White Paper
Sizing Your Alfresco Installation

Document Version : 1.0

Original Date of Issue : 26/08/2014

Author : Luis Cabaceira

Table of Contents

Introduction

This document purpose is to provide guidelines for the determination of the size (hardware size: CPUs, memory, disk) required for an Alfresco system to meet the requirements of the solution. This includes hardware components sizing but also different product components needed to meet the overall requirements.

Alfresco sizing like others 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 (share front-end, repository, indexing, transformation, storage). 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 Alfresco sizing estimates.

Any recommendations, opinions or findings on this document are based on field experiences and internal Alfresco benchmark results.

The benchmarks referred were run against an Alfresco 4.2.2 + Solr Vanilla infrastructure with no customizations. Any changes in such circumstances and facts upon which this report is based may adversely affect the recommendations, opinions or findings contained in this document.

Target audience

This presentation is targeted for the Alfresco Administrators and Architects responsible for maintaining, scaling and managing an Alfresco infra-structure.

What will you get from this document

After this document you will have a strong foundation on how to size your Alfresco infrastructure according to the specifics of your project. You will be able to evaluate the load on the specific layers of the application and the resources that you may need to achieve the expected performance.

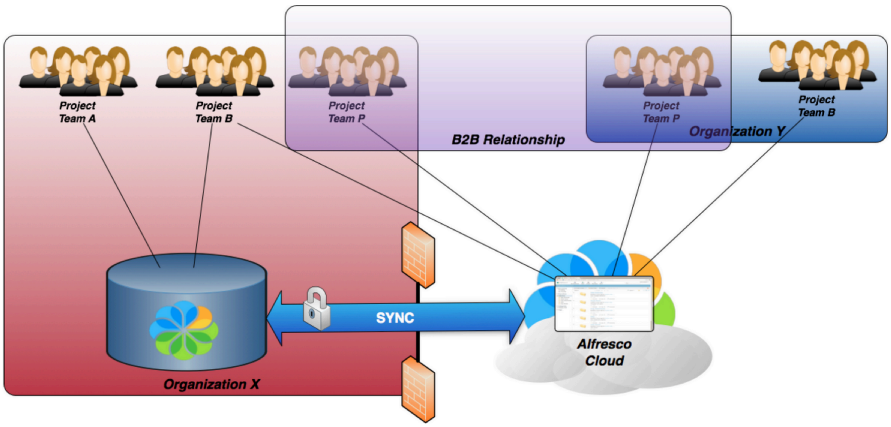
Key Sizing questions

The following list of information/questions represents key information for the sizing exercise. Gathering this information should be the first step in order to do an Alfresco-sizing proposal.

Sizing Area	Question/Information
Use Case	**Gather all information related with 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, sizes and distribution ratios.
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). Time window frames for the operations execution. Averages and peaks.
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, transformations.
Customizations and Integrations	Details around needed 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.
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.

The first 3 shaded rows above indicate the fundamental information. Without this information it is impossible to make an estimate. In many cases organizations are not able to provide the answers to the other questions.

Use Case



At this stage you should gather all information related with how the solution will to be used by final end users. Understanding the purpose of the system is critical to avoid providing values that does not take into account the many subtle (and not so subtle) details that may be hidden behind the rest of the information provided.

You may find out that some solutions aggregate in fact many different use cases that are fairly different and independent making the sizing estimate effort more difficult.

Those situations may also be hard to optimize from a configuration but also from an architecture point of view. For optimized architectures it may be considered as an important improvement to separate those different solutions and use cases into different separated deployments that may be sized also in an independent dedicated manner.

2 Common use cases

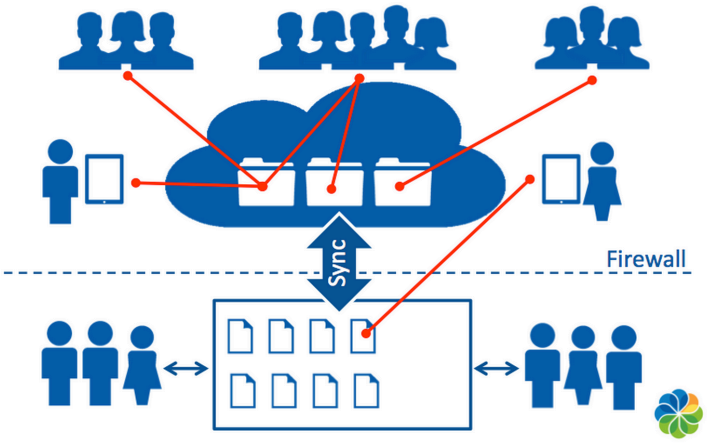
Alfresco use cases vary considerably because of the elasticity of the platform and although we can enumerate some generic common use cases, the details on which each real implementation differs may be very important from architecture and sizing perspective. We can consider that generally alfresco solutions can be classified on one of the 2 following cases:

- Collaboration
- Backend Repository

Some of the fundamental differences we find on our customer's implementations.

Sizing Area	Collaboration	Backend Repository
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 quite big.
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 the custom solution code may live external to Alfresco by using CMIS, public APIs, etc.
Architecture	Architecture options will be in general the standard ones provided by Alfresco (cluster, dedicated index/transformation layers, etc).	Architecture options may vary considerably with more "exotic" solutions being used: proxies, cluster and unclustered layers, sharding of Alfresco repository, etc.
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 it will be very common 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 (Webscripts).
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 or not), custom workflows and scheduled jobs.

Concurrent Users



The concurrent users information is definitely on the top of the importance rank of information to gather for a proper sizing. You will need to size it in a completely different manner, systems with very different concurrency even if they share the same basic architecture skeleton. On the other hand different concurrencies can even demand different architectures.

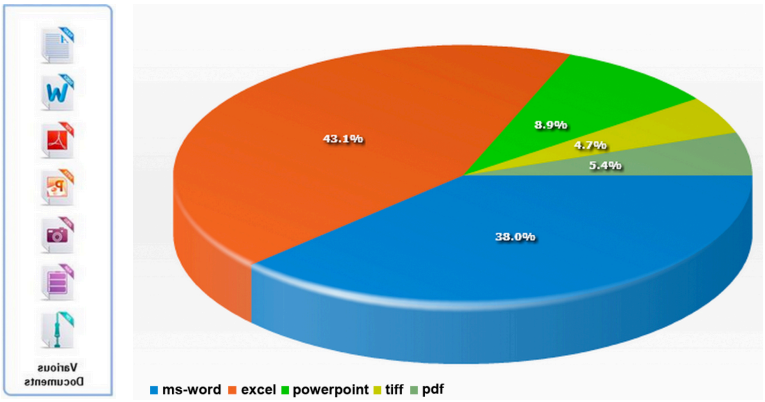
Number of concurrent users (including system users)

Consider detail around average think-time for the number of concurrent users defined

Details on average and peak values.

People often mean considerably different things when they talk about concurrent users. So some agreement is needed on the definition of concurrent user. Generically we consider here 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 requests from N different users. So the term of concurrent user to have meaning needs to have it clarified the think-time being considered. Smaller or bigger think-times being discussed will demand in general an extrapolation of the value to compare to reference cases.

Repository Documents



How many documents in the repository? What types of documents will be used? What are the average document sizes? Distribution ratio of the various types?

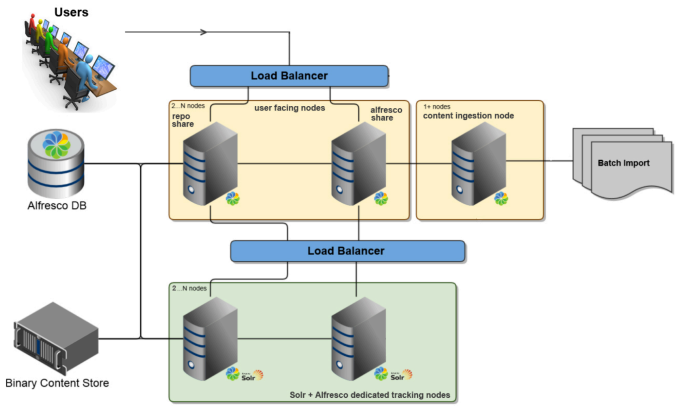
Injection Rate: the repository growth or document upload rate. The document types and sizes will impact transformation, text extraction and indexing. The repository size has impact on performance will grow with the ratio of search operations expected on the solution.

Repository size is also dynamical, and the project may expect even on the near term to go through different phases: first migrating legacy repository, new content roll out, archival, etc. So sizing estimates should cover the different phases especially on 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 on handling the throughput that not only stress the content storage and database (metadata extraction/upload/rules) but also the indexing layer. This may imply depending on the amount of document injection happening that dedicated nodes are reserved for the injection (that can be on cluster or not with the front end service nodes) and also Solr layer scaled up/out.

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 type 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.

Architecture



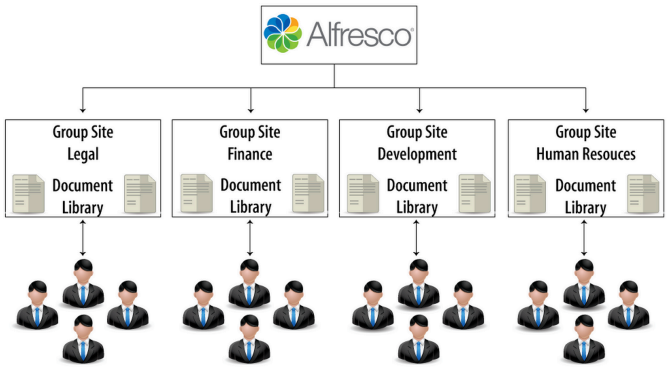
Some typical questions to answer are:

- Is high availability needed? Cluster or failover mechanism is enough?
- Where is load balancing needed?
- Is there a need for a dedicated transformation server?
- Virtualization infrastructure will be used ?
- Is there a need for independent layers/clusters for certain interfaces (injection)?
- Are there reasons to consider a smaller independent layer for Alfresco Share?
- Should dedicated caching elements be introduced?

Architecture expectation will constrain what needs to be sized but also will need to get reviewed according to sizing arising from all the different parts of information gathering and analysis.

Do not expect the architecture to be the result of one direction information flow from the organization to you. Most probably the organizations will expect a coordinated answer (proposal/feedback) on both: architecture and sizing.

Authority Structure



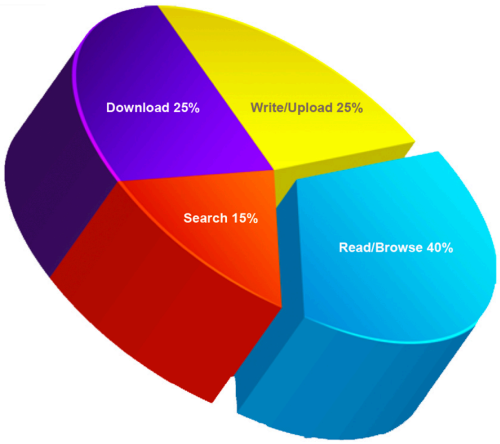
How our user identities organize themselves from a permission control point of view. Basically, the named users and groups, and subgroups, structure. Number of Users, Number of user groups, Group tree/structure depth, Average number of users Per Group , Average number of Groups per user

Consider a fairly complex authority structure, one with considerable number of hierarchy levels in the group structure with the users belonging to many different groups and Consider a very simple authority structure one with a very reduced number of users (most system users) with each user belonging to a fairly small number of groups (5 or less).

Real users corresponding to organization charts usually have a complex authority structure (teams, projects, departments, etc). This is a typical scenario for collaboration use cases. While cases where Alfresco is used as backend without permission control responsibilities will have in general the most simple authority structures.

We know from recent benchmarks comparisons that authority structure has a direct and important impact on performance of SOLR especially. So its important when sizing SOLR the importance of search operations for the solution, the types of searches being executed, the repository size and characteristics but also and equally important the authority structure of the corresponding use case. Collaboration use cases will so be in general more demanding in principle (keeping the other mentioned factors constant) than backend use cases with simpler authority structures.

Operations



Split ratio between browse/download/search/Write, which types of search operations will be executed (global, full text, custom search), Time window frames for the operations execution. Averages and peaks

Browse are “pure” read operations that do not execute searches. They won’t stress the index/search layer and basically will access the database. Each of these operations types will stress different parts of our architecture in a different extent.

Search will stress the Solr layer. Indexing (especially Solr that its not atomic) will in general stress the database but also has a considerable impact at the search level, concurring with search operation for the same resources (disk and memory).

Write/Download should stress the repository/database.

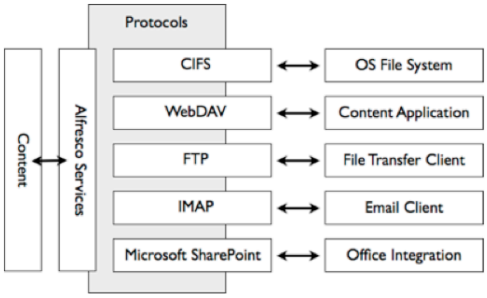
The fact you have 50% of Search operation for example will demand a much bigger sized Solr layer than if you would have in general only 5% of searches. If the write operations ratio is comparable to the reads and downloads we are most of the times talking about cases of massive injection (image capturing, scanning, etc), and those cases may demand a separate dedicated injection layer as already mentioned.

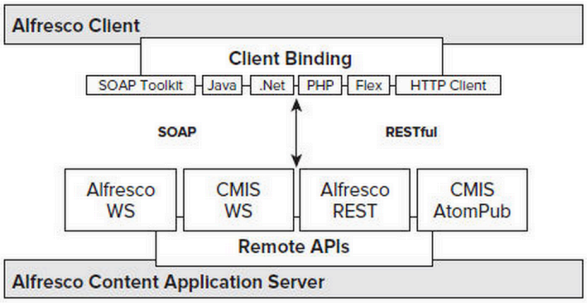
For most of the cases we would expect that read/browse and download operations are the lighter ones. We already discussed the write operations specific case for the Injection Rate topic.

Search operations will also differ from one another and special care should be taken around the type of search operations being executed. Many full text global searches against very large repositories with complex authority structures will certainly have a very important impact on your Solr layer sizing. But content models with few un-tokenized

metadata fields may have better response times, even against a considerably large repository. Finally remember when analyzing the information gathered around operations being executed, the time windows for the execution of those operations. So, as an example, saying a system will have an average of 2000 docs downloaded a day can have quite a different meaning if complemented by the fact this will happen mostly during the first 5 morning hours of the day.

Components, Protocols and Apis





Which components of Alfresco will be used ? Which APIs will be used ? Which protocols will be used?

Some protocols and Apis will be more demanding than others and require a more scaled up or scaled out system to provide acceptable response times. In general some attention should be paid to some problematic protocols. In general this is not Alfresco's fault but because of the protocols definitions themselves. From load point of view CIFS and IMAP are possibly good examples of protocols that affect performance. On the other hand other protocols and interfaces should be considered reasonably light in comparison like FTP and WebDAV.

This topic is in most of the cases (from a sizing perspective) more a matter of proper guiding the organization to the correct and balanced solution, from the combined point of view of their requirements at the functional and sizing level.

Batch Processes

The approximate numbers and types of batch jobs

- Workflows
- Scheduled processes
- Transformations and renditions

Batch operations are especially important for customized cases, where new processes are being executed related to custom workflows, custom scheduled jobs, custom rules and transformations, etc. It's also very hard to have sensitivity at initial stages what will the future custom batch operations be. In any case for those situations when we already have information about it (solution analysis by organization is already advanced) it's an important topic to take into account.

If it is considered important enough, batch jobs might be configured to execute on dedicated nodes/layers in order to minimize the effect on the other service layers. Also care should be taken around customizations to use proper Alfresco cluster job lock service to avoid jobs execution repetition between nodes in a cluster.

Customizations and Integrations

This is definitely a very hard point to measure on initial phases of the project and the best candidate to strong requirements around proper dedicated benchmark and load tests on later phases. And at the same time experience also tell us that it happens too frequently that it's in fact the most important point damaging the correct performance of a system.

Besides all the common warnings around best practices on Alfresco coding, preventive expert code review, unit and load systematic tests, training/certification, etc.

The Alfresco expert responsible for sizing should pay attention to the most probably critical points that may affect the implementation performance in the future considering what's known from the solution requirements. And most especially contextualize the proposed sizing on the adequate terms concerning optimized custom code or (if it really needed) extra scaling up the sizing for taking into account a very possible optimized custom code solution. In any case for very customized scenarios all sizing should be considered really as first test environments setups sizing, and any final sizing for production environments should come from dedicated benchmark results.

- Details around planned customizations
- Customizations impact analysis
- Details on external system integrations.

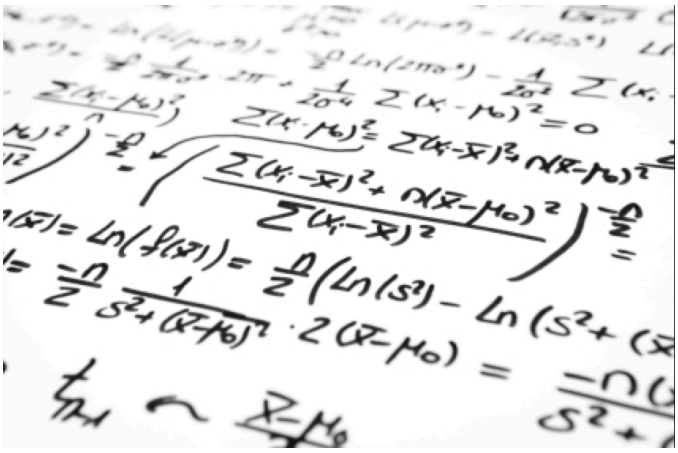
If there's already developed code evaluation on the code performance and possible optimization of the same code.

Response Times Requirements

Last but not least, we need to know the expectations around response time for the different operations being executed. And specifically the important value from sizing point of view will be in general the required response time for peak concurrency (also check the details on response times for the different operations).

From a scalability point of view there might exist requirements around response time degradation with the variation of the different dynamical factors that constrain our solution: concurrent users, but also: repository size, architecture components, etc. Those types of requirements may demand from you a more scaled up/out sizing to cope with the near term possible variations that may affect in a negative way your solution and response times. The same requirements may also demand specific architecture solutions to cope with them.

There is no perfect formula ☺

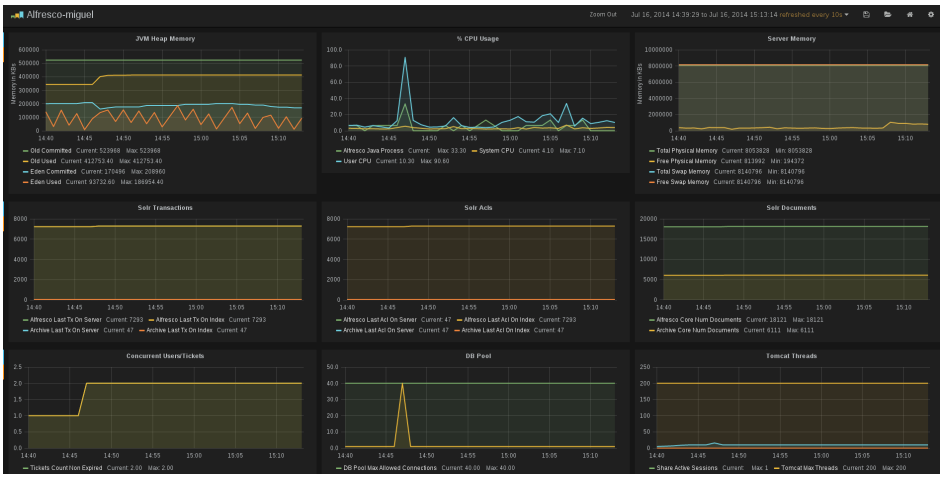


I would love to say that I have a formula that you can apply that by magic will give the exact sizing figures that you need but unfortunately there is no such formula. Sizing and capacity planning depends of several important factors that change from project to project such as:

- Number of customizations
- Types of customizations
- External systems integrations
- Network topology
- Architecture design
- Operating system specifics
- Database vendor
- User behavior
- Other project specific factors

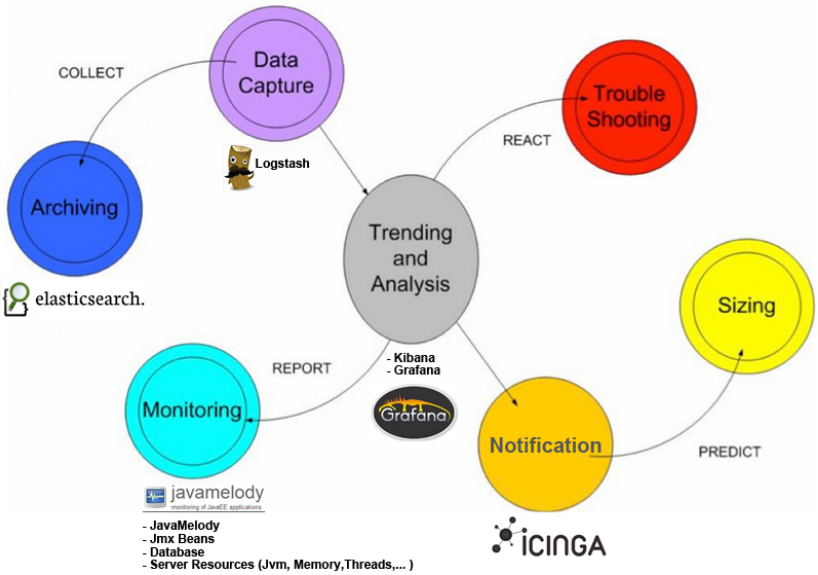
Only with dedicated benchmarks over the final solution on proper setup test environment (reproducing as accurate as possible production conditions, on functional and load terms) its possible to provide empirically tested values. However we can analyze the different layers of a normal deployment and realize what are the factors that will influence them in terms of resource consumption.

Monitoring and Capacity Planning



Before we start with the sizing specific factors I would like to introduce the concepts of Monitoring and Capacity Planning.

Capacity Planning is the science and art of estimating the space, computer hardware, software and connection infrastructure resources that will be needed over some future period of time. It's a mean to predict the types, quantities, and timing of critical resource capacities that are needed within an infrastructure to meet accurately forecasted workloads. Predicting and sizing a system depends on a good understanding of user behavior. The following diagram represents the stages of a capacity-planning scenario.



Miguel Rodriguez has done a project and a presentation in Summit 2014 on how to monitor Alfresco with open source tools. I consider this a very robust solution that you should adopt on your project as it can help you to adjust your sizing predictions based on real data.

The project is open-source and it's made available at https://github.com/miguel-rodriguez/alfresco-monitoring

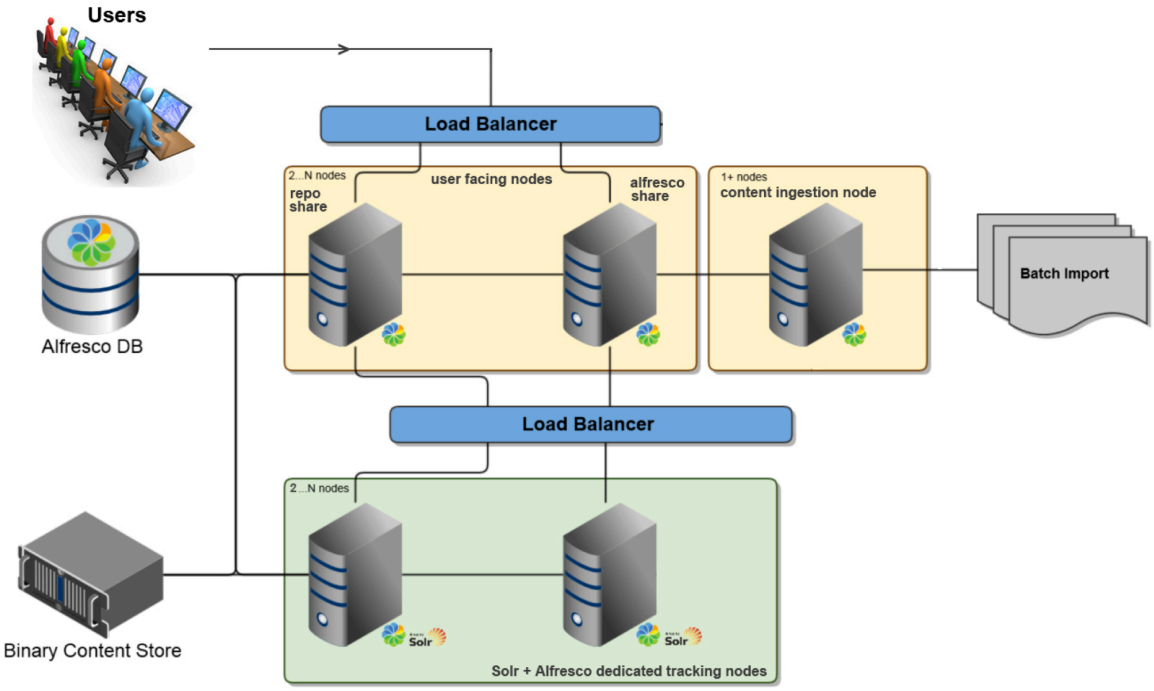
Performing a regular analysis to the monitoring/capacity planning data will help you know exactly when and how you need to scale our architecture, allowing for incremental and continuous improvement. The more data that gets indexed inside elastic search along the application life cycle the more accurate your capacity predictions will be. They represent the "real" application usage and how that usage affects the various layers of your application. This plays a very important role when modeling and sizing the architecture for future business requirements.

The Peak period Methodology is an efficient way to implement a capacity planning strategy allowing to analyze vital performance information when the system is under more load/stress, furthermore it represents YOUR system. The peak period may be an hour, a day, 15 minutes or any other period that is used to analyze the collected utilization statistics. Assumptions may be estimated based on business requirements or specific benchmarks of a similar implementation.

1. Reference Study Architecture

The architecture on the diagram follows includes some Alfresco best practices . It shows a horizontally scalable architecture with Alfresco user facing nodes (2 or more), dedicated tracking alfresco repositories local to the solr servers and a dedicated node for bulk ingestion of content.

Check my blog post on solr tuning for architecture options in http://blogs.alfresco.com/wp/lcabaceira/2014/06/20/solr-tuning-maximizing-your-solr-performance



We will analyze each layer and provide the sizing variables relating those with the sizing information we've gathered previously.

2. Sizing Layers for Alfresco

For and adequate sizing exercise we need to consider all the layers that compose the Alfresco infrastructure. After this presentation you will have a clear idea on the roles and impact of each layer of the application.

The six layers being analyzed are the following:

1 - Database layer
2 - Content Store Layer
3 - Repository Layer (Alfresco user facing nodes)
4 - Repository Layer (Alfresco dedicated tracking nodes)
5 - Repository Layer (Alfresco bulk ingestion nodes)
6 - Solr Layer

During this chapter we will elaborate on each of those six layers. We are not addressing the client layer (Alfresco share or a custom client) as it generally represents less than 10% of the overall server load.

One aspect that affects every layer of the application is the USE CASE. Understanding the purpose of the system is critical to avoid providing a blind value that does not take into account the many subtle (and not so subtle) details that may be hidden behind the rest of the information provided. It's also useful to build sensible assumptions that provide approximate answers to the sizing input not provided by the organization.

2.1. Database layer sizing

The Alfresco database and the file system (where the content store resides) where the content store resides are the brain and heart of Alfresco. Those are the 2 most important layers of every Alfresco architecture, if they are not working well, nothing else will work well. If your project will have lots of concurrent users and operations, consider an active-active database cluster with at least 2 machines.

Most common throughput factor is transactions per second. DB performance in a transactional system are usually the underlying database files and the log file. Both are factors because they require disk 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.

Both the database and the log are on a single disk. This means that for each transaction, the Alfresco DB 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 on such situations but there are lots of ways (scale up, scale out) to increase transactional throughput. It's vital to have the database properly sized and tuned for your specific use case.

To size your alfresco database in terms of space we've done a series of tests by creating content (and metadata) on an empty repository and analyzing the database growth.

Be aware that:

- Content is not stored in the database but is directly stored on the disk
- Database size is un-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

The following factors are relevant to calculate the approximate size for an Alfresco database.

- Number of meta data fields
- Permissions
- Number of folders
- Number of documents
- Number of versions
- Number of users

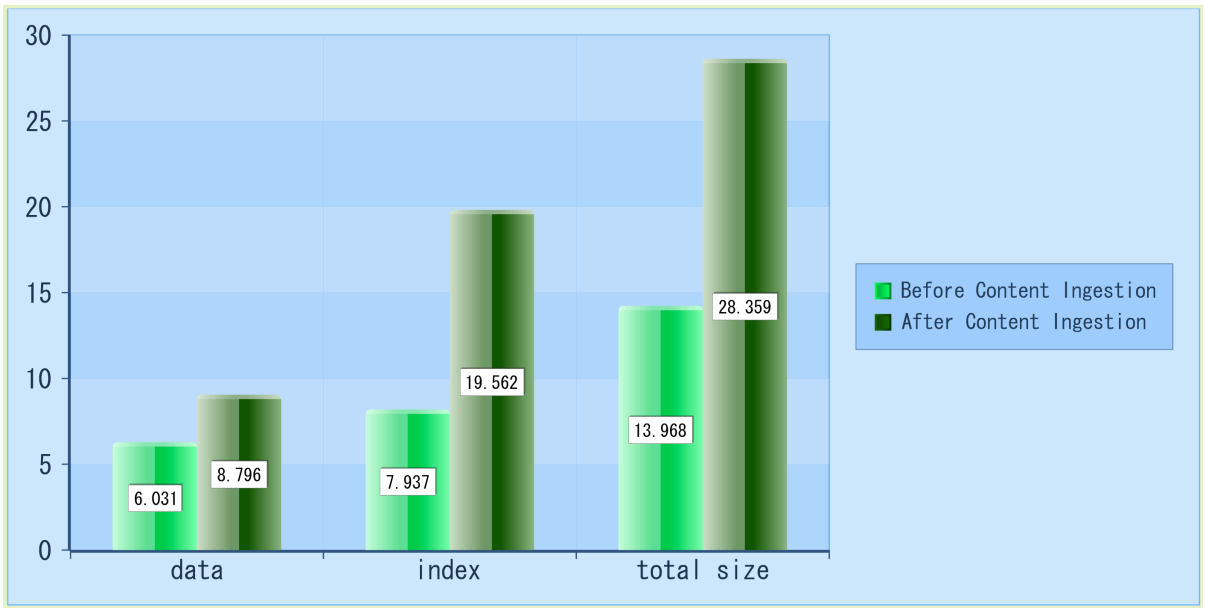
During the tests we've made a bulk import with the following data.

Document creation method	In-place bulk upload
Number of Documents Ingested	148
Total Size of Documents	929.14 MB
Number of metadata fields per document	13 fields
Total number of metadata fields	1924

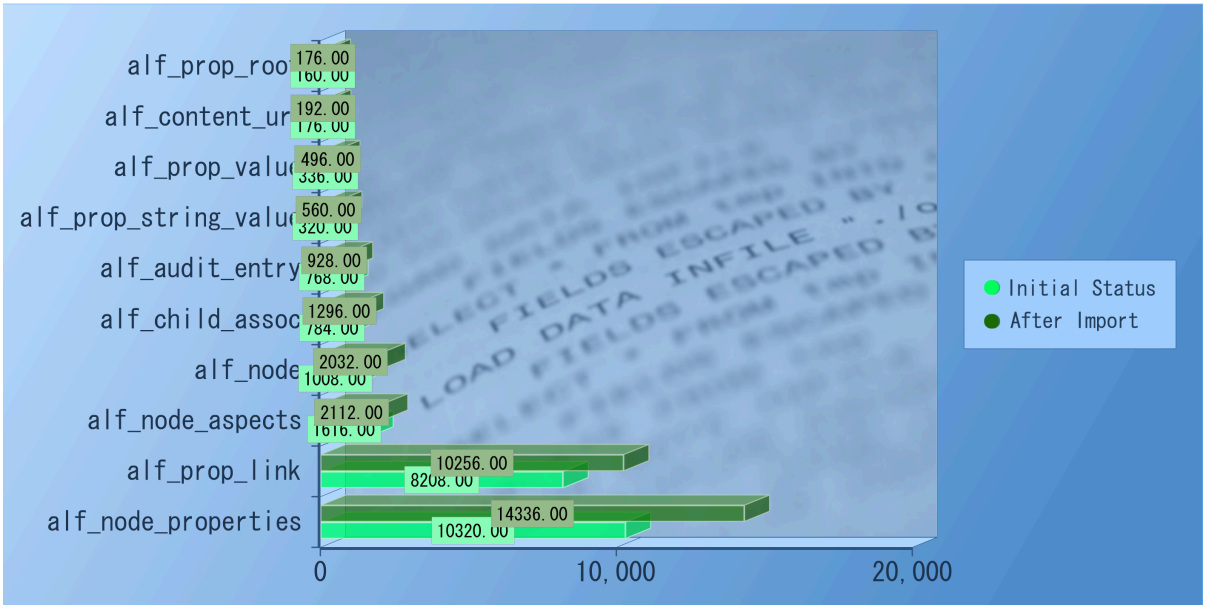
The table below shows the types of documents and its average sizes

Document Type	Extension	Average Size (KB)
MS Word Document	.doc	1024
Excell Sheet	.xls	800
Pdf document	.pdf	10240
PowerPoint presentation	.ppt	5120
Jpg image	.jpg	2048

Checking the diagram below we can see that the database indexes grow more than the data itself. By observing the growth of the database size we've concluded that the average metadata field occupation on the Alfresco database is approximately 5.5 K per metadata field.



Also interesting is to verify the tables that grow in size(KB) after the content ingestion. Note that we are not applying any permission.



2.1.1. Database key questions for your project

To size your database appropriately you must ask the right questions whose answers will help you to determine the database sizing.

0. Estimated number of users in Alfresco
1. Estimated number of groups in Alfresco
2. Estimated number of documents on the first year
3. Documents growth rate
4. Average number of versions per document
5. Average number of meta-data fields per document
6. Estimated number of folders
7. Average number of meta-data fields per folder
8. Estimated number of concurrent users
9. Folder based permissions (inherited to child documents)?

2.1.2. Database sizing formulas

Consider to following figures to determine your approximate database size.

- Average number of document versions (DV)
- Estimated number of folders (F)
- Estimated number of folder metadata fields (standard + custom) (FA)
- Estimated number of documents (D = Number of Documents * DV)
- Estimated number of documents metadata fields (standard + custom) (DA)

The number of records on specific alfresco tables is calculated as follows:

- Number of records on alf_node (TN = F + D * DV)
- Number of records records on node_properties (TNP = F * FA + D * DA)
- Number of records records on node_status = (TNS = F + D)
- Number of records records on alf_acl_member= (TP = D) assuming permission will be set at the folder level and inherited

The approx. number of records in the database will be TRDB = TN + TNP + TNS + TP

The following formula is based on the number of database records. On our benchmarks we've observed that each database record takes about 4.5k of db space.

Database size = TRDB * 4.5K

Alternatively, we can base our formula on the number of metadata fields of the documents, consider 5.5k for each metadata field and use the following formula.

Database size = (D * DA + F * FA) * 5.5K

The 2 formulas provided are only approximations on the size that your database will need and they are based on benchmarks executed against a vanilla Alfresco version 4.2.2.

On top of the formula result you should consider approximately 2k for each user and 5k for each group.

Note that the formulas are not taking in consideration additional space for logging, rollback, redolog, and other vendor specific space requirements that may apply. Also, bear in mind that they only produce approximations to the database size.

2.1.3. Database tuning parameters

In Alfresco the default RDBMS configurations are normally not suitable for large repositories deployments and may result into:

- Wrong or improper support for ACID transaction properties
- I/O bottlenecks in the RDBMS throughput
- Excessive queue for transactions due to overload of connections
- On active-active cluster configurations, excessive latency

Considering that your database layer will be used under concurrent load there are a set of hints that will contribute to maximize your Alfresco database performance.

2.1.3.1 Database Thread Pool configuration

A default Alfresco instance is configured to use up to a maximum of forty (40) database connections. Because all operations in Alfresco require a database connection, this places a hard upper limit on the amount of concurrent requests a single Alfresco instance can service (i.e. 40), from all protocols.

Most Java application servers have higher default settings for concurrent access, and this, coupled with other threads in Alfresco (non-HTTP protocol threads, background jobs, etc.) can quickly result in excessive contention for database connections within Alfresco, manifesting as poor performance for users.

It's recommended to increase the maximum size of the database connection pool to at least [number of application server worker threads] + 75. If tomcat is being considerer

this value is normally 275. The setting is called db.pool.max and should be added to your alfresco-global.properties (db.pool.max=275).

After increasing the size of the Alfresco database connection pool, you must also increase the number of concurrent connections your database can handle, to at least the size of the Alfresco connection pool. Alfresco recommends configuring at least 10 more connections to the database than is configured into the Alfresco connection pool, to ensure that you can still connect to the database even if Alfresco saturates its connection pool. When doing this on the database side, consider the number of alfresco nodes and compute the total of concurrent transactions that your database should be able to handle.

2.1.3.2 Db validation query

By default Alfresco does not periodically validate each database connection retrieved from the database connection pool. Validating connections is, however, very important for long running Alfresco servers, since there are various ways database connections can unexpectedly be closed (for example by transient network glitches and database server timeouts). Enabling periodic validation of database connections involves adding the db.pool.validate.query property to alfresco-global.properties and the query is specific for your database type.

DATABASE	VALUE FOR db.pool.validate.query
MySQL[1]	SELECT 1
PostgreSQL	SELECT VERSION()
Oracle	SELECT 1 FROM DUAL

2.1.4. Database scaling

Alfresco relies largely on a fast and highly transactional interaction with the RDBMS, so the health of the underlying system is vital. Considering our existing customers, the biggest running repositories are under Oracle (most of them RAC).

If your project will have lots of concurrent users and operations, consider an active-active database cluster with at least 2 machines. This can be achieved using Oracle RAC or a

Mysql based solution with haproxy[2] (opensource solution) or a commercial solution like MariaDB[3] or Percona[4].

You can use either solution, depending also you the knowledge that you have in-house. The golden rule is that the response time from the DB in general, should be around 4ms or lower. At this layer we **don't recommend** to use any virtualization technology. The database servers should be physical servers. The high availability and scalability for database is vendor dependent and should be addressed with the chosen vendor to achieve the maximum performance possible.

2.1.5. Database monitoring

Monitoring your database performance is very important as it can detect some possible performance problems or scaling needs.

The following targets that should be monitored and analyzed on a regular base.

- Transactions
- Number of Connections
- Slow Queries
- Query Plans
- Critical DM database queries (# documents of each mime type, …)
- Database server health (cpu, memory, IO, Network)
- Database sizing statistics (growth, etc)
- Peak Period of resource usage
- Indexes Size and Health

[2] http://haproxy.1wt.eu
[3] https://mariadb.org
[4] http://www.percona.com

2.1.6. Database useful queries

The following queries can help you to collect important information in regards to your database.

2.1.6.1. Number of nodes per store

```
SELECT count(*),concat(stores.protocol,concat('://',stores.identifier))
FROM alf_node nodes, alf_store stores WHERE stores.id=nodes.store_id
GROUP BY
concat(stores.protocol,concat('://',stores.identifier));
```

```
1988        user://alfrescoUserStore
1           workspace://lightWeightVersionStore
410767      workspace://version2Store
4054221     workspace://SpacesStore
8           system://system
76621       archive://SpacesStore
```

2.1.6.2. Number of nodes per content-type

```
SELECT
    concat('{',ns.uri,'}',names.local_name) as node_type,
    count(*)  as occurrencies
FROM
    alf_node nodes
JOIN alf_qname names  ON (nodes.type_qname_id = names.id)
JOIN alf_namespace ns ON (names.ns_id = ns.id)
WHERE
    nodes.type_qname_id=names.id
GROUP BY
    nodes.type_qname_id,
    names.local_name
ORDER BY occurrencies desc
```

2.2. Repository user facing nodes sizing

The repository user-facing cluster has the nodes dedicated on serving the usual user requests (reads, manual writes and updates).

The most important factor that stresses these servers is the number of concurrent users hitting the cluster.

2.2.1. User facing nodes key questions for your project

To size your user facing nodes you must ask the right questions whose answers will help you to determine the appropriate sizing.

0. 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.

1. Operations: Percentage of browse, download and write operations.

2. Components, Protocols and API: Which components, protocols and APIs of the product will be used.

3. Response Times: The detail around response time requirements.

0. Concurrent users information is definitely on the top of the importance rank of information to gather for a proper sizing on the user facing nodes. People often mean considerably different things when they talk about concurrent users. An agreement is needed on the definition of concurrent user. Generically we consider here 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. So the term of concurrent user to have meaning needs to have it clarified the think-time being considered. Smaller or bigger think-times being discussed will demand in general an extrapolation of the value to compare to reference cases.

1. Browse/Write/Download operations are those that do not execute searches. They basically will access the database and this database access takes cpu resources on this layer.

2. Components, Protocols and API: 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 some attention should be paid to some problematic protocols. In general this is not Alfresco's fault but because of the protocols definitions

themselves. From load point of view CIFS and IMAP are possibly good examples of protocols that affect performance. On the other hand other protocols and interfaces should be considered reasonably light in comparison like FTP and WebDAV.

3. Response Times: Depending on the expected response times you may have to adjust your sizing for this layer if they are two low.

The User facing nodes perform

1. Transformations for previews when the user access the documents preview, note that when the Alfresco transformation server is being used it will take care of all MS Office documents thumbnails and preview transformation but the rest of the mimetypes will still be handled by these nodes.

2.2.1. User facing nodes Sizing Guidelines

As said before, there is no formula suitable for every installation as cpu consumption will depend on several factors, such as the type of user activities, your architecture, etc.

Having in consideration our experiences and field knowledge from several customer installations we can provide some general recommendations.

Important factors to consider:

- Browsing the repository using a client (share or explorer) performs Acl checks on each item present on the folder being browsed. Acl checks go against the database and occupy a thread in tomcat, causing cpu consumption. As a golden rule consider having a max of 1000 documents on each folder to reduce this overhead.

- No document text extraction occurs on these nodes. Those operations are taken by the dedicated tracking alfresco on the Solr server, this is one of the reasons for having a dedicated tracking alfresco instance as it offloads the user facing alfresco for the transformation and text extraction work.

- Uploads on these servers will still generate thumbnails; note that when the users access the preview of the documents, transformations will occur.

- User operations take application server threads. The more concurrency the more threads will be occupied and the more cpu will be used. Size the maximum number of application server threads (and the number of cluster nodes) according to your expected concurrency.

- Note that each thread normally holds a database connection, so concurrency impacts both alfresco cluster nodes and the database.
- Include the Alfresco scheduled tasks on your cpu calculations, those also require server threads and consume cpu.
- The local repository caches (L2 caches) occupy server memory, size them wisely.
- Customizations on the repository should be carefully analyzed for memory and cpu consumption. Check for unclosed search resultsets, they are a common memory leak.
- User operations take application server threads. The more concurrency the more threads will be occupied and the more cpu will be used. Size the maximum number of application server threads (and the number of cluster nodes) according to your expected concurrency. Include the Alfresco scheduled tasks on your cpu calculations, those also require server threads and consume cpu.

Considering our field experience and our internal benchmark values we recommend adding an extra alfresco node for each 150 concurrent users.

Consider the following assumptions:

- Calculations are based on a (60/40) read/write ratio
- User think time considered (30s)
- Estimations are based on Alfresco internal benchmarks

Number of Nodes = Number of Concurrent Users / 150.

Number of Cpus Per Node = 1 Quad-core (3.2Ghz – 64 bit)

If we assign 2 quad-core cpus to each node, 1 alfresco node can server up to 300 concurrent users.

- Number of Nodes = N Concurrent Users / 300

Whenever you scale out your cluster adding more nodes you should not expect a linear scalability and therefore that the double of the load on a 2 times bigger (more nodes) cluster will necessarily deliver the same response time. In fact in most of the cases you should expect some degradation of the response time arising from some cluster cache synchronization extra load (this effect has been much reduced on latest Alfresco 4 versions). So to compensate this effect, keeping the same response time, it is important in general also scale up a little each node in your cluster.

Transformations considerations on user facing nodes

To create the documents previews and thumbnails Alfresco needs to perform some operation on the document. Some of the content need to be transformed into little PNG files (via PDF: Both oo.org and ImageMagik are used) thumbnails and SWF (via PDF: Both oo.org and pdf2swf.exe are used) for preview purposes.

When the Transformation server is not being used Alfresco is responsible for this transformations, for which it uses Open Office, which 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 jodConverter.

If have lots of uploads and you need faster (concurrent) transformations, consider raising the number of libreoffice threads. Have this in consideration on your memory calculations for these servers, it takes typically 1GB for each Libreoffice thread. Transformation server can be an option.

L2 Cache considerations on user facing nodes

The local repository caches (L2 caches) occupy server memory, size them wisely.

Check my blog post on the alfresco repository caches.

http://blogs.alfresco.com/wp/lcabaceira/2014/05/29/repository-caches/

2.3. Alfresco dedicated tracking instances sizing

The dedicated tracking nodes exist on each Solr server and they are the only reference to Alfresco that Solr knows about. The sizing of these instances is part of the sizing for the Solr server as they reside on the same machine. Note that they normally have their own JVM and application server instance.

The dedicated tracking Alfresco instances perform.

1. Text extraction of the document's content to send to Solr. (Alfresco api call)
2. Allow for local indexing (document's do not have to travel over the wire during tracking)

Important factors to consider:

- Dedicated alfresco tracking nodes serve as a database and solr proxy, they are used for FTS transformations.
- Memory is impacted by the number of documents and transactions.

2.3.1. Dedicated tracking nodes key questions for your project

To size your dedicated facing nodes you must ask the right questions whose answers will help you to determine the appropriate sizing.

0. Operations: Percentage of write operations.
1. Document Details: types, sizes and distribution ratios, injection rate (repository growth rate).
2. Ingestion Rate

0. Write (Upload) operations are those that generate transformations. Text extraction will occur on the tracking nodes to send the document's content to Solr for indexing. Transformations will also occur to generate the thumbnails and previews for each document uploaded.

1. Document Types and size: Depending on the type of documents uploaded a different transformer will be executed. The size of the documents will impact text extraction and also the preview generation, the bigger the documents the more resources those tasks will need. 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 type 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.

2. Ingestion Rate: The injection rate has an obvious impact on the server around the capability of the indexing layer.

2.3.2. Dedicated tracking nodes sizing guidelines

Sizing the resources of these nodes will depend on how fast do you want your text extractions to occur. Indexing (Text extraction) impacts CPU.

Normally these nodes do not have high requirements on memory and CPU but sizing will depend on the number of expecting uploads and transactions on the system. Indexing (Text extraction) impacts CPU and it's API call that occurs exclusively on these nodes.

The best way to size these nodes is to start with a conservative approach for example 1 CPU quad-core and 8GB of RAM and to setup a proper monitoring scenario gathering the relevant information for any necessary adjustments. Cpu and memory requirements are normally not very high at this layer, you should size them according to the number of expecting uploads.

2.4. Bulk Ingestion nodes sizing

When there are scheduled and frequent bulk ingestion processes it makes sense having a dedicated bulk ingestion server. Content ingestion consumes application server threads (and cpu), having a dedicated bulk ingestion node offloads work from the main (user facing nodes) maximizing resource usage for real user interactions.

These nodes are mainly a database solr proxys. They read xml files and there is almost no memory or CPU consumption. Number of documents ingested will impact Solr indexing. Sizing the nodes dedicated to content ingestion is very similar to sizing the tracking nodes, these nodes do not take any user requests and don't receive search requests but they do generate document thumbnails.

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.

2.4.1. Bulk ingestion nodes tuning golden rules

- Disable unneeded services, many standard services are not required when bulk loading content.
- Disable behaviours (Rules evaluations, cm:auditable, versioning, quotas (system.usages.enabled=false))
- Verify the existence of rules on target folder that can delay the speed of the ingestion and disable when applicable.
- Minimize network and file I/O operations
- Get source content as close to server storage as possible
- Tune JVM, Network, Threads, DB Connections

No need to configure cluster caches from this server with the front end. Background index properties and/or content, indexes will catch up from DB transactions.

Benefits: No Cache update/invalidation overhead.

To exclude servers(s) from cluster, do not set cluster name for bulk load servers in alfresco-global.properties alfresco.cluster.name=

During the ingestion process we expect most of the load to occur on the database, the shared file system (I/O) and the user facing Alfresco nodes.

Rule of thumb: Prefer scale up over scale out as it provides simpler deployment and management.

Beware of network latency to RDBMS, **Alfresco to Database is Key,** Latency is key e.g. > 5ms is absolute max

Disable behaviours (Rules evaluations, cm:auditable, versioning, quotas (system.usages.enabled=false))

2.5. Content Store Sizing

The performance of the storage where the content store resides is vital for the Alfresco overall performance. 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. We support different binary content storage for dedicated advanced file systems/servers.

To size your content store consider

- Number of documents (ND)
- Number of Versions (NV)
- Average document size (DS)
- Annual Growth rate (AGR)

Content Store Size (Y1) = (ND * NV * DS) + (Previews and Thumbnails estimated size)

If you have thumbnails and previews enabled also consider the pdf and swf versions of each document and the documents thumbnails (png images). Those are also stored inside the content store.

2.6. Alfresco Solr nodes sizing

We know from recent benchmarks comparisons that authority structure has a direct and important impact on performance of SOLR especially. So its important when sizing SOLR the importance of search operations for the solution, the types of searches being executed, the repository size and characteristics but also and equally important the authority structure of the corresponding use case. Collaboration use cases will so be in general more demanding in principle (keeping the other mentioned factors constant) than backend use cases with simpler authority structures.

Authority Structure

We will name as fairly complex authority structure, one with considerable number of hierarchy levels in the group structure with the users belonging to many different groups. On the opposite side we will say it's a very simple authority structure one with a very reduced number of users (most system users) with each user belonging to a fairly small number of groups (5 or less).

As we said before, real users corresponding to organization charts usually have a complex authority structure (teams, projects, departments, etc). This is a typical scenario for collaboration use cases. While cases where Alfresco is used as backend without permission control responsibilities will have in general the most simple authority structures.

Repository Size

When extrapolation to larger repositories is needed it's important to know that 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. For those cases the ones most affected will also be the ones for which the user-group authority structure is more complex, with users belonging to many groups in average. The layer most affected by the increase on repository size will definitely be the search one (Solr). If your case does not fit the scenarios just described a larger repository size wont probably affect the performance of your system in a such determinant manner. Notice some organizations may impose performance requirements over average operation response times but also limit values for specific operations. For those cases even if search operations for example may represent just a small portion of operations executed, they must be taken into account from sizing point of view in order to fulfill the organization requirements.

To calculate the number of Solr nodes on your cluster you should consider the search response times, the frequency of the search requests and the size of your repository.

2.6.1. Solr Architecture options

There are several architectural options that should be considered depending on the search requirements of the solution.

There are 4 different architecture variations than can be considered while using Solr with Alfresco on a cluster. For the scope of this white-paper we will only be addressing cluster-based configurations that include the following advantages:

Alfresco – > Solr search load balancing

This is the most obvious use case for scalability purposes. Search requests are directed from Alfresco to a pool of Solr instances, each of which contains a full copy of the index and is able to service requests in a purely stateless fashion.

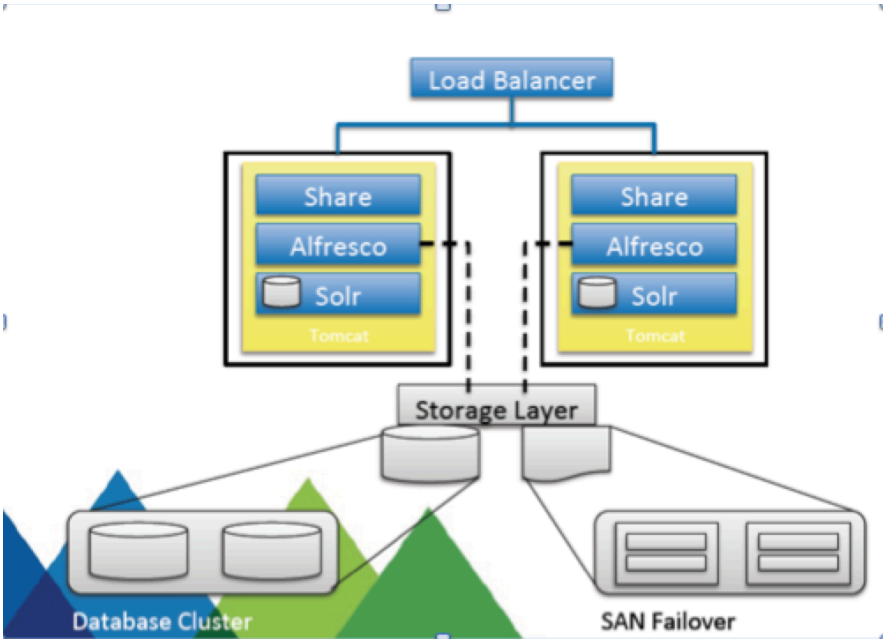
Solr -> Alfresco index tracking balancing

In the other direction, Solr nodes use a load balancer to redirect their index tracking requests to one or multiple dedicated/shared Alfresco nodes. This is useful in case of large indexing load, due to a heavy concurrent write/update scenario.

Consider possibility of having 2 servers (or clusters), each indexing a specific core. You can also have several servers load-balancing the search requests, each one holding a copy of the index with its own tracking server.

Also note the possibility of using the new index replication feature, allowing for logical separation and independent tuning on the search and index layers of Solr.

2.6.1.1. Option 1 – Solr in the same machine as alfresco, non dedicated tracking



On this architecture we have a solr instance deployed on the same application server of both alfresco and share web-applications.

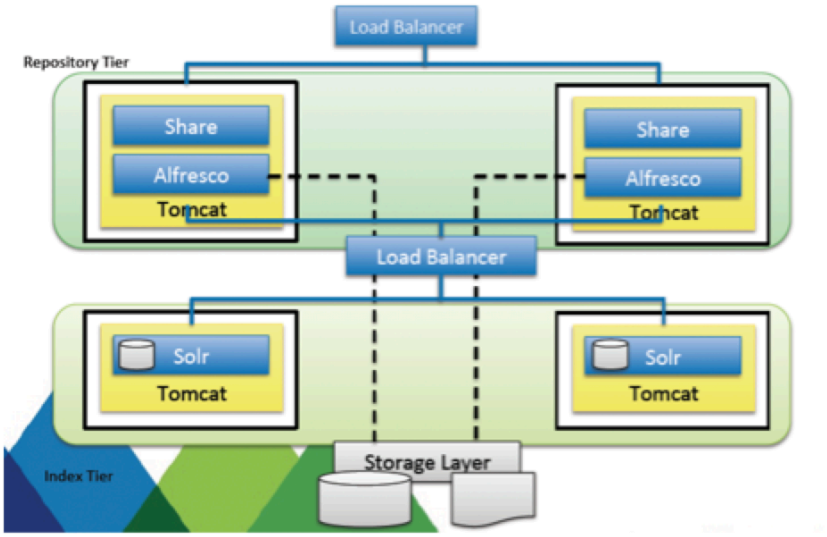
ADVANTAGES

- Easy to maintain / backup.

DISADVANTAGES

- Shared JVM, if Solr crashes both Alfresco and Share become unavailable.
- Shared hardware, Memory is shared between all layers of the application
- When Solr downloads content, there is transformation load to generate the indexing text file (CPU/Memory intensive) on the Alfresco side, having everything on the same box impacts both search and indexing as all the applications are on the same application server sharing its resources like the connection pools, threads, etc.
- Only possible to scale vertically (Only possible to add more CPU and Memory)

2.6.1.2. Option 2 – Solr separated from alfresco, non dedicated tracking



On this architecture variation we have the Solr instances deployed on separated machines and application servers from alfresco and share.

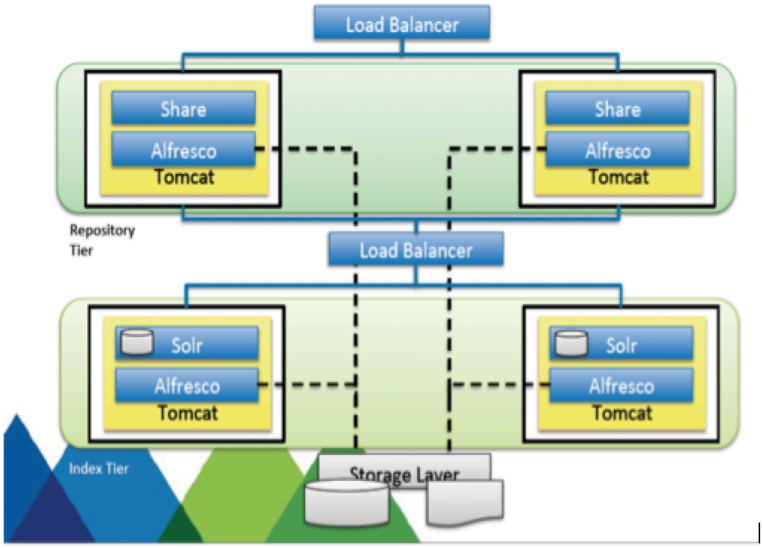
ADVANTAGES

- Simple upgrade, administration and maintenance of Solr server's farm
- Allows for vertical and horizontal scalability
- Introduces the ability to load balance the queries
- Ready for Future Solr sharding feature
- It's expected that alfresco will support, on a near future the ability to slit the index on different solr instances that will lead to increased performance, this architecture is ready to implement that feature.

DISADVANTAGES

- Remote Tracking can stress the network, if network problems occur solr performance gets affected.

2.6.1.3. Option 3 – Solr server with a dedicated tracking alfresco instance



On this architecture variation we use dedicated alfresco instances on the solr servers that are only used for indexing and do not receive or process any user's requests. These local alfresco instances take care of important CPU/Memory intensive operations such as the transformations and the overall tracking and indexing actions. With this scenario the repository Tier is released from those operations resulting on a overall performance gain. This Alfresco instances are not part of the cluster and do not require ehcache configuration.

Note: When Solr requests Alfresco for the content to be indexed, it's the Alfresco server that is responsible for perform the content transformation onto a plain text file, only then the content is sent to Solr for indexing. This is an IO/CPU/Memory intensive operation that can decrease the overall alfresco performance.

ADVANTAGES

- Indexing operations offloaded from repository and client tier
- Dedicated Alfresco for transformation operations (text extraction)
- Allow for specific tuning on the index tier and on the repository tier considering the use cases.
- Allows for Vertical and horizontal scalability
- General performance increase

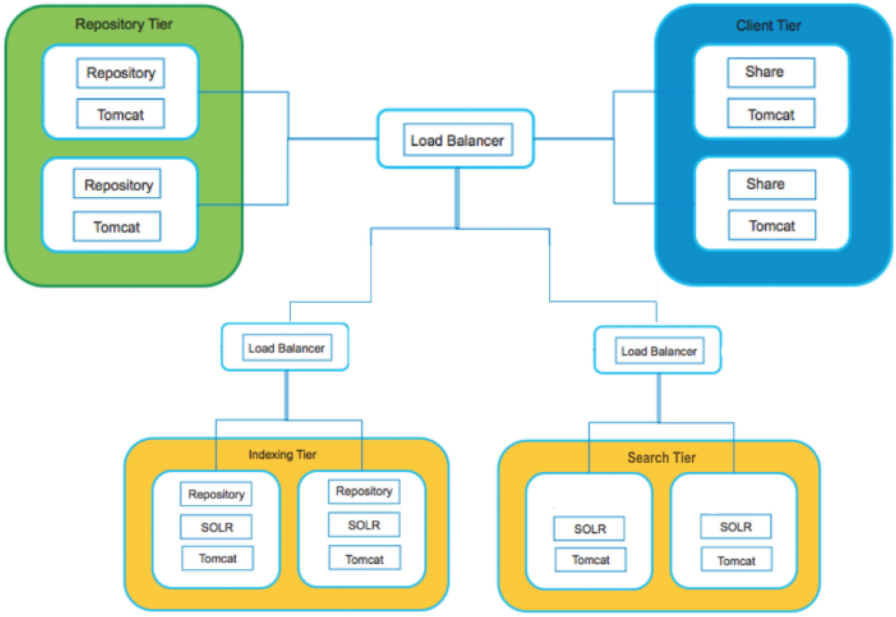
DISADVANTAGES

- None, in my opinion one of the best options

2.6.1.4. Option 4 – Logical Separation of indexing and search

This architecture variant its not yet fully supported by Alfresco (not fully Q/A) but it can be implemented for alfresco version 4.1.8 and above.

This architecture variation is the one that can perform better allowing for independent configuration and optimization on both the search and indexing operations concerning Solr. It uses the index replication feature to sync the indexes on both layers.



ADVANTAGES

- Indexing operations offloaded from repository and client tier
- Dedicated Alfresco for transformation operations (text extraction)
- Allow for specific tuning on the index tier
- Allow for specific tuning on the search layer
- Allows for Vertical and horizontal scalability
- General performance increase while indexing and searching at the same time

DISADVANTAGES

- None, in my opinion one of the best options

2.6.2. Solr Nodes Sizing guidelines

Collaboration use cases will so be in general more demanding in principle (keeping the other mentioned factors constant) than backend use cases with simpler authority structures.

When sizing Solr we should analyze the **types of searches** that will be executed and the authority structure of the corresponding use case

Important to notice that 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 on repository size.

Notice that some organizations may impose performance requirements over average search operation response times but also limit values for specific operations.

The solr nodes are the one's that need more memory. Indexing impacts CPU usage, but the memory is the most impacted factor on these servers. Solr performance depends on the cache, bigger repositories require more memory (bigger caches).

2.6.2.1. How to size solr nodes memory

You should use the online formula published by alfresco to calculate the necessary memory for your solr nodes. Note that the formula provided gives results for each core and it considers only one searcher. There can be up to 2 searchers per core, depending on the concurrency, so the values you get from the formula you should multiply by 4.

http://docs.alfresco.com/4.2/concepts/solrnodes-memory.html

Total Memory = Result x 2 cores x 2 searchers

2.6.2.2. Sizing Solr Nodes Cpu

Solr nodes are known to have high memory requirements but CPU generally is not that affected. When you are re-indexing your CPU usage will be impacting, but during normal operations CPU usage will not be very high. Start with a conservative approach with 1 Quad-Core cpu and adjust considering your monitoring data.

2.6.2.3. Solr Indexes Size

You can expect the size of your indexes (when using FTI) to be between 40-60% the estimated size of your content store.

How to Minimize Solr memory requirements

To minimize memory requirements on solr you can :

- Reduce the cache sizes and check the cache hit rate.
- Disable ACL checks.
- Disable archive indexing, if you are not using it.
- Since everything scales to the number of documents in the index, add the Index control aspect to the documents you do not want in the index

3. A real life sizing sample

SIZING AREA	INFORMATION
USE CASE	Collaboration
CONCURRENT USERS	600
REPOSITORY	4M docs/ Avg doc size is 3mb/ Ingestion Rate is 75000 docs /Year
ARCHITECTURE	High availability required on Solr and Alfresco user facing nodes
AUTHORITY STRUCTURE	8000 named users, 100 groups, user belongs to 4 or 5 groups, Group hierarchy max depth 4.
OPERATIONS	
COMPONENTS PROTOCOLS AND APIs	
CUSTOMIZATIONS IMPACT	None
BATCH OPERATIONS	Ad Synchronization, External loaded dictionary data
RESPONSE TIMES	Not Defined

ALFRESCO NODES	CPU	JVM MEMORY
2	2 Quad-Core	8 GB

SOLR NODES	CPU	JVM MEMORY
2	1 Quad-Core	20GB

4. C.A.R – Capacity of Alfresco repository

How do we determine the throughput of an Alfresco repository server ?

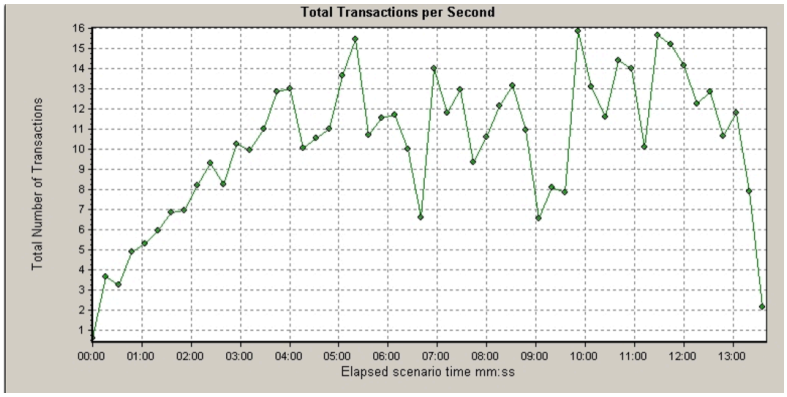
A ECM repository is very similar to a database in regards to the type of events that they manage and execute, specially when we think about "transactions per second" where a "transaction" is considered to be a basic repository operation (create, browse, download, update, search ,delete, acl_change_set, …).

The aim is to define common standard figure that can empirically define the capacity of a repository instance.

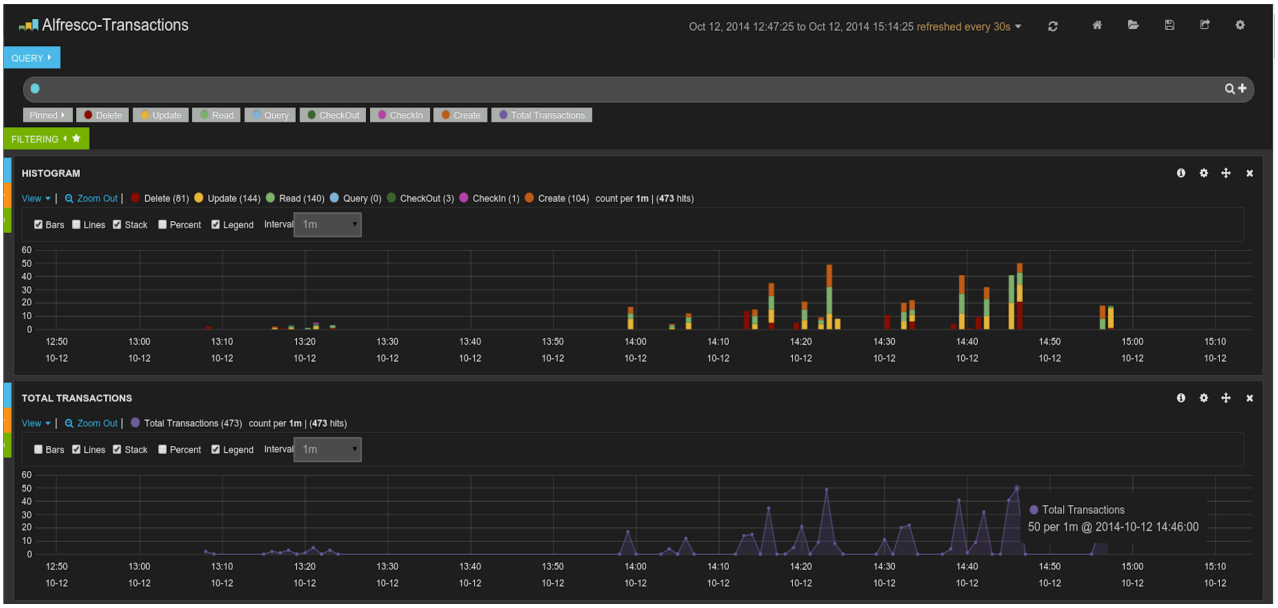
Let's assume that the capacity of and Alfresco repository can be represented by the following sentence.

"The maximum number of transactions that can be handled in a single second before degrading the expected performance"

The unit of measure of would be is transactions per second. The image below shows the average number of transactions per second of a real alfresco-repository during business hours. This represents the server heartbeat in regards of transactions per second.



By configuring the Alfresco audit to log the target transactions we can have your monitoring details on the transaction and types of transactions that are executed on our repository.



Let's consider now 3 new figures:

EC = The expected concurrency represented in number of users.

TT = user think time represented in seconds

ERT = Expected/Accepted response times

Consider ERT to be an object representing the response times being considered including the weight of each type and the layer that it affects. It takes known repository actions as arguments.

OPERATION	VALUE	WEIGHT	LAYER	CPU	MEM	I/O
Download	3 sec	20	Repo			
Write/Upload	5 sec	10	Repo/Solr/DB			
Delete	3 sec	5	Repo/DB			
Browse/Read	2 sec	50	Repo/Solr/DB			
Search Metadata	2 sec	10	DB/Solr			
Full Text Search	5 sec	5	Solr			

Note that when we need to decrease the ETR means the necessity to scale (out or up) the capacity of the alfresco repository server or scaling the related components referenced in the ERT object.

With the inclusion of new variables we can tune our C.A.R. definition and redefine it as.

*"**Number of transactions that that the server can handle in one second under the expected concurrency(EC) with the agreed Think Time (TT) ensuring the expected response times(ERT)** "*.

4.1. Some initial lab test results

We've executed some lab tests, one server running Alfresco and another running the database, simple collaboration use case on a repository with 5000 documents.

Alfresco Server Details

- Processor: 64-bit Intel Xeon 3.3Ghz (Quad-Core)
- Memory: 8GB RAM
- JVM: 64-bit Sun Java 7 (JDK 1.7)
- Operating System: 64-bit Red Hat Linux

Test Details

- ERT = The sample ERT values shown before on the presentation
- Think Time = 30 seconds.
- EC = 150 users

The C.A.R. of the server was between 10-15 TPS during usage peaks. Through JVM tuning along with network and database optimizations, this number can rise over 25 TPS.

4.2. Predicting the future

If you need to predict the resources that your system will need to cope with a specific future load (high number of transactions per second) you should.

- Re-evaluate your sizing introducing new factors.
- Take a dynamic / use case oriented approach to identify a formula, built on a system of attributes, values, weights and affected areas.
- Introduce one or more ERT objects representing use case specific response times and operations acting as influencers on the server throughput.