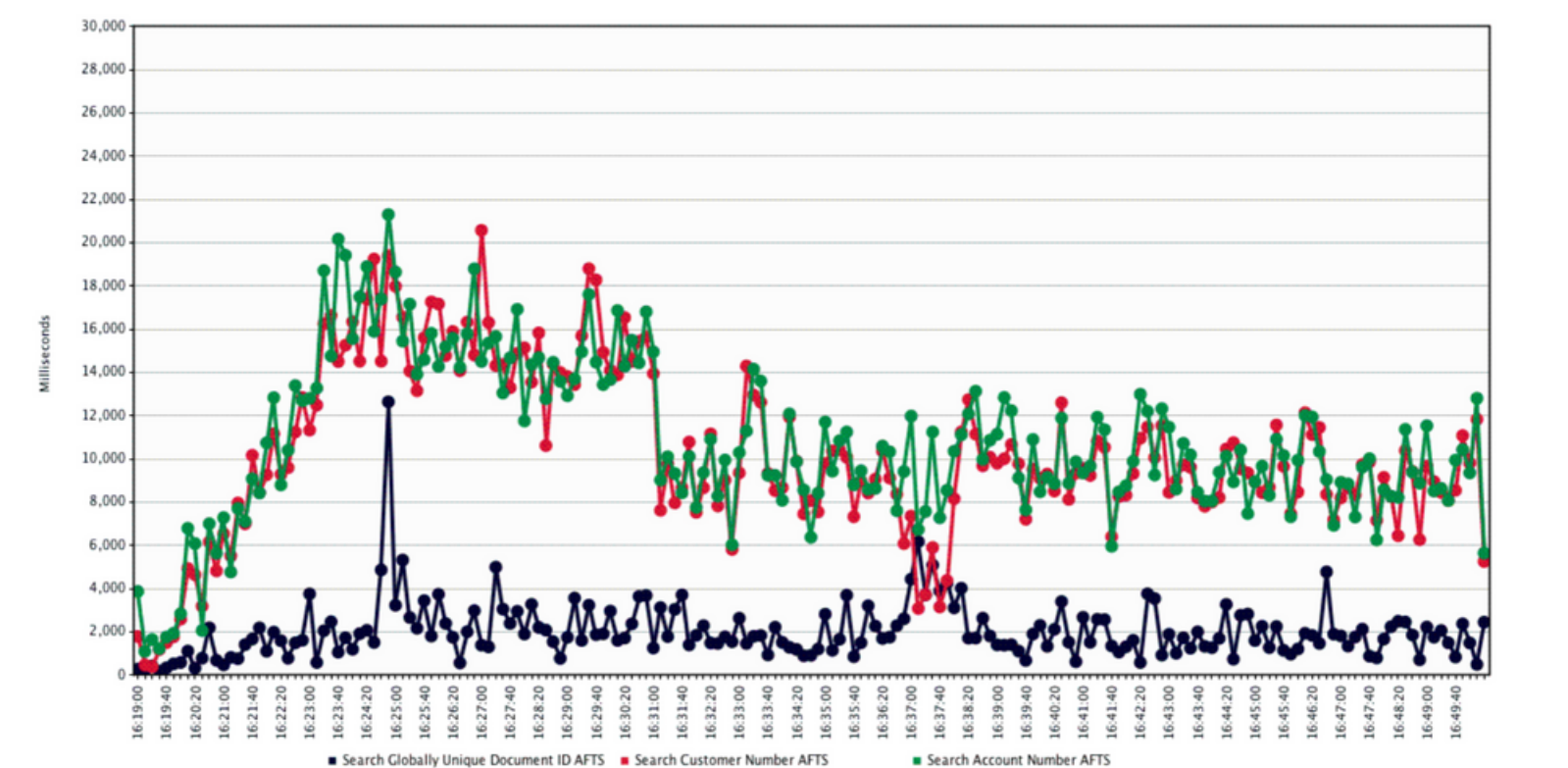
Sharding Solr

We looked at indexing 330M documents using 2 different strategies. The first set was storing 50M per shard while the second set stored 100M per shard. When the query was executed against both sets it was clear that first set using 50M per shard was far more performant.



Response time for a 50M document instance

That said the second set holding 100M per shard performed well and matched the global unique id search but under performed on the other queries compared to the first set.



Response time for 100M document instance

Note that the more we store on disk the slower the response will become. To some users this might be within the acceptable response time, but we encourage you to perform your own analysis on your data. When analysing the performance, make sure to use different queries, as some are more taxing than others and this will give you a better picture. If the response time is below the expectation and getting worse, we would recommend looking into sharding.

As for the maximum number of documents one could store per instance is limited by the disk space. It is worth noting the more data you ingest the harder it is to manage the index, and this is down to how Lucene works. Lucene writes the information into segments; each segment represents a collection of data. When the segment number and shape meet the conditions set in solrconfig.xml, they are merged into one bigger segment. In order to merge the segments, there needs to be enough disk space to complete the task.

Lack of disk space when merging the segments will soon become apparent as you will not be able to write to disk. We have seen cases where the segment count was 176 and the index size was 750GB. This translates to 176 segment each holding a rough average of 4200MB. To merge these segments there needs to be enough disk space for the current segments along with the new combined segment. As a result the spike in disk usage required at least 1.5TB of free space or the operation would not be able to complete the merge. In this case the commit hung, and the operation never completed.

In many cases the 50M documents per instance is about right but we always recommend performing additional analysis to ensure it meets your requirements. We recommend measuring the response time of queries and commits. Evaluate the average document size and the number of fields used. Hopefully this helps explain the mechanism behind the process and clarify how one can estimate the number of documents to index into a single Alfresco Search Services.