Alfresco One 5.1

Sizing Guide

Contents

Document History

Version	Author	Date
0.1	Luis Cabaceira	March 11, 2016
0.2	Luis Cabaceira	April 28, 2016
0.9	Luis Cabaceira	June 17, 2016
1.0	Luis Cabaceira	August 09, 2016

Introduction

Sizing a large ECM deployment is a complex topic. Understanding the right variables that influence system sizing requires an intimate understanding of the expected use case. Understanding the factors that influence resource usage is key to providing more accurate sizing estimates. The number of concurrent users, the data volume, and desired response times all influence sizing decisions and calculations.

The most important sizing factor will always be the use case, depending on which sizing figures must be adjusted.

Executive Summary

Alfresco has proven itself as a highly scalable and elastic ECM solution. Document repositories in excess of one billion documents are possible with performance that remains fast and predictable. This document aims to guide the readers on how to size their Alfresco infrastructure to achieve their business goals with the expected response times. The scope of this document is constrained to the Alfresco stack, including the Alfresco repository, Share and Solr applications. The baseline for this Sizing Guide is the Alfresco One on AWS: Reference Architecture (8) and the benchmark tests executed against a repository with 50 and 100 million nodes.

Intended Audience

This document is intended for Alfresco developers, administrators, and architects to help them understand the factors that influence the size of their Alfresco projects.

Disclaimer

Alfresco sizing, like other systems, has many subtleties that are hard to fully systematize. Each deployment will have specificities that will demand consideration while estimating sizing for the different layers (repository, indexing, transformation, storage, database, search). In any case this document assumes that there are certain fairly generic steps and concerns that are mostly common between the different sizing efforts. The purpose of this document is to provide a guide around those generic procedures and requirements for providing Alfresco sizing estimates.

All technical scaling strategies, performance metrics and Alfresco's internal benchmark scope and test results are deeply explained in the Alfresco Scalability Blueprint (9) document. More information about the environment used is in the section Lab Definition.

Baseline benchmark for this document

Recently Alfresco has executed several benchmark tests, testing its repository with up to 1 billion nodes.

The assumptions and suggestions in this Sizing Guide are based on the benchmarks tests executed against repositories with 50 million and 100 million documents.

With this approach we intend to provide sizing figures for a more common customer deployment scenario. Further benchmarks are planed and new versions of this document may include sizing scenarios for bigger repositories.

Specific exclusions

This document excludes any tuning advice specific to a particular operating system, virtualization platform, storage device, or network hardware.

Also excluded from this document are any performance metrics around specific transformers. General guidance for scaling transformations is available, but in order to accurately measure index-tracking performance in isolation, transformations were taken out of scope.

The Alfresco test environment uses automatically generated content, comparable to an in-place loading. This will affect the download benchmark accuracy. However, we see this as an acceptable tradeoff since download performance is largely a function of bandwidth between the client and server and will vary widely in actual customer environments.

Labs definition

Alfresco has built two test lab environment on AWS. AWS offers several advantages for this type of testing, especially the ability to rapidly scale the test lab environment up and down as needed to simulate a wide variety of different scenarios.

Despite being AWS a singular environment all metrics and specifications can be also be used to size other environments with certain degree of accuracy. Consider the definitions of the AWS Reference Architecture as the starting point for your sizing.

Alfresco's scalability team conducted testing in three separate environments, identified here as Lab 1, Lab 2, and Lab 3.

Common configuration

The following sections describe the AWS environment on all labs where benchmark tests were executed. The variations on the different labs are explained on each lab definition.

Alfresco application server Hardware/Software specifications

Hardware component	Specifications
AWS Server Type	c3.2xlarge
Memory RAM	15GB RAM
CPU	8vCPU Intel Xeon E5-2680 v2 (Ivy Bridge), 2.80 GHz.
Storage	2x80 GB SSD (gp2, max throughput 160MiB/s, burstable to 3000 IOPS)
Software component	**Specifications**
Operating System	Red Hat Enterprise Linux 6.5
Alfresco version	Alfresco One 5.1
Database	Amazon Aurora (MySQL compatible).
Alfresco apps	Alfresco One 5.1 repository and Alfresco Share 5.1
Alfresco configuration	**Specifications**
JVM max memory	-Xmx6G
Tomcat Threads	200 (default)
Database pool connections	500
JodConverter	Disabled
Alfresco cache settings	**Specifications**
cache.node.nodesSharedCache.maxItems	500000
cache.node.aspectsSharedCache.maxItems	500000
cache.node.propertiesSharedCache.maxItems	500000
cache.authorityToChildAuthoritySharedCache.tx.maxItems	100000
cache.authoritySharedCache.tx.maxItems	100000
cache.personSharedCache.tx.maxItems	50000
cache.userToAuthoritySharedCache.tx.maxItems	50000
cache.aclEntitySharedCache.maxItems	200000
cache.aclEntitySharedCache.cluster.type	invalidating
cache.authoritySharedCache.maxItems	200000
cache.aclSharedCache.maxItems	200000
cache.userToAuthoritySharedCache.maxItems	200000
cache.authenticationSharedCache.maxItems	20000
cache.authorityToChildAuthoritySharedCache.maxItems	200000
cache.zoneToAuthoritySharedCache.maxItems	10000
cache.personSharedCache.maxItems	100000
Share configuration	**Specifications**
JVM max memory	-Xmx4G
Tomcat Threads	200

Solr4 application server hardware specifications

Hardware component	Specifications

AWS Server Type	c3.8xlarge
Memory RAM	58GB RAM
CPU	32vCPU Intel Xeon E5-2680 v2 (Ivy Bridge), 2.80 GHz.
Storage	2x320GB SSD (io1, max throughput 320MiB/s, 10K IOPS sustained) Separate EBS volume for Solr content caches (gp2, max throughput 160MiB/s, burstable to 3000 IOPS)
Solr Sharding	Yes
Dedicated Tracking	Yes
Software component	**Specifications**
Operating System	Red Hat Enterprise Linux 6.5
Solr version	Solr 4
Alfresco apps	Alfresco One 5.1 repository and Solr 4
Solr configuration	**Specifications**
JVM max memory	-Xmx30G
Tomcat Threads	100
Alfresco configuration	**Specifications**
JVM max memory	-Xmx10G
Tomcat Threads	100
Database pool connections	500

Note: The Lab 1 environment uses separate JVMs for each deployed web application.

Database server

Hardware component	Specifications
AWS Server Type	db.r3xlarge
Memory RAM	15GB RAM
CPU	2vCPU Intel Xeon E5-2680 v2 (Ivy Bridge), 2.80 GHz.
Storage	EBS Optimized, 1000Mb/s
Number of Servers	1

Lab 1 definition

This Lab has 2 nodes on the repository layer and 2 Solr nodes shard instances.

- 2 x Alfresco/Share stacks
- 2 x Solr shards (shard1, shard2) with Alfresco in cluster

Lab 2 definition

Lab2 adds 2 more nodes on the repository layer and 2 more Solr shard instances.

- 4 x Alfresco/Share stacks
- 4 x Solr shard instances (shard1, shard1, shard2, shard2) with Alfresco in cluster

Lab 3 definition

Same as Lab2 but doubles the memory and cpu capacity on the Aurora Database

- Database is Aurora db.r3.xlarge (**4 vCPU, 30GB**)

About Aurora database

The environment described in this document uses AWS Aurora as the relational database. Aurora is a highly scalable MySQL compatible relational database. While Aurora is not an option for on-premises Alfresco deployments, it provides a stable, scalable and high performance platform for Alfresco benchmark testing.

Alfresco One on AWS: Reference Architecture

Alfresco has defined and documented a reference deployment architecture for Alfresco One on AWS. This architecture has been developed over the course of several years with input from engineering, support, partners, and customers. It is possible to deploy the AWS Reference Architecture across a variety of both cloud and on-premises environments such as AWS, virtualized platforms, or physical hardware. The AWS: Reference Architecture seeks to separate the different components of the platform so that they may be scaled and tuned independently without running into contradictory requirements.

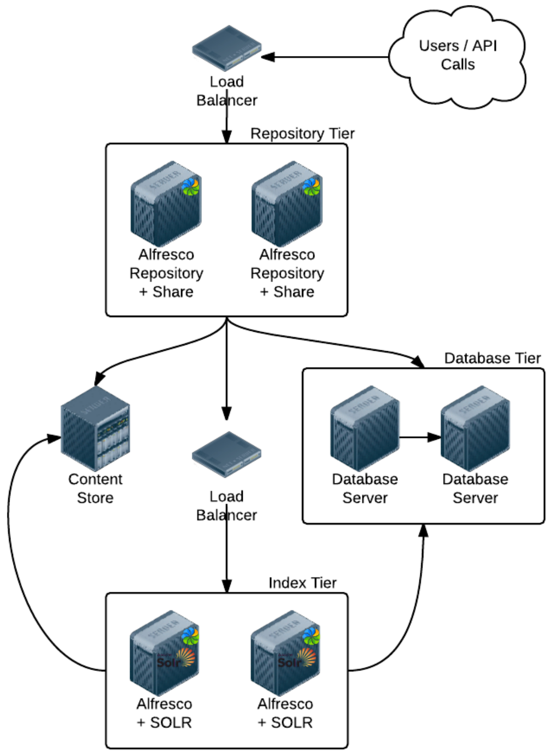

Figure 1

This architecture is explained in detail in the **Alfresco One on AWS: Reference Architecture** (8) whitepaper. Refer to this document for all technical details around this architecture.

Benchmarks Methodology overview

The benchmarks executed against the Alfresco repository, with 50 and 100 million documents, had the following important characteristics:

- Only text files were created (100K)
- Share tests simulated a maximum of 150 concurrent users per node
- Site-based access control (no permissions associated to nodes)
- Users are all Alfresco NTLM
- No wildcard searches or complex search queries
- OOTB content model (OOTB metadata attributes)
- Max 1000 files per single folder (some smaller sites had less than 1000)
- Folder hierarchy depth = 2
- CMIS tests (Read/Write/Search Split = **27%, 28%, 45%)**
- Share tests (Read/Write/Search Split = **70%, 5 % ,25 %)**
- Think time = 30 seconds

The sizing benchmark tests were conducted against a snapshot build of Alfresco One 5.1. While we do not anticipate any large variance between the snapshot and final builds of version 5.1 in terms of performance, it is possible that some minor differences will exist between the snapshot and final builds.

Each test was executed using the Alfresco Benchmark Framework (1). This open source framework allows for distributed, highly scalable testing. More information on the Alfresco Benchmark Framework can be found on the Github project page.

Current benchmarks

Alfresco runs a number of benchmarks intended to test various parts of the Alfresco stack. These benchmarks are intended to model different types of real world performance scenarios.

Standard Share benchmark

Alfresco's standard Share benchmark tests various common actions performed in Alfresco Share, such as logging in, navigating to a site dashboard, viewing a document library, uploading a file or searching for people. The tests use Alfresco's Webdrone (5, 6) in conjunction with Selenium to ensure that results mimic real world browser scenarios as closely as possible.

Share benchmarks results and analysis

Alfresco's Lab1 and Lab2 environments (as described in the section Lab Definition) were used to perform testing along the concurrent user/session dimension. Repositories of 50M and 100M documents were created, and testing was executed at several levels of user and API session concurrency. Both the Alfresco Share and CMIS tests were run in this environment.

Share benchmark results analysis

Alfresco Share tests were performed on BM6 with user concurrency levels between 24 and 96 users with a very aggressive think time (10 seconds). This will correspond to approximately **250 concurrent** users with a standard think time of 30 seconds. Above 96 users, we began to experience some events failing and timeouts occurring in the test (Figure 1). Analysis shows this was largely related to search performance.

Search performance is an integral part of the Share experience, even when it is not performed explicitly as a document search. For example, loading the document library, viewing the content of a folder, listing sites, and listing users are all examples of non-explicit searches that still depend on search subsystem performance for a good user experience.

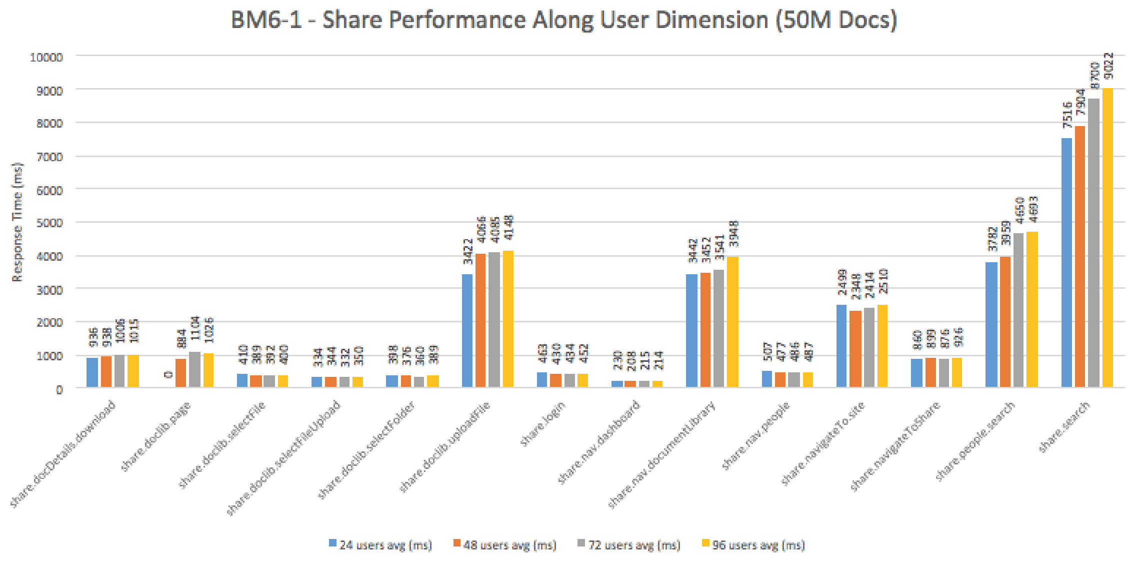

Figure 2

As the events (especially those with a search component) began to take longer to complete, the overall operations per second began to decline (Figure 2). This is indicative of a bottleneck in the repository and search tiers.

This limit/bottleneck was addressed by scaling up the cluster size. We've executed the same test in Lab 2 and we've reached user concurrency levels up to **480 users** within the acceptable response times, as shown on the next figure.

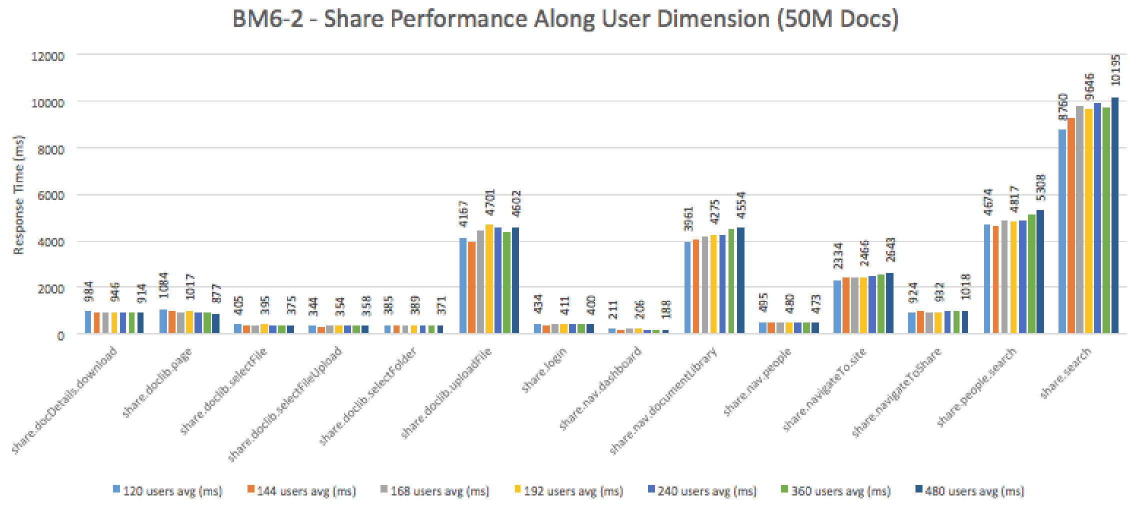

Figure 3

Share shows some minor performance degradation as the number of concurrent users increases. This degradation is gradual, predictable, and largely confined to operations that depend on search. Most basic navigation functionality remained stable throughout testing. Operations that depend heavily on the database or search, as expected, began to take longer as more users were added to the test.

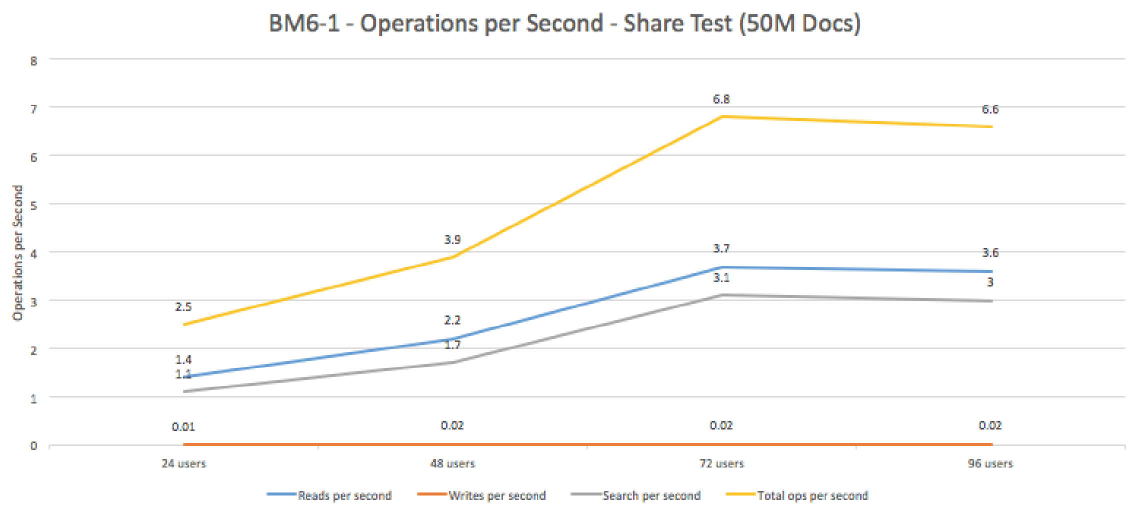

Figure 4

Improving performance under increasing levels of user concurrency could be achieved in this case by additional repository or index nodes. Let's take a look at how the diagram for operations per second looks when we execute the same test on Lab2.

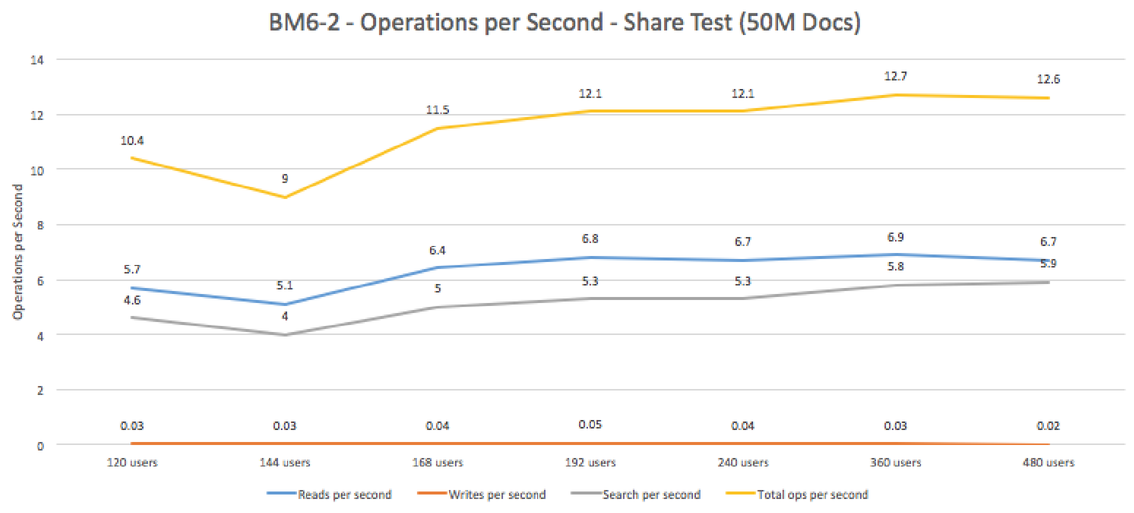

Figure 5

Total operations per second trended upward, as would be expected (Figure 8). However, because page load times also increased somewhat, the operations per second did not increase at the same pace as the number of users. It is worth noting that between Lab1 and Lab2 testing, the peak number of Share operations per second doubled. This confirms that scaling the repository horizontally does not improve single operation performance, but it does enable a larger number of concurrent operations, and it is the appropriate strategy for supporting larger numbers of Share users.

Share benchmark sizing conclusions

Considering the AWS: Reference Architecture, and assuming a user think time of 30 seconds, we can conclude that the limit, in terms of concurrent users will be **250**, from that point, the architecture needs to be scaled.

CMIS benchmark

CMIS is a widely used interface between the Alfresco repository and external systems. Alfresco includes a CMIS benchmark in the test suite. This benchmark tests the performance of common CMIS operations such as finding, creating, and deleting folders, uploading and retrieving content, and searching.

CMIS benchmark results analysis

CMIS testing was also conducted in the Lab1 environment. These tests revealed patterns similar to the Share tests. The CMIS tests create a specific number of sessions per second, and measure the performance of operations performed within that session. As the Lab1 environment scaled up to creating 8 sessions per second, the average response time climbed significantly (Figure 3).

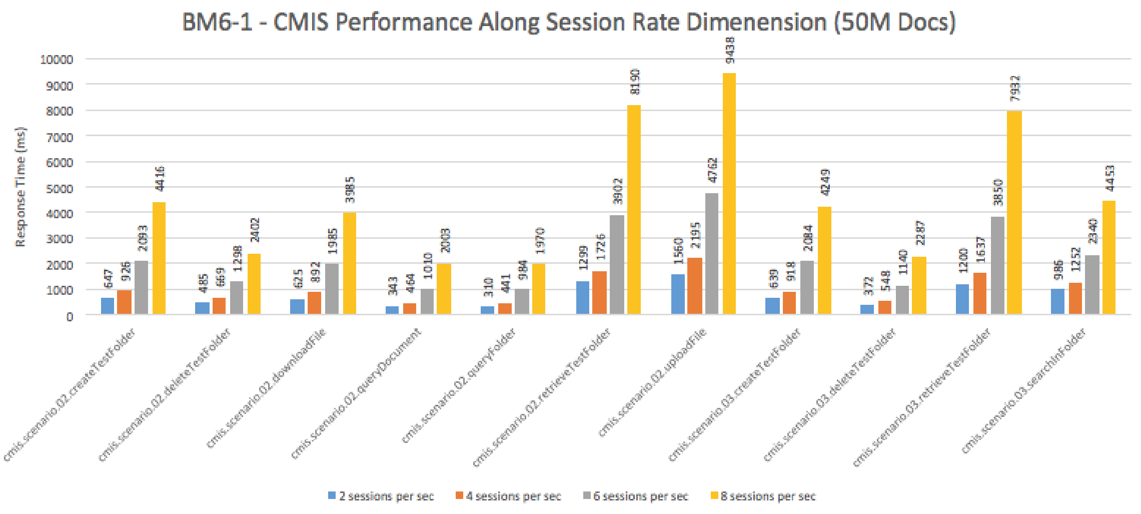

Figure 6

CMIS testing in the Lab 2 environment revealed patterns similar to the Share tests. Increasing the number of sessions created per second (and thus, the level of concurrency) has a measurable, predictable impact on response times per request.

Monitoring during the Lab 2 CMIS testing showed the database was at its capacity during the 8 sessions per second test. This exposes the database as the bottleneck for CMIS tests.

A peak performance of 56 operations per second was achieved in Lab 1.

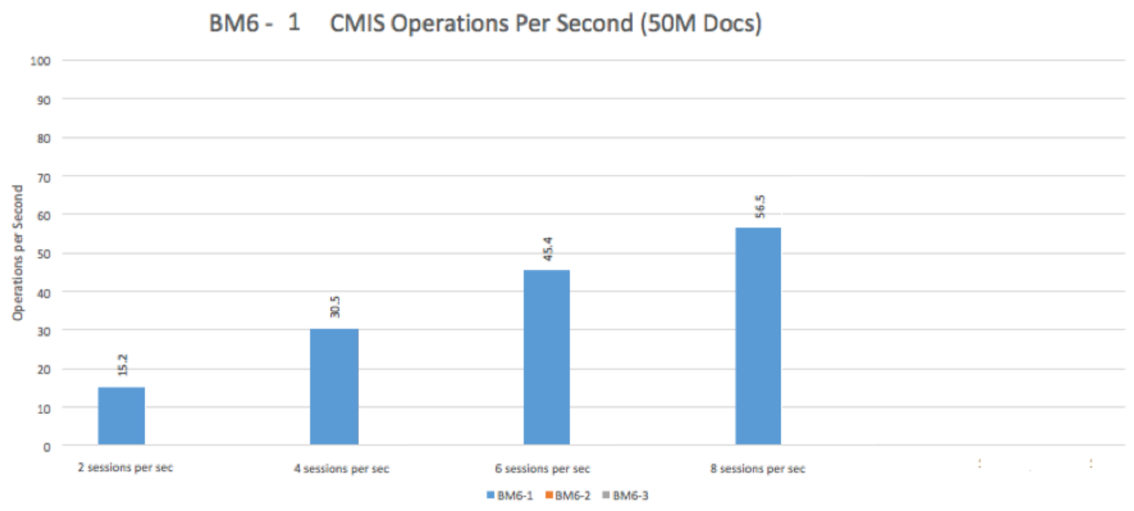

Figure 7

CMIS benchmark sizing conclusions

CMIS tests were scaled as CMIS sessions per second, where each session went through a sequence, as defined by three different scenarios, as quickly as possible. The peak performance of the AWS Reference Architecture was 56 operations per second simulating 8 sessions per second. These tests were re-run in the Lab2 environment to investigate the effects of scaling out the repository and index tiers on CMIS performance.

Additional cluster nodes improved performance at the 8 session per second level slightly, but reaching acceptable performance beyond that level required an increase to the database capacity. Test results show that using CMIS, while maintaining acceptable per-request performance and operations per second, is largely dependent on the database.

Full repository size benchmarks

The most recent Alfresco Benchmarks tests measured the performance of some key Alfresco Share features as the test lab was loaded.

Measurements were taken at 33M, 77M, 103M, 140M, 220M, 306M, 442M, 698M, and 1068M documents using 45 concurrent users.

These tests were executed on a special Lab with 10 Alfresco nodes and 20 Solr nodes/shards. The details of this special Lab are in Appendix B.

As expected, the database grew along with the repository in a predictable fashion, as did the number of Alfresco transactions. This is shown in the graph below.

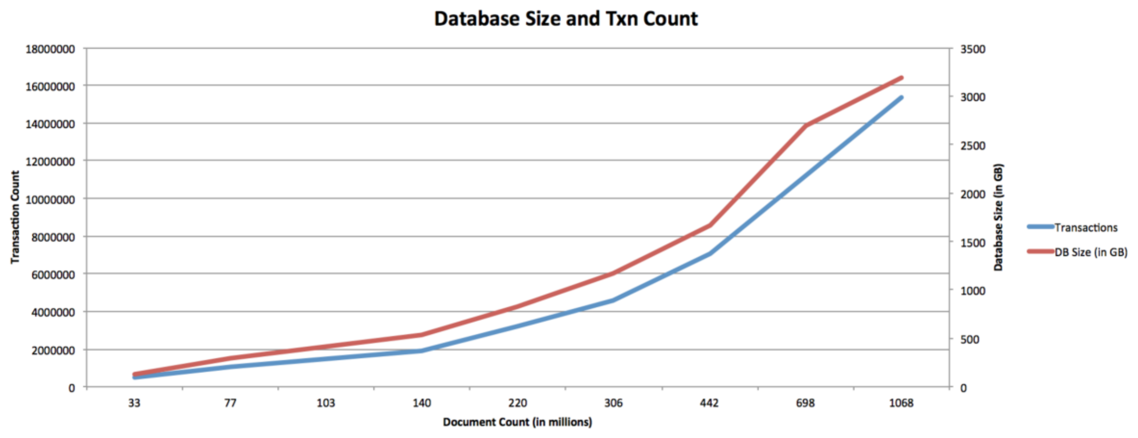

Figure 8

Alfresco Share performance was predictable and reasonably flat as the repository grew in size.

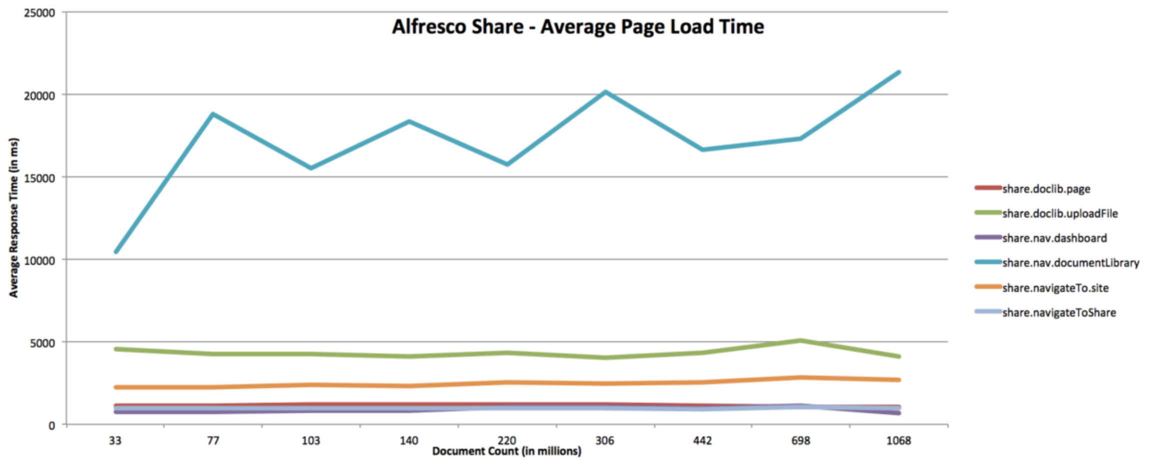

Figure 9

Once the repository was fully loaded to 1068M documents, additional testing was performed using the Alfresco CMIS benchmark. Testing at the 1068M document level reveals that the performance of this API is still acceptable. The CMIS test was conducted with 60 concurrent sessions.

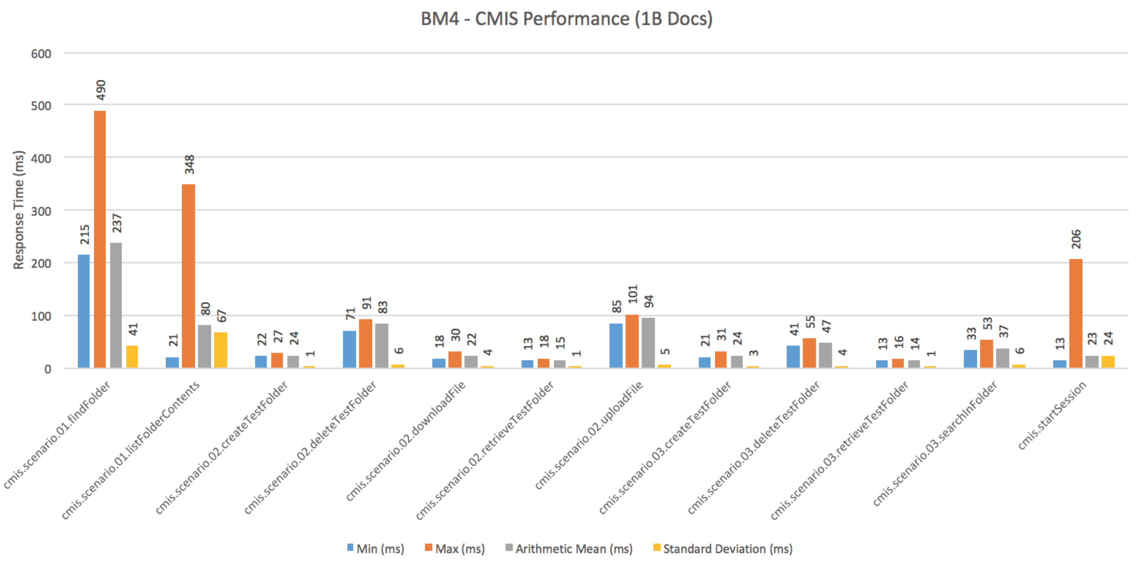

Figure 10

The Alfresco REST API tests were also executed against the fully loaded repository in the test environment. Much like the CMIS benchmark, response times for the Alfresco REST API were found to be well within the acceptable range. The REST API benchmark was also conducted with 60 concurrent users.

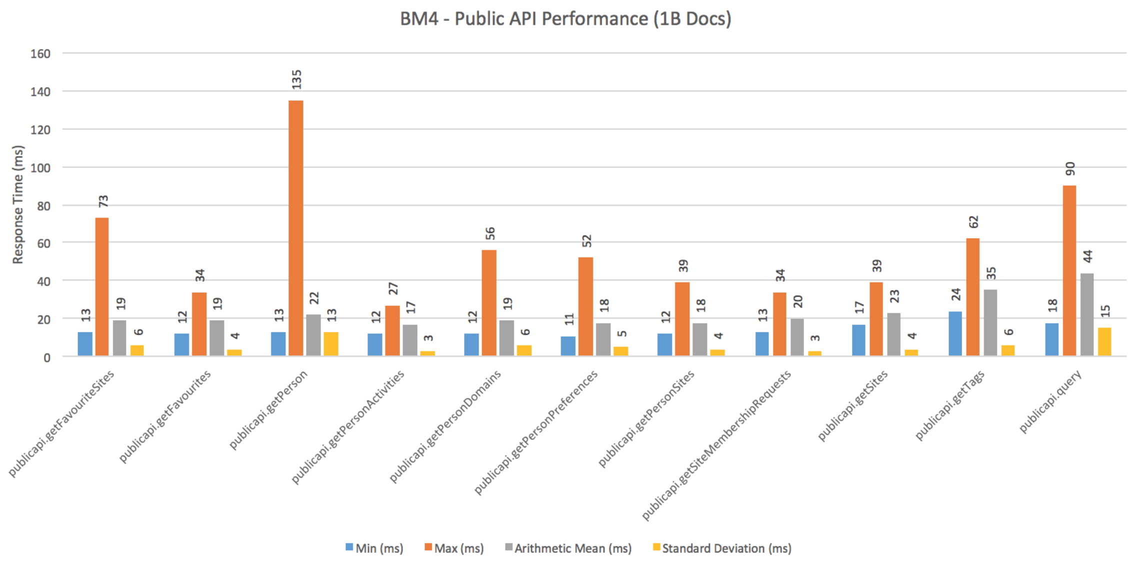

Figure 11

Testing at the billion document level with Alfresco Share, the CMIS test and the public REST API test on AWS Aurora reveals that with a high performance database underpinning the repository, Alfresco is capable of supporting very large amounts of content with little performance degradation as the repository grows along the content dimension.

Sizing Guide – factors that influence sizing

There are several key factors we must consider as ground base for our sizing decisions. These factors are main influencers for sizing calculations. The table below shows the factors and key information that should be collected at the sizing stage of your Alfresco project. That information will be used to adjust the sizing formulas provided in this guide

Sizing factor	Information
Use Case	**Gather all information related to how the solution is intended to be used by final end users and integrated systems.**
Concurrent Users	**Number of concurrent users (including system users). Include detail around average think-time for the concurrent user number defined and average and peak values.**
Repository	**Document types, average sizes, and distribution ratios.**
Response Times	**The detail around response time requirements. May include requirements over the average operations response time, but also limits for each single operation and per operation type. See Response Times Matrix**
Architecture	Gather overview of architectural requirements and Alfresco components role in overall architecture. Solution architecture options for optimized sizing. Is high availability needed? Failover? Does it use virtualization or not? Stack components being used.
Authority Structure	Number of users, number of user groups, group tree/structure depth, average number of users per group and average number of groups per user.
Operations	Percentage of read and writes operations. Split ratio between browse/download/search operations. Which types of search operations will be executed (global, full text, custom search).
Components, Protocols and API	Which components, protocols, and APIs of the product will be used ?
Batch Operations	Details on number and types of batch jobs: workflows, scheduled processes, and transformations.

Sizing factor	Information
Customizations and Integrations	Details around required customizations and possible impact. Details on external system integrations. If there's already developed code, evaluation on the code performance and possible optimization of the same code.

The first 4 shaded rows above indicate the fundamental information. This Sizing Guide primarily focuses on these factors. The accuracy of the sizing exercise is affected by the level of detail at the information gathering stage.

Use case

It is critical to understand the purpose of the system to avoid providing values that do not take into account the many subtle (and not so subtle) details that may be hidden behind the rest of the information provided. Alfresco use cases vary considerably because of the elasticity of the platform, and although we can enumerate some generic common use cases, the detail of each real implementation may vary from an architecture and sizing perspective. This Sizing Guide focuses on 2 common use cases that require different sizing approaches:

- **Collaboration**: Big corporations having thousands of users loading data and using Alfresco also as collaboration suite.
- **Headless Content Platform**: Alfresco is used as a document repository, normally a group of operators/scanners or third party components are injecting documents to Alfresco using the CMIS APIs in most cases.

The table below shows the fundamental differences between these 2 use cases.

Sizing Area	Collaboration	Headless Content Platform
Search	Search is usually just a small portion of the operations percentage (around 5%)	In most of the cases especially for very large repositories there wont be full text indexing/search.
Permissions	Permission control happens at Alfresco layer. User authority structure will be complex. With users belonging to many groups in average.	Most of the times permission control is happening elsewhere. Authority structures will be in general fairly simple.
Ingestion	Ingestion rates are usually not important uploads are normally manually driven.	Injection rates are usually very important. Dedicated layers/nodes may be needed.
Repository Size	Repository sizes are usually of small or intermediate size.	Repository sizes are usually big.

Sizing Area	Collaboration	Headless Content Platform
Customization	Level of customization will vary but in most cases will concentrate at the front end (Share).	Customizations are usually important. Lately, more and more custom solution code may live external to Alfresco by using CMIS, public APIs, and so on.
Architecture	Architecture options will be in general the standard ones provided by Alfresco (cluster, dedicated index/transformation layers, and so on).	Architecture options may vary considerably with more “exotic” solutions being used: proxies, cluster and unclustered layers, sharding of Alfresco repository, and so on.
Concurrency	Concurrent users will possibly be many, with average and peak values important to be considered.	Concurrent users will be in general few but think times will be much smaller than for collaboration.
Interfaces	You may expect mostly the Share interface to be used, but also very common may be SPP, CIFS, IMAP, WebDAV and other public interfaces (CMIS) for other interfaces (mobile).	Most of the load should concentrate around public API (CMIS) and custom developed REST API (web scripts).
Batch	Batch operations should mostly be around human interaction workflows and the standard Alfresco jobs.	Batch operations will usually have a considerable importance, including content injection processes (bulk import), custom workflows and scheduled jobs.

Concurrent Users

Another sizing key factor is the number of concurrent users. People often mean considerably different things when they talk about concurrent users. For the purpose of this document, we consider that a system with an average of N concurrent users will mean that in average for a period of time (30 to 100s think time) the system will receive the requests from N different users. Smaller or bigger think times being discussed will demand in general an extrapolation of the value to compare to reference cases.

As the number of concurrent users increases, the number of database operations, reads/writes against the content store, search operations against the Solr index, and index tracking requests also increases.

Repository Size, Document Types, NodeTree

Repository size is the most important factor to consider on Alfresco sizing. As the repository grows, so does the database, content store, and index. The rate at which these parts of Alfresco grow depends on the type of content, the index configuration, the complexity of the content model, and the number of sites, folders, and documents.

The requirements around uploading and downloading large or small documents, and indexing, transforming different types of documents will vary. This can have architecture consequences. Preference for certain types of protocols and mechanisms for large files for example: FTP, bulk ingestion; or use of dedicated caching solutions for large documents downloads. But this also has consequences at the sizing level. It will be very different having to index large documents than smaller ones, and also between different types. Applications that store a very large number of files in a single folder, or applications with a very deep folder tree will exhibit different performance characteristics.

Response Times matrix

This matrix represents the boundaries of acceptable response times.

Share User Interface Alfresco Response Times Matrix	
Operation	**Maximum Response Time**
Login to Alfresco	2 seconds
Browse Repository	4 seconds
Execute search	5 seconds
Search for User	5 seconds
Create Site	4 seconds
Navigate to Site	4 seconds
Upload Document (100k)	4 seconds
Upload New Version (100k)	4 seconds
Preview Document	4 seconds
Download Document (100k)	3 seconds
Bulk Import of documents (200k)	10 documents per second
CMIS Response times Matrix	
Operation	**Maximum Response Time**
List Folder Contents	5 seconds
Create Folder	4 seconds
Delete Folder	4 seconds
Download File	3 seconds
Query Document	4 seconds
Query Folder	4 seconds
Upload File	3 seconds
Cmis Search in Folder	4 seconds

When the boundaries are exceeded, the sizing needs to be adjusted.

The main factors that influence response times are explained in the **Alfresco Scalability Blueprint** (9), and they are referred to as "dimensions of scalability".

Alfresco repository sizing

The formulas derived in this guide are based on Alfresco benchmarks executed against a repository with 50 and 100 million documents. See Benchmarks Methodology overview for details.

The Alfresco repository tier is where many expensive operations such as metadata extraction and content transformation typically occur, making the repository tier a common target for vertical scaling. Additional or faster CPUs can help the repository tier to handle more concurrent operations, or to complete operations more quickly.

Ultimately, however, most of the operations the repository performs are dependent on the performance of the underlying content store, index, and (especially) the database. Careful analysis is important to determine where bottlenecks exist.

Adding more cluster nodes will not improve the performance of individual transactions, but it will greatly increase the number of transactions that can be serviced concurrently. This is especially true if the type of work being done by an Alfresco installation mostly resides in the repository tier. Examples include metadata extraction, many out of the box transformations, and complex document actions.

Important factors for repository sizing

The following considerations are useful for adjusting the sizing for use cases that are substantially different from the ones simulated by Alfresco benchmarks.

Authority structure/permission checks

Browsing the repository using a client (share or other client) performs ACL checks on each item present on the folder being browsed. ACL checks go against the database and occupy a thread of the application server, causing CPU consumption. As a golden rule, consider having a max of 1000 documents on each folder to reduce this overhead. Our benchmarks are using a site based access control, so no permissions are applied to nodes or parent container nodes.

Typical user operations

User operations take application server threads. When there is more concurrency, then more threads will be occupied, and more CPU will be used. Size the maximum number of application server threads (and the number of cluster nodes) according to your expected concurrency. Each thread normally holds a database connection, so concurrency impacts both Alfresco cluster nodes and the database.

Small customizations - big impact
Customizations on the repository should be carefully analyzed for memory and CPU consumption. Experience tells us that this is one of the most important factors contributing to decreased performance. Note that our benchmarks were executed against a vanilla Alfresco with no customizations.

Protocols and APIs
Some protocols will be more demanding than others and require a more scaled-up or scaled-out system to provide acceptable response times. In general, attention should be paid to some problematic protocols. For example, from a load point of view, CIFS and IMAP are protocols that may affect performance. Other protocols and interfaces, like FTP and WebDAV, should be considered reasonably light in comparison.

Scale up before scale out
Whenever you scale out your cluster, adding more nodes, you should not expect linear scalability. So, doubling the load on a cluster that is two times bigger will not necessarily deliver the same response time. In most cases, you should expect some degradation of the response time arising from cluster cache synchronization extra load. To compensate this effect, keeping the same response time, it is important to also scale up each node in your cluster a little when deciding to add more nodes to the cluster. Consider optimizing your different layers and scaling up each of your nodes before deciding to increase the number of nodes of your cluster.

Repository size is dynamical
Repository size is dynamical, and the project normally expects, even on the near term, to go through different phases: first migrating legacy repository, new content roll out, archival, and so on. Sizing estimates should cover the different phases, especially in the short term. The injection rate has an obvious impact on the server concerning near future repository sizes, but also around the capability of the different solution layers for handling the throughput, which not only stresses the content storage and database (metadata extraction/upload/rules) but also the indexing layer. This may imply, depending on the amount of document injection, that dedicated nodes are reserved for the injection (which can be on a cluster or not with the front end service nodes) and also the Solr layer is scaled up/out.

Batch/bulk operations
Batch operations are especially important for back-end repository cases where new processes are being executed in relation to custom workflows, custom scheduled jobs, rules, bulk imports, transformations, and so on. In use cases that justify, batch jobs may be configured to execute on dedicated nodes/layers in order to minimize the effect on the other service layers. When using customizations, ensure that you use the official Alfresco job lock service to avoid job execution repetition between nodes in a cluster.

Transformations

To create the documents previews and thumbnails, Alfresco performs some operations on the document. Some of the content need to be transformed into little PNG files (via PDF using both oo.org and ImageMagik); thumbnails and SWF (via PDF using both oo.org and pdf2swf.exe) for preview purposes.

When the Document Transformation Server is not being used, Alfresco is responsible for these transformations, for which it uses OpenOffice. However, OpenOffice can be heavyweight and resource consuming. By default, Alfresco has only 1 thread configured for LibreOffice, but the number of threads can be increased using the port numbers on the JodConverter. Note that document uploads generate thumbnails; when users access the preview of documents, transformations will occur.

If you have lots of uploads and you need faster (concurrent) transformations, consider raising the number of LibreOffice threads. Make sure that you consider this for your memory calculations on these servers. It takes typically 1GB for each LibreOffice thread. The Document Transformation Server can be an option.

Sizing formula for Alfresco/Share nodes

The AWS Reference Architecture (2 Alfresco nodes and 2 Solr nodes) benchmark tests (see the benchmark tests definition section) showed that, when increasing the number of concurrent users to more than 250, response times (especially in Alfresco Share) exceed the acceptable values mentioned on the response times matrix.

Considering a usage scenario similar to the one described in this document, the following formula provides an approximation for calculating the number of Alfresco repository/Share nodes.

Consider the following assumptions:

- User think time of 30 seconds
- Calculations based on Share and CMIS tests
- Calculations based on the OOTB Alfresco deployment
- Alfresco node is running in AWS with 8vCPUs. Additional or faster CPUs can help the repository tier handle more concurrent operations, or to complete operations more quickly.

ALFRESCO REPOSITORY NODES	
NUMBER OF ALFRESCO CLUSTER NODES = NUMBER_OF_CONCURRENT_USERS/125	
CPUS PER NODE	**MEMORY/RAM PER NODE**
2 QUAD_CORE	16 GB

Fine tuning the sizing formula for Alfresco/Share nodes

This formula should be fine-tuned according to each use case.

Adding more cluster nodes **will not improve** the performance of individual transactions, but it will greatly increase the **number of transactions that can be serviced concurrently.** This is especially true if the type of work being done by an Alfresco installation mostly resides in the repository tier. For each use case, you must consider the Sizing factors and adjust the sizing estimates accordingly.

Sizing Guide for the Alfresco database

The Alfresco database is one of the most important factors that can influence global performance of your deployment. If your database is not performing well, nothing else will.

Because Alfresco is database agnostic, we do no provide sizing for the database server in terms of CPU, memory, configuration, and disk requirements. Across this guide, we do provide relevant information on how Alfresco uses the database. Leveraging this information will allow the designated DBA to provide an accurate sizing estimate in terms of CPU, memory, configuration, and disk requirements. Later on this guide, we provide formulas that help to estimate approximate database size.

General database information

The vital components of a transactional system are usually the underlying database files and the log file. Normally the database will require fast I/O, which is slow relative to other system resources such as CPU. In the worst-case scenario (big databases, big number of documents), if the database is too large for any significant percentage of it to fit into the cache, it can result in a single I/O per requested key/data pair.

Normally, both the database and the log are stored on a single disk. This means that for each transaction, the Alfresco database is potentially performing several file system operations:

- Disk seek to database file
- Database file read
- Disk seek to log file
- Log file write
- Flush log file information to disk
- Disk seek to update log file metadata (for example, inode information)
- Log metadata write
- Flush log file metadata to disk

Faster disks normally can help in such situations but there are lots of ways (scale up, scale out) to increase transactional throughput. It is vital to have the database properly sized and tuned for your specific use case.

Be aware that:

- Content is not stored in the database but is directly stored on the disk
- Database size is not affected by size of the documents or the document's content
- Database size is affected by the number/type of metadata fields of the document

Database space sizing formulas

The Alfresco database size is very sensitive to the type of metadata stored. Different databases use different amounts of space to store each metadata-type. A formula that takes into consideration the types of metadata fields, and the space they take on each database, can produce a more accurate estimate.

Consider the following:

- Only metadata field types with values occupy space on the database
- Different types use different amount of space
- Different databases use different spaces to store the different metadata field types

There are two possible ways to determine the approximate size of the database: the general approach, and the detailed approach. These are discussed in the following sections.

General Approach

To determine your approximate database size, consider the following figures.

Database Sizing Variables	
Variable Name	**Expression**
NVER	Average Number of document Versions
NDOC	Number of documents
NDOCMETA	Average number of document metadata fields with values
NFOL	Number of Folders
NFOLMETA	Average number of folder metadata fields with values
USERS	Number of Users
GROUPS	Number of Groups

The sizing formula is based on the estimated number of records that will be created on specific Alfresco tables.

Alfresco Database Tables Number of Records	
Table Name	**Records Calculation Formula**
ALF_NODE	**AN= NDOC * NVER + NFOL**
ALF_NODE_PROPERTIES	**ANP= (NFOL * NFOLMETA) + (NDOC * NVER * NDOCMETA)**
ALF_NODE_STATUS	**ANS= NDOC * NVER + NFOL**
ALF_ACL_MEMBER	**AACL= NFOL**
Alfresco Database Total Records	
TOTAL_RECORDS = AN + ANP + ANS + AACCL	
Alfresco Database Estimated Size	
ESTIMATED_DATABASE_SIZE (kb) = (TOTAL_RECORDS * 2K) + (GROUPS * 2K) + (USERS * 2K)	

The above formula is based on the number of database records. On our benchmarks, we observed that each database record on the referred tables takes about 2K of database space.

Note: the formula does not take in consideration additional space for logging, rollback, redolog, and other vendor specific space requirements that may apply.

Detailed Approach

The Alfresco database size is **very sensitive to the type of metadata stored**. This means that depending on the attribute base type and the amount of data stored on each type, the database size will vary. Our benchmarks only use the OOTB metadata attributes and just fill the title of the document.

If you already have a young Alfresco database containing some documents with metadata fields with values, you can execute the following script in order to determine the database size and approximate future database growth.

See Database Count Script on Appendix D section

Note that having multiple Solr search servers independently tracking the repository and building a full copy of the index increases pressure on the database.

If you plan to scale up by adding further independent Solr nodes, plan your database capacity accordingly.

Content store sizing formulas

Alfresco relies on an abstraction known as a content store to store the actual content of documents in the repository. This abstraction can be backed by a number of different types of storage. The most common storage is a simple disk that Alfresco can use to store and retrieve content. In AWS deployments, S3 is used.

The performance of the storage where the content store resides is vital for the overall performance of Alfresco. The content store is accessed at any time that content is read or written. Therefore, it is important that the content store is not only of sufficient size, but also capable of handling highly concurrent read/write scenarios.

We recommend a fast storage (RAID5) shared with SAN and connected with the Alfresco server via optical fiber or, as an alternative, 100GB Ethernet connection. The high availability and scalability for the binary content storage layers is vendor dependent. Alfresco supports different binary content storage for dedicated advanced file systems/servers.

We will use 2 steps to calculate the approximate size of the Alfresco content store

- Calculate the **binaries occupation size**
- Estimate the size of **previews, images, and thumbnails per MIME type**

Content store sizing – document binaries

Since the documents binary files are directly stored in the content store, we can easily calculate the size they will occupy in the content store.

Alfresco Content Store Sizing Formulas	
Variable Name	**Description**
NDOCS	**Number of Documents**
NVERSIONS	**Number of Versions**
AVGDOCSIZE	**Average Document Size**
Alfresco Content Store Binaries Size	
ESTIMATED_BINARIES_SIZE (kb) = (NDOCS * NVERSIONS * AVGDOCSIZE)	

Alfresco stores more than the content binary files, so we need to take those extra elements in consideration to accurately predict the content store size. Once a file is uploaded, Alfresco will try to create a thumbnail image for that content, if there is a transformer available a thumbnail image is created and stored in the content store. Once the content is accessed inside Share, Alfresco will try to create a preview of that document. The platform does that by creating a series of images, and converting them into a PDF document.

Those images and PDF files are also stored inside the content store and must be considered when estimating its total size. If you have thumbnails and previews enabled (true by default) we need to consider the PDF and IMG versions of each document, as well as the thumbnails (PNG images) generated for those documents.

The next section provides a way to calculate approximate values for those elements.

Content store sizing – extras

To increase the accuracy of the sizing estimates, we need to provide a mechanism for calculating the space taken by these files. We will use well known MIME type specific characteristics while interacting with the Alfresco platform.

Using MIME type constants and input to size the content store extras

We've executed a series of dedicated tests that helped us to predict the approximate size of the images and preview PDF files for each of the common MIME types that are known to have previews in Alfresco.

We've determined the percentage of the root document size that will be used for creation of the images and the PDF preview file per MIME type that is stored inside the Alfresco content store related to the original document.

Alfresco Content Store Extras Sizing Constants			
Mime Type	**Description**	**Preview Images**	**Preview Pdf**
.doc,.docx	MS Word	0.7 %	154%
.xls,.xlsx	MS Excell	**0.4 %**	**47%**
.ppt,.pptx	MS Powerpoint	**25 %**	**0.9%**
.pdf	PDF document	**3.4 %**	**0%**
.txt	Text document	**0.7 %**	**78%**
.gif,.jpg,.tiff,.png	Image Files	**0.8 %**	**20%**

The above percentages help to calculate the size necessary to accommodate previews, images, and thumbnails on the content store. In the next table, we introduce the variables we need to define and apply the complementary formula. Variables are shown in UPERCASE.

Mime Types distribution ratio Variables					
Mime Type	**%**	**Average Size**	**Docs**	**Images Size**	**Pdf Size**
.doc,.docx	DOCXPERCENT	DOCAVGSIZE	NDOC	DOCIMGSIZE	DOCPDFSIZE
.xls,.xlsx	XLSPERCENT	XLSAVGSIZE	NXLS	XLSIMGSIZE	XLSPDFSIZE
.ppt,.pptx	PPTPERCENT	PPTAVGSIZE	NPPT	PPTIMGSIZE	PPTPDFSIZE
.pdf	PDFPERCENT	PDFAVGSIZE	NPDF	PDFIMGSIZE	N/A
.txt	TXTPERCENT	TXTAVGSIZE	NTXT	TXTIMGSIZE	TXTPDFSIZE
.gif,.jpg,.tiff,.png	IMGPERCENT	IMGAVGSIZE	NIMG	IMGIMGSIZE	IMGPDFSIZE
other mime-types	OTHERPERCENT	OTHERAVGSIZE	NOTHER	N/A	N/A

Global Variables	
Variable Name	**Description**
TOTAL_REPO_DOCUMENTS	The total number of documents expected on the repository
GROWTHRATE	The repository annual growth rate

We introduce variables for each MIME type that will be input for the content store extras sizing formula.

- <MIMETYPE>PERCENT – Percentage of documents of <MIMETYPE>
- <MIMETYPE>AVGSIZE – Average document size of <MIMETYPE>
- N<MIMETYPE> - Number of <MIMETYPE> documents
- TOTAL_REPO_DOCUMENTS
- GROWTHRATE

N<MIMETYPE> is calculated with the TOTAL_REPO_DOCUMENTS and <MIMETYPE>PERCENT and represents the estimated number of documents of that mime type.

Calculating the total number of elements of each MIME type

NUMBER OF ELEMENTS PER MIME-TYPE
N<MIMETYPE> = (TOTAL_REPO_DOCUMENTS * <MIMETYPE>PERCENT) / 100

TOTAL SIZE (Kb) OF MIME-TYPE
TOTAL_SIZE_OF_<MIMETYPE> = (N<MIMETYPE> * <MIMETYPE>AVGSIZE)

The next formulas are used to calculate the size of the images created to generate the preview of each MIME type that is known to create a preview:

DOCIMGSIZE = **0.7** * 100/TOTAL_SIZE_OF_DOC
XLSIMGSIZE = **0.4** * 100/TOTAL_SIZE_OF_XLS
PPTIMGSIZE = **25** * 100/TOTAL_SIZE_OF_IMG
PDFIMGSIZE = **3.4** * 100/TOTAL_SIZE_OF_PDF
TXTIMGSIZE = **0.7** * 100/TOTAL_SIZE_OF_TXT
IMGIMGSIZE = **0.8** * 100/TOTAL_SIZE_OF_IMG

Finally, we define formulas to calculate the size of the preview of each MIME type:

DOCPDFSIZE = **154** * 100/TOTAL_SIZE_OF_DOC
XLSPDFSIZE = **47** * 100/TOTAL_SIZE_OF_XLS
PPTPDFSIZE = **0.9** * 100/TOTAL_SIZE_OF_PPT
TXTPDFSIZE = **78** * 100/TOTAL_SIZE_OF_TXT
IMGPDFSIZE = **20** * 100/TOTAL_SIZE_OF_IMG

Alfresco Content Store Extra Size
ESTIMATED_EXTRAS_SIZE (kb) = DOCIMGSIZE + XLSIMGSIZE + PPTIMGSIZE + PDFIMGSIZE + TXTIMGSIZE + IMGIMGSIZE + DOCPDFSIZE + XLSPDFSIZE + PPTPDFSIZE + TXTPDFSIZE + IMGPDFSIZE

If you use other MIME types and you are going to provide a custom transformer, you need to consider them for your content store sizing formulas.

To calculate the total size estimate for the content store, you need to sum the estimated size for the binaries with the estimated size for the extra elements (images and PDF preview files).

Total Sizing Alfresco Content Store
TOTAL CONTENT_STORE_SIZE = ESTIMATED_BINARIES_SIZE + ESTIMATED_EXTRAS_SIZE

Solr servers sizing

Alfresco relies on Solr for indexing and search, which in turn relies on Lucene. Lucene can be very I/O intensive and almost always benefits from a faster I/O subsystem. Consider SSDs, which are much more affordable than they used to be. Alfresco currently recommends a minimum disk read speed of 200MB/s (2). Larger repositories and indexes under heavy load may need to be higher.

Solr nodes are known to have high memory requirements but the CPU is generally only impacted by indexing activities. The CPU footprint is usually low, except when indexing is high (lots of document uploads) and the eventual consistency requirements are high.

Repository sizes should mostly only affect average response times for search operations, and more importantly for certain kinds of global searches very sensitive to the overall size of the repository. **This is the layer most affected by the increase of the repository size.**

Alfresco Solr performance can be broken down into two broad categories: index performance, and search performance.

Indexing performance

Index performance refers to how well Alfresco performs when building or updating an index as content is added or changed.

Solr indexing performance is directly affected by the speed of the repository and the speed of the disks on which the indexes reside. If Solr tracking queries to the repository are slow, then this will result in long index build times. Likewise, if the index is on a slow disk, then merges and updates that occur during the index build process will be slow, and will affect the speed of the index build. Indexing performance is also affected by the performance of the content store, and how quickly the repository can transform content to text for indexing.

Content store and repository performance do not affect the speed of Solr search, but they can affect the performance of indexing new or existing content.

Search performance

Search performance refers to how well Alfresco performs when locating content based on a query.

A key factor (together with fast IO disks) in search performance is the use of caches by Solr. Adding more memory to the system will allow you to increase Solr cache sizes. Conversely, lowering the cache sizes to allow for a smaller memory footprint will have a negative effect on search performance.

Adding more Alfresco Solr nodes can increase the number of concurrent search requests that the system can handle. Alfresco can achieve this in two ways:

Parallel independent indexes

If parallel independent indexes are used, each Solr server maintains a complete copy of the index. All search requests can be serviced by any node, since each node has a complete index of the repository. Parallel independent indexes can therefore service a higher number of concurrent requests, but the performance of each query will be limited by the size of the index.

Solr Sharding

As of Alfresco One 5.1, sharding of the index is supported. The index can be split along a specific dimension into many parts. Sharding the index allows for better horizontal scaling. Each index server maintains a smaller index, which allows for faster searches. Since each index server is tracking only a subset of the index, the index build process can be parallelized, enabling a much faster index build. Alfresco shards the index along the ACL_ID, which keeps both a node and its ACLs on the same Solr shard, speeding up permission checks. As is the case with maintaining multiple independent indexes, sharding can increase the pressure on the database, and so you should plan database capacity accordingly.

When sizing Solr, we should analyze the **types of searches** that will be executed and the authority structure of the corresponding use case. The authority structure has a direct and important impact on the performance of Solr due to ACL checking activities.

Solr nodes sizing formula

Determining the number of Solr nodes in the cluster is highly dependent on the Solr (search) requirements. As verified in the Labs configuration defined in our benchmarks, a good practice is to **align the number of Solr nodes with the number of repository nodes.** That can guarantee response times within the limits shown on the Response Times Matrix.

In our benchmarks, as we increased the number of concurrent users (and concurrent searches), we scaled-out the repository nodes and Solr nodes to guarantee search performance, and to service a higher number of concurrent requests. Note that adding more Solr nodes to the cluster will not increase the performance of each query. That performance is limited by the size of the index.

ALFRESCO SOLR NODES	
NUMBER OF ALFRESCO SOLR NODES = NUMBER_OF_CONCURRENT_USERS/125	
CPUS PER NODE	**MEMORY/RAM PER NODE**
CALCULATED ON IT'S OWN SECTION BELOW	CALCULATED ON IT'S OWN SECTION BELOW

Solr servers memory sizing formulas

Solr nodes need more memory. Indexing impacts CPU usage, but memory is impacted most on Solr nodes. Solr search performance depends on the cache, and bigger repositories require more memory (bigger caches).

By default, there are two cores in Solr:

- WorkspaceSpacesStore
- ArchiveSpacesStore

Each core can have a maximum of two searchers.

You can use a formula to calculate the memory needed for the Alfresco internal data structures used in Solr for PATH queries and read permission enforcement. This formula is applied to each core.

ALFRESCO SOLR MEMORY FORMULA	
Variables Definition	
NUMNODESSTORE	Number of nodes in the store
TRANSACTIONSREPO	Number of transactions in the repository (same for each core)
ACLSREPO	Number of ACLs in the repository (same for each core)
ACLTRANSACTIONSREPO	Number of ACL transactions in the repository (same for each core)
FILTERCACHESIZE	Solr filter cache size (defaults to 64)

QUERYRESULTCACHESIZE	Solr query results cache size(defaults to 64)
AUTHORITYCACHESIZE	Solr authority cache size
PATHCACHESIZE	Solr path cache size

There are 2 ways to determine the values for these variables.

Using your Alfresco database

If an Alfresco database exists, use the following queries to derive the variable values:

select * from
(select count(*) N_Alfresco from alf_node where store_id = (select id from alf_store where protocol = 'workspace' and identifier = 'SpacesStore')) as **NUMNODESSTORE1** ,
(select count(*) N_Archive from alf_node where store_id = (select id from alf_store where protocol = 'archive' and identifier = 'SpacesStore')) as **NUMNODESSTORE2**,
(select count(*) T from alf_transaction) as **TRANSACTIONSREPO**,
(select count(*) A from alf_access_control_list) as **ACLSREPO**,
(select count(*) X from alf_acl_change_set) as **ACLSTRANSACTIONSREPO**;

Basic assumptions

If you are at the initial stage of the project and you don't have an Alfresco database, you must provide approximate values for those variables, but take the following assumptions into account.

NUMNODESSTORE = Estimated number of documents
TRANSACTIONSREPO = Estimated number of documents
ACLSREPO = Estimated_Total_folders, assuming ACL's are applied to the parent node and then inherited.
ACLTRANSACTIONSREPO = Estimated_Total_folders

Solr memory formula for Alfresco data structures in GB

The following formula is used to calculate the memory necessary for the Alfresco data structures associated with one searcher.

SOLR MEMORY FORMULA FOR ALFRESCO DATA STRUCTURES
SDS = 120* NUMNODESSTORE + 32(TRANSACTIONSREPO + ACLSREPO + ACLTRANSACTIONSREPO) /1024/1024/1024 GB

Solr memory formula for Solr caches GB

The following formula is used to calculate the memory necessary for the Solr caches associated with one searcher

SOLR MEMORY FORMULA FOR SOLR CACHES
SOLRCACHES = (FILTERCACHESIZE+QUERYRESULTCACHESIZE+AUTHORITYCACHESIZE + PATHCACHESIZE) * (2 NUMNODESSTORE + TRANSACTIONSREPO + ACLSREPO + ACLTRANSACTIONSREPO)/8 bytes)) GB

Solr total memory requirements formula (GB)

The formulas for SDS and SOLRCACHES provide results for each core with only one searcher. There can be up to 2 searchers per core, depending on the concurrency, so for the values you get from the formula, you should multiply by 4.

SOLR MEMORY FORMULA FOR ALFRESCO DATA STRUCTURES
SOLRMEMORY = 4 * (SDS + SOLRCACHES)

You find detailed information on http://docs.alfresco.com/5.1/concepts/solrnodes-memory.html

Solr server CPU sizing formulas

Search operations are not CPU intensive but indexing and eventual consistency is.

If we focus on indexing and eventual consistency, the CPU capacity has a big impact on indexing performance. Changing the CPU capacity may produce lower values for eventual consistency.

There is no specific formula to size the Solr nodes CPUs as it mostly depends on the use case, and especially on the indexing and eventual consistency requirements.

It is important to consider the amount of indexing activities. In headless content repository scenarios, where there are frequent bulk import operations, CPU capacity will generally be higher, as opposed to collaboration scenarios with lower content ingestion rates.

SOLR CPU FORMULA
SOLR NODES CPUS = 8 CORES (Adjust according to the Indexing and Eventual consistency requirements)

Evaluate your use case. Start with a conservative approach and monitor the CPU usage while executing dedicated benchmarks that generate indexing activities (for example, content ingestion). Adjust the Solr CPU by analyzing your monitoring data. Check the CPU characteristics already used on Solr in our Lab Definition, where we've executed our benchmark tests.

Solr indexes sizing

Not all content types stored in Alfresco affect the platform in the same way. Content types that can be transformed to text and indexed will (by default) require more resources to ingest, and will lead to a larger index.

Take two pieces of content as an example. The first is a Microsoft Word document, and the second is an image file of equivalent size. When the Word document is uploaded to the repository, it will be transformed to text so that its content can be indexed. This leads to more load on the server when the document is ingested, and additional index entries to support full text search. The image file, on the other hand, will not be transformed to text. Even though the two files are of equivalent size, the content type leads to different storage requirements.

Very large documents can also present sizing challenges, consuming large amounts of system resources to generate a text rendition suitable for indexing.

Note that during index optimization, total space required could temporarily be X times the normal index size.

We adopted a strategy based on a constant representing the Index Factor. This constant is used to calculate the size of the index in relation to the total size of the content (excluding preview and thumbnails).

This is an approximation based on an extensive analysis of existing repositories. We've analyzed several repositories and the indexing constant shows values from 4.1 to 7.0. Most of the repositories showed a indexing constant near 5.2, which we will assume as the indexing constant to calculate the approximate index size.

SOLR INDEXING CONSTANT
INDEXING_CONSTANT = 5.2

SOLR INDEX SIZE FORMULA
INDEX_SIZE = (CONTENT_SIZE / INDEXING_CONSTANT) * 2

Since the size of the index will mostly depend on the document's content, it is not simple to define the size of the indexing by considering only the size and MIME type of the source file. For that reason, using the INDEXING_CONSTANT is a more general approach.

Dedicated tests using template documents of each MIME type that are similar to use-case specific documents can help to define a more deterministic formula to calculate the index size. This can be done on a use case basis, by following the same approach taken to calculate the Extra elements on the content store sizing.

Other sizing decisions

Including Ingestion Nodes on your Sizing

Some use cases, especially those with very high content ingestion rates, frequent bulk upload, and scheduled processes, justify the inclusion of a separated ingestion repository node.

This node will be connected to the same database and content store but will not receive any user requests. This node is excluded in the load-balancer configuration that is normally in front of the Alfresco user facing nodes. Ingestion nodes do not take any user requests and don't receive search requests but they do generate document thumbnails.

Content ingestion consumes application server threads (and CPU) from the repository nodes. Having a dedicated bulk ingestion node offloads work from the main (user facing) nodes, maximizing resource usage for real user interactions.

Ingestion nodes are mainly database Solr proxys. They consume CPU when they perform text extraction to send to Solr for indexing.

INGESTION NODE CPU SIZE FORMULA
NCPUS = 1 (Quad-core)
RAM = 4 to 8 GB

Start with 1 CPU (quad-core) and monitor the server resources during upload. Memory consumption is generally low; normally a heap of 4G is enough.

Consider including an ingestion node if:

1. Your project has intensive content ingestion during business hours.
2. You need offload the front-end servers from the text extraction activities.

Including Document Transformation Server nodes in your sizing

To create document previews and thumbnails, Alfresco performs a series of operations on the document. Some content is transformed into small PNG files (via

PDF using both oo.org and ImageMagik), or thumbnails and SWF (via PDF using both oo.org and pdf2swf.exe) for preview purposes.

When the Document Transformation Server is not being used, Alfresco is responsible for these transformations, for which it uses OpenOffice. However, OpenOffice can be heavyweight and resource consuming.

If you have many uploads of Microsoft MIME types and you need faster (concurrent) transformations, consider including a Document Transformation Server instance in your deployment.

The Document Transformation Server is a stable, fast, and scalable solution for high-quality transformations of Microsoft Office documents. It is an enterprise alternative to LibreOffice .

More information http://docs.alfresco.com/5.1/concepts/transerv-overview.html

Appendix

A. Acronyms

Acronym	Definition
RDBMS	Relational Database Management System
ECM	Enterprise Content Management
I/O	Input / Output
SSD	Solid State Disk
AWS	Amazon Web Services
vCPU	Virtual CPU
OOTB	Out of the Box

B. References and external links

1. Alfresco Benchmark Framework: https://github.com/AlfrescoBenchmark/alfresco-benchmark
2. Validating the Architecture: http://docs.alfresco.com/5.1/tasks/configuration-checklist-arch.html
3. AWS EBS Volume Types and Performance: http://docs.aws.amazon.com/AWSEC2/latest/UserGuide/EBSVolumeTypes.html
4. AWS DB Instance Types and Configuration: http://docs.aws.amazon.com/AmazonRDS/latest/UserGuide/Concepts.DBInstanceClass.html
5. Load Testing Share Using the Benchmark Framework: http://summit.alfresco.com/cmis/views/workspace%253A%252F%252FSpacesStore%252Fd2ad6d73-3f4f-4aee-a6b3-75005f83dc90
6. WebDrone binaries and Javadoc: https://maven.alfresco.com/nexus/content/groups/public/org/alfresco/webdrone
7. Alfresco One 5.1 REST API Reference: http://docs.alfresco.com/5.1/pra/1/topics/pra-welcome.html
8. Alfresco One on AWS: Reference Architecture: https://d0.awsstatic.com/whitepapers/aws-alfresco-enterprise-reference-architecture.pdf
9. Alfresco Scalability Blueprint: https://www.alfresco.com/resources/whitepapers/alfresco-scalability-blueprint

C. Lab 3 - Special Lab for Billion documents

Alfresco application server:

Deployed webapps	Alfresco One 5.1, Share 5.1
Operating system	Linux
AWS instance size	c3.2xlarge 8 vCPU 15 GiB memory
Storage	2x80 GB SSD (gp2, max throughput 160MiB/s, burstable to 3000 IOPS)
Number deployed	10

Alfresco Solr application server:

Deployed webapps	Alfresco One 5.1, Solr 4
Operating system	Linux
AWS instance size	c3.8xlarge 32 vCPU 60 GiB memory
Storage	2x320GB SSD (io1, max throughput 320MiB/s, 10K IOPS sustained) Separate EBS volume for Solr content caches (gp2, max throughput 160MiB/s, burstable to 3000 IOPS)
Number deployed	20

Aurora DB server

Deployed webapps	N/A
Operating system	N/A
AWS instance size	db.r3.4xlarge 16 vCPU 122 GiB memory
Storage	EBS Optimized, 2000Mb/s
Number deployed	1

D. Database Count Script

```
-- A script to pull out detailed counts of common types in an Alfresco system
--   The Alfresco database size is very sensitive to the type of metadata stored.
--   Use this script against several times against a young database in order to approximate future
database growth
--
--   This script will take a long time to run against a large database (100M nodes).
--   Set the DBNAME variable, run as 'root' (to access the information_schema) and the script will do
the rest

SET @DBNAME = 'root';

SET @WORKSPACE_SPACESSTORE_ID = (select s.id from alf_store s where s.protocol = 'workspace'
and s.identifier = 'SpacesStore');
SET @ARCHIVE_SPACESSTORE_ID = (select s.id from alf_store s where s.protocol = 'archive' and
s.identifier = 'SpacesStore');
SET @WORKSPACE_VERSION2STORE_ID = (select s.id from alf_store s where s.protocol =
'workspace' and s.identifier = 'version2Store');
SET @SITE_QNAME_ID = (select qn.id from alf_qname qn join alf_namespace ns on (ns.id = qn.ns_id)
where qn.local_name = 'site' and ns.uri = 'http://www.alfresco.org/model/site/1.0');
SET @FOLDER_QNAME_ID = (select qn.id from alf_qname qn join alf_namespace ns on (ns.id =
qn.ns_id) where qn.local_name = 'folder' and ns.uri = 'http://www.alfresco.org/model/content/1.0');
SET @CONTENT_QNAME_ID = (select qn.id from alf_qname qn join alf_namespace ns on (ns.id =
qn.ns_id) where qn.local_name = 'content' and ns.uri =
'http://www.alfresco.org/model/content/1.0');
SET @VERSIONHISTORY_QNAME_ID = (select qn.id from alf_qname qn join alf_namespace ns on
(ns.id = qn.ns_id) where qn.local_name = 'versionHistory' and ns.uri =
'http://www.alfresco.org/model/versionstore/2.0');
SET @THUMBNAIL_QNAME_ID = (select qn.id from alf_qname qn join alf_namespace ns on (ns.id =
qn.ns_id) where qn.local_name = 'thumbnail' and ns.uri =
'http://www.alfresco.org/model/content/1.0');

SET @WORKSPACE_NODES = (select count(n.id) as value
  from alf_node n where store_id = @WORKSPACE_SPACESSTORE_ID);
SET @ARCHIVE_NODES = (select count(n.id) as value
  from alf_node n where store_id = @ARCHIVE_SPACESSTORE_ID);
SET @VERSION_NODES = (select count(n.id) as value
  from alf_node n where store_id = @WORKSPACE_VERSION2STORE_ID);
SET @SITES = (select count(n.id) as value
  from alf_node n where type_qname_id = @SITE_QNAME_ID);
SET @FOLDERS = (select count(n.id) as value
  from alf_node n where type_qname_id = @FOLDER_QNAME_ID);
SET @FILES = (select count(n.id) as value
  from alf_node n where type_qname_id = @CONTENT_QNAME_ID);
SET @VERSIONS = (select count(n.id) as value from alf_node n
where type_qname_id = @VERSIONHISTORY_QNAME_ID);
SET @THUMBNAILS = (select count(n.id) as value from alf_node n where type_qname_id =
@THUMBNAIL_QNAME_ID);
SET @FOLDER_PROPS = (select count(*) as value from alf_node_properties np join alf_node n on
(np.node_id = n.id) where n.type_qname_id = @FOLDER_QNAME_ID);
SET @FILE_PROPS = (select count(*) as value from alf_node_properties np join alf_node n on
(np.node_id = n.id)
where n.type_qname_id = @CONTENT_QNAME_ID);
SET @TXNS = (select count(t.id) as value from alf_transaction t);
SET @DB_SIZE = ( select sum(data_length + index_length) size FROM information_schema.TABLES
where table_schema = @DBNAME);
SET @AUDIT_SIZE = (select sum(data_length + index_length) size FROM
information_schema.TABLES where table_schema = @DBNAME and table_name like 'alf_audit_%');
SET @ACTIVITIES_SIZE = (select sum(data_length + index_length) size FROM
information_schema.TABLES where table_schema = @DBNAME and table_name like 'alf_activity_%');
select
  @WORKSPACE_NODES as workspaceNodes,
  @ARCHIVE_NODES as archiveNodes,
  @VERSION_NODES as versionNodes,
  @SITES as sites,
  @FOLDERS as folders,
  @FILES as files,
  @FILE_PROPS as fileProperties,
  @FOLDER_PROPS as folderProperties,
  (@FOLDER_PROPS/@FOLDERS) as propsPerFolder,
  (@FILE_PROPS/@FILES) as propsPerFile,
  (@VERSIONS/@FILES) as versionsPerFile,
  (@THUMBNAILS/@FILES) as thumbnailsPerFile,
  @TXNS as transactions,
  @DB_SIZE/1024/1024/1024 as dbSizeGB,
  (@DB_SIZE-@AUDIT_SIZE-@ACTIVITIES_SIZE)/1024/1024/1024 as coreSchemaSizeGB;
```