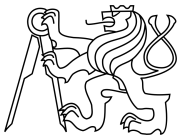
CZECH TECHNICAL UNIVERSITY IN PRAGUE
FACULTY OF INFORMATION TECHNOLOGY



ASSIGNMENT OF BACHELOR'S THESIS

Title:	Load Balancing and Failover Features of Documentum Content Server
Student:	Štěpán Staniek
Supervisor:	Mgr. Jiří Robenek
Study Programme:	Informatics
Study Branch:	Information Technology
Department:	Department of Computer Systems
Validity:	Until the end of summer semester 2017/18

Instructions

1. Describe the possibilities for a specific use of high availability mode in Documentum.
2. Install and configure the Content Server in high availability mode.
3. Select an appropriate tool for test availability of the Content Server and suggest testing scenarios.
4. Perform availability testing and collect the results.
5. Evaluate the measured test results and assess the suitability configuration.

References

Will be provided by the supervisor.

L.S.

prof. Ing. Róbert Lórencz, CSc.
Head of Department

prof. Ing. Pavel Tvrdík, CSc.
Dean

Prague October 6, 2016

Czech Technical University in Prague
Faculty of Information Technology
Department of Computer Systems

Bachelor's thesis

Load balancing and failover features Documentum

Štěpán Staniek

Supervisor: Mgr. Jiří Robenek

5th January 2017

Acknowledgements

This way I would very much like to thank the mentor of my thesis Mgr. Jiri Robenek for his valuable advice in regards of writing the thesis.

Declaration

I hereby declare that the presented thesis is my own work and that I have cited all sources of information in accordance with the Guideline for adhering to ethical principles when elaborating an academic final thesis.

I acknowledge that my thesis is subject to the rights and obligations stipulated by the Act No. 121/2000 Coll., the Copyright Act, as amended, in particular that the Czech Technical University in Prague has the right to conclude a license agreement on the utilization of this thesis as school work under the provisions of Article 60(1) of the Act.

In Prague on 5th January 2017 .....

Czech Technical University in Prague
Faculty of Information Technology

Citation of this thesis

Staniek, Štěpán. *Load balancing and failover features Documentum.* Bachelor's thesis. Czech Technical University in Prague, Faculty of Information Technology, 2017.

Abstrakt

Documentum je Enterprise content management system pro správu velkého objemu dokumentů používán velkými korporacemi. Tento system umožňuje přístup tisícům uživatelů v jakoukoliv dobu a je opravdu důležité zajistit dobrou stabilitu a dostupnost systemu napříč celou společností. Využití režimu High Availability pro konfiguraci Documentum systemu nabízí vhodné řešení jak toho docílit. Pro testování High Availability módu a rozložení zátěže, bylo připraveno a konfigurováno prostředí Documentum Content Server ve verzi 6.5. Pro potřebu testování funkcionality rozložení zátěže byla vytvořena jednoduchá Java aplikace. Funkce této aplikace je simulovat požadavky uživatelů, ke zjištění, jak jsou distribuovány přístupy ke Content Serveru. Za účelem testování výpadku a obnovení funkce instance Content serveru byl vytvořen příklad s cílem ověřit chování systemu při výpadku serveru. Praktické ukázky a testování mi daly možnost poodkrýt rozsáhlé možnosti při konfiguraci systemu Documentum. A to navzdory skutečnosti, jak je system Documentum robustní, že z výsledků jsou patrné problémy staré verze 6.5 systemu Documentum při snaze obnovení z havárie.

Klíčová slova ECM, DFC, DMS, Documentum, Content Server, High Availability, Connection Broker, Load balancing

Abstract

Documentum is Enterprise content management system for managing huge amount of documents usually used by huge corporation. It is usually accessed by thousands of users at any given moment and it is really important to assure a good stability and availability of system across an organization. Configuration of Documentum system in High availability mode offers the right solution of how to do it. Sample workspace of Documentum Content Server version 6.5 was prepared and configured for high availability and load balance. In order to test load balancing functionality a small Java application was prepared to simulate users' requests to see how sessions were distributed across content servers. In order to test fail-over and recovery functionalities of a content server instance an unexpected failure was created to verify a behavior of the system. The practical testing gave me an opportunity to see extensive possibilities in regards of configuring Documentum. Despite the fact of how robust Documentum system is, from the results of the testing is visible that there were issues with the older version 6.5 for recovering from a crash.

Keywords ECM, DFC, DMS, Documentum, Content Server, High Availability, Connection Broker, Load balancing

Contents

List of Figures

List of Tables

Introduction

I have been interested for more than two years in an area of Enterprise content management more precisely in capturing data from paper, dynamic biometric signature and managing of unstructured information. I considered this thesis as a very good opportunity to extend my skills with Documentum system at current employment at NNIT Czech Republic. The thesis is based on a typical customer requests and providing useful information with regards to behavior of Documentum Content Server in version 6.5SP3.

Documentum is an advanced Enterprise content management platform. The main task of Documentum is to solve unstructured information management problems using relational database technologies. Unstructured information refers to information that does not have a formal data structure – documents, images, audio, video, etc. With Documentum systems we are able to capture them, organize, store, maintain, and selectively publish the thousands of pages product sheets, manuals, etc. The core of this is Content Server which allows us to manage the content.

Our business at NNIT is based on customers from Life science area, but this thesis is independent of use Documentum by customers. Our team at NNIT takes care of many Documentum platforms including many customized applications connected to content servers. The main mission is to keep maximal availability of applications. Very typical customer request is to suggest the best configuration of content servers to be able to assure the maximal high-availability and well-balanced utilization of content servers. This thesis is a demonstration of how the testing should look like. The goals of this thesis is to explain and describe the process of the testing, show some possibilities for configuring of Content Server for high availability and also prepare sandbox Documentum environment for testing purposes. As it has been already mentioned, a behavior of the load balancing will be tested on Content Servers and in the second part a behavior of the system while an unexpected failure

occurs will be monitored including a recovery behavior of a crashed content server.

The thesis is divided into three chapters. The first chapter is a general description of the system and its product Documentum Content Server. Second chapter deals with a specific mode for High Availability configurations in the environment of Content Server. The last chapter is about developing the required tools, testing and evaluation of measured results.

Chapter 1

Content Server

EMC Documentum Content Server is a powerful, robust, and scalable enterprise content management system that provides advanced content management and process management functions that let you organize, control, and access all your information assets in your organization. Content Server manages content stored in object-based repositories. A repository consists of two main components: a storage that stores native content files, and a relational database management system (RDBMS) that stores properties of these content files called metadata, such as document owner, version, and creation date. Metadata describes the content object and its relationship between other objects in repository, and is used to manage and search for content. [1]

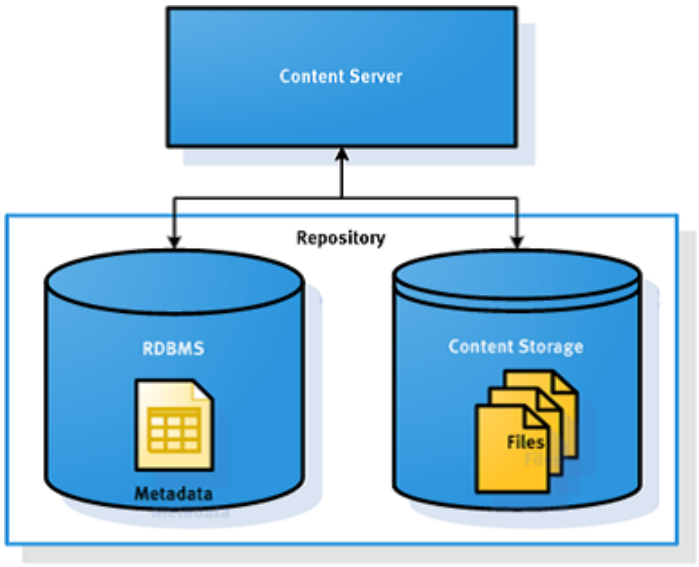


Figure 1.1: Content Server fundamental architecture. [1]

Content Server itself consists of several distinct processes and components that are mostly transparent to the user during installation:

- Application server - Content Server uses a private embedded application server as a container for Java Method Server (JMS), Accelerated Content Server (ACS), and other components.
- Java Method Server (JMS) - Java Method Server (JMS) is a customized version of JBoss that executes Content Server Java methods. One Java Method Server is installed with each Content Server installation.
- Accelerated Content Services (ACS) server - Accelerated Content Services (ACS) Server is a lightweight server that is automatically created during Content Server installation. The ACS server reads and writes content for web-based client applications using HTTP and HTTPS protocols. ACS servers do not modify object metadata but write content to storage areas.
  ACS aim to improve performance for retrieving content through WDK applications. Normal content retrieval using a web client will first bring content to the application server and then to the web client. If ACS is used, the web client is instructed to pull the content directly from ACS (which is located on the content server by default) which means that less traffic is going thru the application server which has positive impact on the performance.
- Documentum Foundation Classes (DFC) - provides the programming interface that client applications use to communicate with Content Server.

1.1 Connection broker

Content Server clients connect to Content Server through connection brokers. A connection broker is a process that provides client sessions with Content Server connection information, such as their IP addresses and port numbers, as well as proximity values of their network locations. The connection brokers that handle a client connection request are defined in the dfc.properties file of the client. When a user or application requests a repository connection, the request goes to a connection broker identified in the client dfc.properties file. The connection broker returns the connection information for the repository or a particular server identified in the request. Connection brokers do not request information from Content Servers, but rely on the servers to regularly broadcast their connection information to them. When Content Server starts, it automatically broadcasts information about itself to one or more connection brokers. Each connection broker that receives the broadcast adds the Content Server to its list of available servers. The information on connection broker is configured in the server config object (dm_server_config)

of the server. Each Content Server installation must have at least one connection broker. The first connection broker is started as part of the installation process. When a client application wants to connect to a repository, the following occurs:

1. The client contacts the connection broker and requests the information it needs to connect with a Content Server for the requested repository.
2. The connection broker sends back the IP address for the host on which the Content Server resides and the port number that the Content Server is using.
3. The client application uses that information to open a connection to Content Server.

1.2 Content Server installation models

Content Server and repositories can be installed and configured in many different ways to meet various content management requirements. Content Server installation supports the following types of configurations:

- Basic - In this basic model, Content Server, repository (including an RDBMS and a content storage), and connection broker are all installed on a single host. This is the simplest and most straightforward way of implementing Content Server, and is typically used in development and test environments.
- High-availability (HA) - In this model, multiple redundant Content Server instances and components are installed on a single host or multiple hosts and configured to eliminate single-point-of failure and achieve high-availability.
- Distributed - In the distributed model, one or more repositories span multiple hosts and are configured to be accessed from multiple sites.

1.3 Basic installation model

In the basic installation model, Content Server, repository (including an RDBMS and a content storage), and connection broker are all installed on a single host. This is the simplest and most straightforward way of implementing Content Server, and is typically used in development and test environments. In production environments, Content Server and RDBMS are almost always installed on different hosts for better performance.

Figure 1.2 shows the key interrelated components in the basic installation model and the order in which they are installed. Components that are

installed as a part of the Content Server installation process covered in this document are highlighted in yellow. Dotted lines indicate connections that are not persistent.

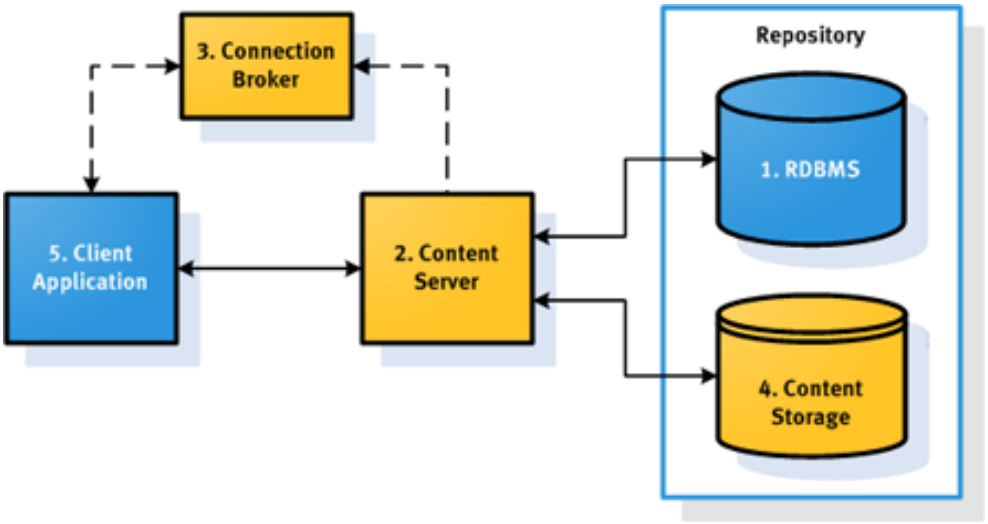


Figure 1.2: Content Server Basic installation model. [1]

1. Relational database management system (RDBMS), Microsoft SQL Server, DB2 or Oracle Database - which stores content metadata. This is a part of the repository and a prerequisite software component that must be installed before prior to Content Server installation.
2. Content Server, which manages content stored in the repository.
3. Connection broker, which provides connection information to client applications.
4. Content storage, which stores native content files.
5. Content Server client application, which provides a user interface for accessing Content Server functionalities and managing repositories.

Chapter 2

High availability

No matter what kind of software product, Customers major factor to choice is how fast and available their application is. Content Server support two options for High availability mode.

1. Failover - In a failover setup, if one of the Content Servers fails, the other Content Servers in the failover setup continue with the service.

2. Load balancing - Load balancing involves operating redundant Content Servers where the service load is balanced between Content Servers to maximize performance. In a standard Content Server load-balancing scenario, proximity values are used to determine which Content Server processes an item. In a cluster scenario, third-party load-balancers are used.

2.1 Content Server HA configuration

Architecture of Content Server installation is similar to basic installation mode, but Content Servers are running on two server. This setup arranges needed power and effectivity. Figure 2.1 describe a fully redundant HA system. This description include two Content Servers, connection brokers, JMS, full-text indexing system and applications serving the content from repository. In specific case repository can be running on cluster load balancer. The figure illustrates an HA system built on the EMC Documentum platform.

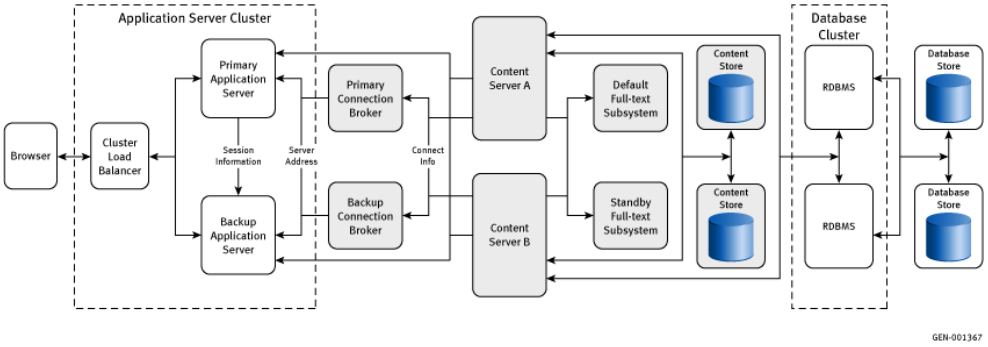


Figure 2.1: Content Server High availability decription. [1]

Having multiple servers dedicated for repository can serve content to application for many users at parallel much faster then single server. HA provides an option to dedicate one server to specific group of users or to a particular application. Configuring CS for load balancing must project on identical proximity values to given connection brokers. In that way, when a DFC client determines which server, it will randomly pick one of the servers. If the values are different, the DFC client will always choose the server with the lowest proximity value. If a Content Server stops and additional servers are running against the repository with proximity values less than 9000, the client library, with a few exceptions, will gracefully reconnect any sessions that were connected to the stopped server to one of those servers. The exceptions are:

1. If the client application is processing a collection when the disconnection occurs, the collection is closed and must be regenerated again when the connection is reestablished

2. If a content transfer is occurring between the client and server, the content transfer must be restarted from the beginning

3. If the client had an open explicit transaction when the disconnection occurred, the transaction was rolled back and must be restarted from the beginning

4. If the additional servers known to a session's connection broker do not have the same proximity value, the client library will choose the next closest server for failover. Sessions cannot failover to a Content Server whose proximity is 9000 or greater. Content Servers with proximity values set 9000 or higher are called remote Content Servers, usually located at remote, distributed sites. [1]

2.2 Configuring Content Server for HA

This section contains an instructions how to prepare and configure High availability of two Content Servers for load balancing. The first Content Server, CS1, is the primary server, and the second Content Server, CS2, is the secondary Content Server.

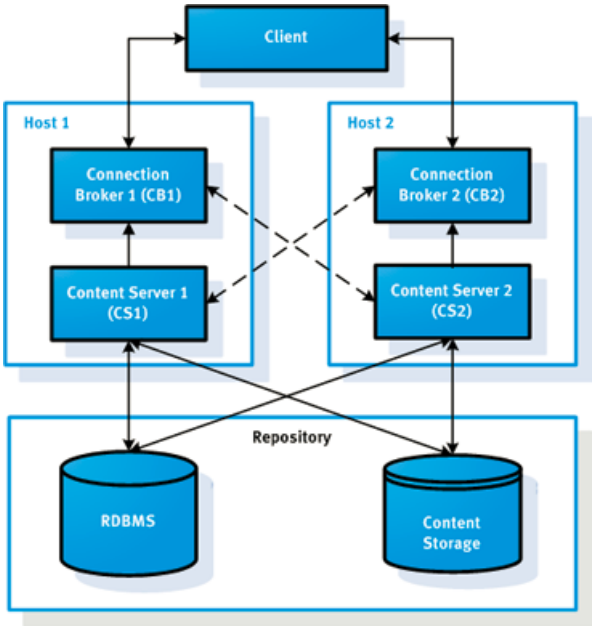


Figure 2.2: HA Configuration of two Content Server. [1]

As a first is necessary to prepare virtual machines where the whole system would be running. I have chosen am using VMWare Cloud solution which is the recommended solution and supported solution. I have dedicated to separate servers as much as possible (more detailed description in section 3.1). The figure 2.2 shows relation between content servers, DB server and a content storage. Remote control of VMs is possible using VMWare application or Microsoft Remote Desktop application using the Remote Desktop Protocol (RDP) also known as Terminal Server Connection. My personal preference is Remote Desktop. Using RDP application RDBMS has been installed as the storage for Documentum metadata. CS1 is installed and configured using normal installer for a creation of a single instance comparing to CS2 where the configuration utility for creating HA was used. The utility is called CFSConfiguration. This program is used to configure the content-file server and the remote Content Server. This utility creates the second server configuration object, and copies the required files such as aek.key, dbpasswd.txt, server.ini, webcache, and so on. To achieve load balancing and failover, you must ensure that the following criteria are met before starting the setup process:

- The Content Servers must share the Content Storage. Content Storage must not be distributed.
- Proximities must be identical.
- Use the CFSConfiguration utility to install additional instances of Content Server.

The process of setting up Content Servers for load balancing and fail-over includes the following tasks:

1. Installing Content Server on CS1.
2. Configuring the repository on CS1.
3. Installing Content Server on CS2.
4. Running the CFSConfiguration utility on CS2.

The CFSConfigurtation Utility installs Content Servers with a proximity value of 9000 or more. Proximity decides the distance between the connection broker and the server. Proximity must be less than 9000 to achieve failover. Proximity values of the connection broker must be equal to achieve load balance. If the servers have identical proximity values, clients pick one of the servers, randomly. If the proximity values are different, clients will always choose the server with the lowest proximity value.
Requirements for load balancing Content Server and configuring fail over on the server are as follows:

- The database client software must be installed on the content-file server hosts.
- The values used on the primary and remote hosts for database connectivity must be identical and must be valid on the remote hosts.
- Content Server and the file store must be in the same domain. The installation user account of Content Server must be available on the domain.
- The installation user account must have full access control to the file store.
- Ensure that the installations of the primary and secondary servers are completed by a domain user account.

2.2.1 Verify failover

- On a client computer, ensure that the dfc.properties file's entries refer to both the connection broker and host's IP.
- On a server computers, ensure the server.ini file's entries refer to both connection broker and proximity.
- Stop CS1. Ensure that DFC client application connects through other CS2 which is running.
- Stop CS2 and start CS1. Ensure that DFC client application connects through running CS1. [1]

2.3 JMS HA configurations

Despite the fact that configuring and testing of JMS high availability hasn't been intended (as it is pretty straightforward and there is not much options to be adjusted) as a part of this thesis, it is worth of mention possibilities which are offered.
JMS can be configured in these three options:

- Content Server and JMS on a single host
- Content Server and JMS on two hosts
- Content Server and JMS on multiple hosts

Description of these options below at each subsections.

2.3.1 Content Server and JMS on a single host

At Figure 2.3 you can see architecture of two Content Servers and their JMS instances running on single host. JMS of CS1 is installed by default with Content Server installation but JMS of CS2 is not installed. Instead, it shares the JMS of the first Content Server. Both CSs are serving one repository.
However this configuration doesn't have good performance and in case failover of host means critical incident.

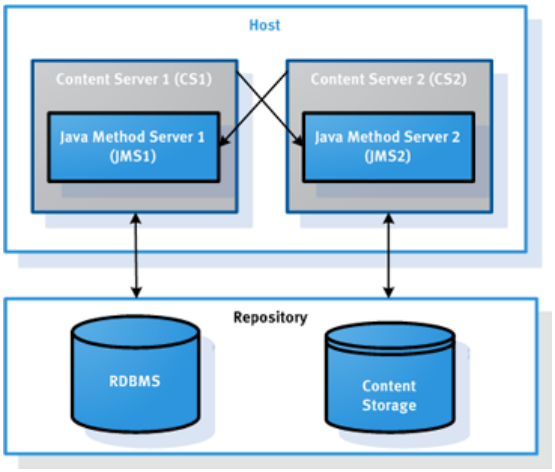


Figure 2.3: Content Server and JMS on a single host. [1]

2.3.2 Content Server and JMS on two hosts

This installation divides each Content Server and one instance of JMS to single host. JMSs are configured by default via installation of Content Server. All Content Server must serve one repository. Case of failover of host is safely secure by double instances of Content Server.

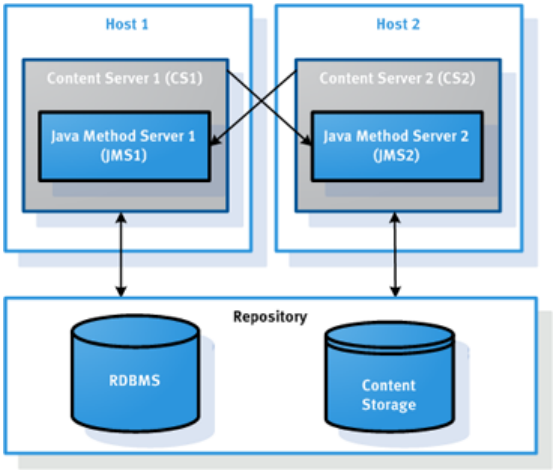


Figure 2.4: Content Server and JMS on two hosts. [1]

2.3.3 Content Server and JMS on multiple hosts

Figure 2.5 depicts multiple Content Servers and multiple instances of JMS set up on multiple machines, serving one repository. In this configuration,

during failover, JMS requests are distributed in a round-robin fashion to the remaining failover nodes.

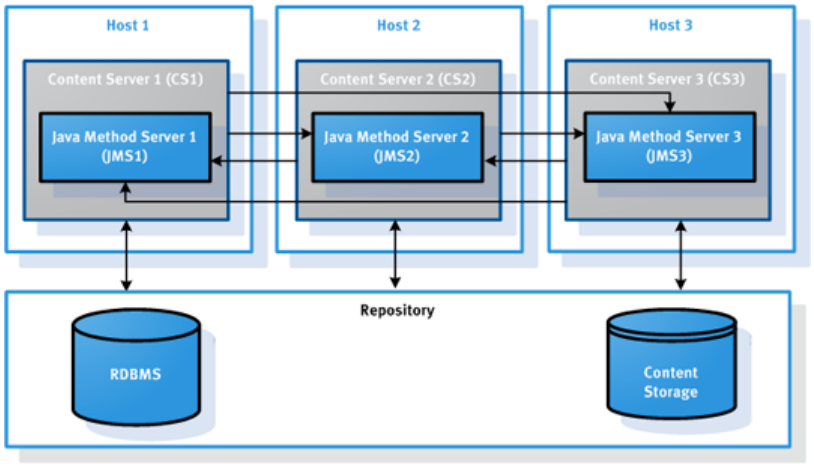


Figure 2.5: Content Server and JMS on multiple hosts. [1]

2.4 Configuring JMS on multiple hosts

Instance of JMS running on primary Content Server is shared on other hosts by default. To achieve JMS HA, a second CS must be associated with JMS. Configuration of primary instance JMS is original setup for other JMS. This configuration can be packed by jmsConfig tool on Content Server and then distributed to other CSs. The following figure 2.6 shows two embedded JMS instances, each connected to its own Content Server on a two hosts supporting one repository to two JMS instances set up for HA with Content Servers on two hosts supporting one repository.

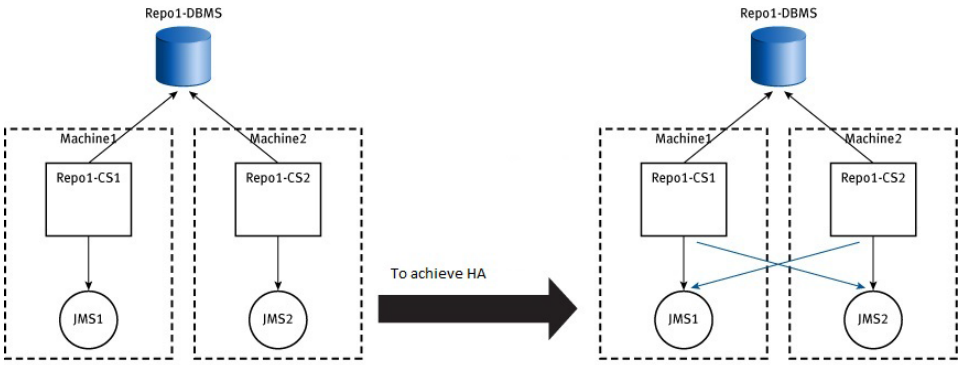


Figure 2.6: JMS instances configured for HA on two hosts. [1]

Chapter 3

Configuring High Availability of the Content Server

In order to be able to verify the load balancing and the fail over possibilities, an environment with current configuration was created. Where Content Server running on Node 1 and Node 2 serve to a repository called "Test" more at section 3.3. As it is visible from the table below some other components were installed as well. Those are required by a content server and some of them were used for testing purposes (described later).

3.1 Test environment description

Table 3.1: Names of used servers

Name of the server	Components installed	Operation System
Node 1	Documentum CS 6.5 SP3 Patch 24	Windows 2008 R2
Node 2	Documentum CS 6.5 SP3 Patch 24	Windows 2008 R2
DB Server	Oracle 11g	Windows 2008 R2
DFC Server	Documentum Foundation Classes 7 Java 1.8 Eclipse (Mars)	Windows 2008 R2
JMeter	Apache JMeter 2.11 Google JMeter plugins Java SDK Tomcat 7	Windows 2008 R2
DocWeb	Documentum Web Services Documentum Administrator	Windows 2008 R2

3.2 Configuration of Content Server nodes for HA

In order to configure for load balancing/fail over, content servers must project identical proximity values to any connection broker which means that server.ini file on both content servers has to be updated (this configuration can be extended in dm_server_config object which overrides server.ini, but in this case only server.ini was updated). The server installation procedure created both files automatically and the files are called when the server is started. The server.ini file contains information provided during the installation process, including the repository name and the repository ID. That information allows the server to access the repository and contact the RDBMS server.

In order to configure load balancing/fail over on the content server level only a section in server.ini called DOCBROKER_PROJECTION_TARGET was updated as can be seen below (it defines the connection brokers to which the server sends its connection information).

Node 1

```
[DOCBROKER_PROJECTION_TARGET]
host = Node1
port = 1489
proximity = 1
[DOCBROKER_PROJECTION_TARGET_1]
host = Node2
port = 1489
proximity = 1
```

Node 2

```
[DOCBROKER_PROJECTION_TARGET]
host = Node2
port = 1489
proximity = 1
[DOCBROKER_PROJECTION_TARGET_1]
host = Node1
port = 1489
proximity = 1
```

As can be seen above, servers' proximity value is defined as 1 for all connection broker projection targets. The proximity value represents the server's physical proximity to the connection broker. When clients receive server information from a connection broker, by default they choose to connect to the server with the smallest proximity value (representing the closest available server). If two or more servers have the same lowest value (both servers Node1 and Node 2 have a value of 1) then the client makes by default a random choice between the servers (or choice defined on other values which are defined later).

3.3 Verifying of HA configuration for Node1

In order to prove that high availability configuration is working a utility called dmqdocbroker was launched to get list of servers visible for the repository. The utility is located on both content servers.

The basic syntax of the command looks like:

```
dmqdocbroker -t <server_name> -p 1489 -c getservermap <name_of_repository>
```

Executed command:

```
dmqdocbroker -t Node1 -p 1489 -c getservermap Test
```

Printed output:

```
dmqdocbroker: A DocBroker Query Tool
dmqdocbroker: Documentum Client Library Version:
    6.5.0.323SP3P0900
Using specified port: 1489
************************************************
**     D O C B R O K E R     I N F O            **
************************************************
Docbroker host            : Node1
Docbroker port            : 1490
Docbroker network address : INET_ADDR: 02 5d2 0a0b22d6
                            HLUW1575S 10.11.34.214
Docbroker version         : 6.5.0.355 SP3P0900 Win32
************************************************
**           S E R V E R     M A P              **
************************************************
Docbase Test has 2 servers:
------------------------------------------------
  server name          :  Node2_Test
  server host          :  Node2
  server status        :  Open
  client proximity     :  1
  server version       :  6.5.0.355 SP3P2400 Win32.Oracle
  server process id    :  3660
  last ckpt time       :  12/21/2016 2:12:26 PM
  next ckpt time       :  12/21/2016 2:17:26 PM
  connect protocol     :  TCP_RPC
  connection addr      :  INET_ADDR: 02 2715 0a0b22d7
                          Node2 10.11.34.215
  keep entry interval  :  1440
  docbase id           :  12345
------------------------------------------------
```

```
server name          :  Test
server host          :  Node1
server status        :  Open
client proximity     :  1
server version       :  6.5.0.355 SP3P2400 Win32.Oracle
server process id    :  4296
last ckpt time       :  12/21/2016 2:12:22 PM
next ckpt time       :  12/21/2016 2:17:22 PM
connect protocol     :  TCP_RPC
connection addr      :  INET_ADDR: 02 ba0d 0a0b22d6
                        Node1 10.11.34.214
keep entry interval  :  1440
docbase id           :  12345
```

As it is visible from the output of the command above, the docbroker contains a two pieces of information

- both nodes are visible
- proximities are identical which is the required behavior.

3.4 Verifying of HA configuration for Node2

Executed command:

```
dmqdocbroker -t Node2 -p 1489 -c getservermap Test
```

Printed output:

```
dmqdocbroker: A DocBroker Query Tool
dmqdocbroker: Documentum Client Library Version:
    6.5.0.323SP3P0900
Using specified port: 1489
************************************************
**      D O C B R O K E R     I N F O           **
************************************************
Docbroker host             : Node2
Docbroker port             : 1490
Docbroker network address  : INET_ADDR: 02 5d2 0a0b22d7
                             Node2 10.11.34.215
Docbroker version          : 6.5.0.355 SP3P0900 Win32
************************************************
**           S E R V E R     M A P              **
************************************************
Docbase Test has 2 servers:
```

```
-------------------------------------------------------------
  server name          :  Node2_Test
  server host          :  Node2
  server status        :  Open
  client proximity     :  1
  server version       :  6.5.0.355 SP3P2400 Win32.Oracle
  server process id    :  3660
  last ckpt time       :  12/21/2016 2:22:26 PM
  next ckpt time       :  12/21/2016 2:27:26 PM
  connect protocol     :  TCP_RPC
  connection addr      :  INET_ADDR: 02 2715 0a0b22d7
                          Node2 10.11.34.215
  keep entry interval  :  1440
  docbase id           :  12345
-------------------------------------------------------------
  server name          :  Test
  server host          :  Node1
  server status        :  Open
  client proximity     :  1
  server version       :  6.5.0.355 SP3P2400 Win32.Oracle
  server process id    :  4296
  last ckpt time       :  12/21/2016 2:22:22 PM
  next ckpt time       :  12/21/2016 2:27:22 PM
  connect protocol     :  TCP_RPC
  connection addr      :  INET_ADDR: 02 ba0d 0a0b22d6
                          Node1 10.11.34.214
  keep entry interval  :  1440
  docbase id           :  12345
-------------------------------------------------------------
```

As it is visible from the output of the command above, the docbroker contains a two pieces of information

- both nodes are visible
- proximities are identical which is the required behavior.

It means that content servers were successfully adjusted in HA mode.

Chapter 4

Testing High Availability of the Content Server

Usually the typical requests coming from customers side is to have good session management meaning that sessions should be distributed evenly between the content servers to fully utilize a potential of high availability configuration. Session management coming from external application can be configured on DFC layer which provides an interface for communication with Content servers. In this chapter I will try to describe and verify that load balancing can be affected by changing settings on DFC layer as well as I demonstrate a configuration and basic functionality of the fail over.

4.1 Load balancing test

Load balancing is a methodology used to distribute workload evenly across two or more computers. Documentum Content Server load balancing means that subsequent client requests are distributed to two or more Content Servers to distribute the load.

Most of the configuration parameters that configure how repository sessions are handled by DFC are defined in the dfc.properties file. In order to test and note results for particular settings on the DFC level, a simple DFC application was written in Java.

```
package haTesting;

import com.documentum.com.DfClientX;
import com.documentum.com.IDfClientX;
import com.documentum.fc.client.IDfClient;
import com.documentum.fc.client.IDfSession;
import com.documentum.fc.client.IDfSessionManager;
```

```
import com.documentum.fc.common.IDfLoginInfo;
import java.text.DateFormat;
import java.text.SimpleDateFormat;
import java.util.Date;

public class loginTest {

public static void main(String[] args) throws Exception{

        DateFormat dateFormat = new SimpleDateFormat
        ("yyyy/MM/dd HH:mm:ss");
        Date date = new Date();
        System.out.println("**************TEST START "
        + dateFormat.format(date));
        long time = System.currentTimeMillis();
        IDfClientX clientx = new DfClientX();
        IDfClient client = clientx.getLocalClient();
        time = System.currentTimeMillis() - time;
        long timestore = time;
        time = System.currentTimeMillis();
        IDfLoginInfo loginInfo = clientx.getLoginInfo();
        loginInfo.setUser("dmadmin");
        loginInfo.setPassword("Test123");
        IDfSessionManager sMgr = client.newSessionManager();
        sMgr.setIdentity("Test", loginInfo);
        time = System.currentTimeMillis() - time;
        timestore += time;
        time = System.currentTimeMillis();
        IDfSession session = sMgr.getSession("Test");
        if (session == null) {
                    System.out.println(" FAILED
                    CONNECT TO THE REPOSITORY");
        } else {
                    System.out.println(" CONNECTED
                    TO REPOSITORY");
        }
        System.out.println("CONNECTED TO: " +
        session.getServerConfig().getValue("object_name"));
        time = System.currentTimeMillis() - time;
        timestore += time;
        System.out.println(" The overall time needed to
        login:" + timestore + " milliseconds");
        date = new Date();
        System.out.println("**************TEST END "
```

```
        + dateFormat.format(date));
        }
}
```

The utility measure a time required for getting a session and returns object_name of a dm_server_config object which uniquely defines the server on which the current session landed.
The utility is repeatedly called from a batch file and the output is redirected to a text file.

```
@echo off
:loop
java -classpath ".;Shared\dfc.jar;hatesting" \\
haTesting.loginTest >> FAILOVER.txt
if %ERRORLEVEL% == 0 goto loop
ping 1.1.1.1 -n 1 -w 2000 > nul
:end
```

As it has been already mentioned, handling of sessions coming to the content servers is configured on any server which contains DFC libraries (every application or DFC code which want to create a connection to a content server has to use DFC). In order to adjust some DFC behaviours a modification of dfc.properties file has to be done. The file enables to set preferences for how DFC handles certain choices in the course of its execution.

I created a minimal configuration of dfc.properties used by the application which contains all entries that are required for further testing.

```
dfc.docbroker.host[0]=Node1
dfc.docbroker.port[0]=1489
dfc.docbroker.host[1]=Node2
dfc.docbroker.port[1]=1489
dfc.docbroker.search_order=sequential
dfc.session.load_balance_strategy=sequential
dfc.docbroker.timeout = 10
dfc.globalregistry.repository=Test
dfc.globalregistry.username=dm_bof_registry
dfc.globalregistry.password=NrGNKhLDrkoASDZE0RGJiu==
```

dfc.docbroker.host

Defines a docbroker to which the application is connecting.

dfc.docbroker.port

Defines a port on which the docbrokers are running.

dfc.dobroker.timeout

The amount of time in seconds to wait for a response from a connection broker. If set to 0 then DFC will not impose a time limit of its own and will rely upon the standard TCP/IP timeout. (prototype value: 0 min value: 0, max value: 60).

dfc.session.load_balance_strategy

Determines the algorithm to use when selecting a server when multiple server choices exist with the same proximity. A list of servers is obtained from the available docbrokers and the load balance strategy determines how sessions are distributed among those servers. Valid values are random, sequential, balanced.

dfc.docbroker.search_order

Determines whether the connection request is sent first to the primary connection broker or to a randomly selected connection broker (that has been configured). Valid values are sequential, random, balanced.

Random chooses a randomly selected server. There is no history of requests, so one server could get more requests than others, in the short term.

Sequential uses an ordered list of available servers and distributes sessions to the servers in that order. The initial order of the list is random. When the end of the list is reached, the algorithm starts over at the beginning of the list.

Balanced uses an ordered list of available servers and distributes sessions to the servers in that order. The initial order of the list is random. Each time the end of the list is reached, the list is reordered randomly.

Using the utility described in chapter 3. A 300 sessions was created for each possible configuration to see how the sessions were distributed between the two nodes.

Table 4.1: Count of sessions created on particular server

	Random	sequential	balanced
Node1	168	155	138
Node 2	132	145	162

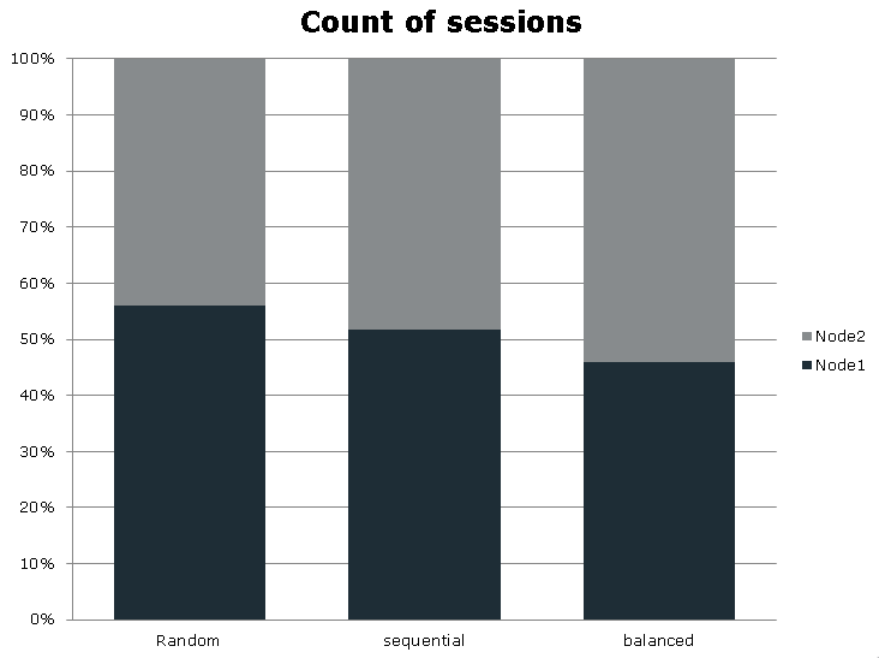


Figure 4.1: Count of sessions created on particular server

Based on the results from the testing it seems that the best option for distributing sessions the best way is "sequential" where difference is only around 6.5%. On the other hand the "Random" option seems like the option which distributes sessions not so reliably as the other two options.

4.2 Fail over test

Failover testing validates a system's ability to be able to allocate extra resource and to move operations to back-up systems. Testing failover is used to verify an IT system's ability to continue operations while the processing capability is being transferred to a back-up system. It determines whether a system is able to allocate extra resource such as another host server during critical failures or at the point the system reaches a predetermined performance threshold. [1]
In a failover scenario, if one Content Server has failed, all the client requests are switched to the second Content Server working in parallel with the first Content Server. In the current setup we will perform load balance and failover, simultaneously. [2]
In order to verify that fail over is working as expected the behavior of the system was observed in a way that TEST repository shut down on either content server and activated again. The expected y is that all sessions should be

[1] http://www.testingperformance.org/definitions/what-is-failover-testing

transferred on the node which remained running.

This chapter is to summarize the results from the fail over testing of the Documentum Web Services (further referred to as DocWeb services) and Accelerator Content Service (further referred to as ACS) applications in the environment defined in table 3.1. In order to be able to simulate real load a test where a 100-user load test sending 24000 service requests every hour was prepared to observe the behavior of the system when the TEST repository and then ACS was shut down on either content server and activated again.

Table 4.2: DocWeb services deployed on the Tomcat server and their distribution over time.

Service name	Distribution [%]
query	30%
getPropertiesOfLatestVersion	3%
getProperties	18%
updateProperties	7%
downloadDocument (getContentUrl + ACS download)	20%
createDocument	17%
checkOutLatestVersion	4%
checkIn	1%

The tool which was chosen to execute the load test was Apache JMeter 2.11 which is free and open source tool built on Java platform. Test scripts used by the JMeter were prepared in a way that each JMeter script represented one of the DocWeb services. Testing documents were uploaded to the TEST repository under the subfolders created in /Temp/DocWeb using the Documentum Administrator application (average size around 200kB). A 100 test users were created as well.

The monitoring of the DocWeb system was set up only for the content servers Node1 and Node2. Since the content servers run on Windows Server 2008 R2, Perfmon was used to collect utilisation data for resources such as CPU, memory, disk and network interface. The Perfmon utility was run on the DFC server.

4.2.1 Test scenario verification

To verify that the scenario: *"A 100-user load test sending 24000 service requests every hour."*, works as expected a small performance test was done.

Scenario name:	load_100users_24000rph
Number of virtual users:	100
Maximum generated load:	24000 service requests per hour

The load in the test was distributed among the DocWeb services as specified in the following table:

Table 4.3: Load distribution among the DocWeb

Request name	Total hourly usage (req/h)	Users
query	7200	30
getPropertiesofLatestVersion	720	2
getProperties	4320	18
updateProperties	1680	7
downloadDocument (getContentURL + downloadContent)	4800	20
createDocument	4080	17
checkOutLatestVersion	960	4
checkIn	240	2

The table below provides some basic statistics that were calculated from the DocWeb service response times during the load test. The table rows in red highlight the query service processing the most complex query in the list and the top 3 most time consuming services in terms of their average response times.

Table 4.4: Test of 100 virtual users - 24000 DocWeb service requests/hour.

Service Name	#Samples	Avg[ms]	Median[ms]	Min[ms]	Max[ms]	Error Rate[%]	Throughput[req/s]	Downloaded Data[kB/s]
createDocument	4167	390	300	180	6117	0.00%	1.10	1.10
downloadContent	4881	71	66	59	295	0.00%	1.30	133.60
getContentURL	4901	163	116	71	3177	0.00%	1.30	2.40
getProperties	4412	109	74	46	3281	0.00%	1.20	16.00
getPropertiesOfLatestVersion	735	302	261	1	856	0.27%	0.20	3.20
checkIn	246	525	432	238	1103	0.00%	0.07	0.07
checkOutLatestVersion	981	315	309	131	3546	0.00%	0.30	0.50
query_H	816	21909	21197	14166	48146	0.00%	0.20	32.20
query_L	6531	350	307	50	3309	0.00%	1.80	163.60
updateProperties	1716	170	115	71	1314	0.00%	0.50	0.50

As can be seen from the results the most time consuming DocWeb Service was *query_H* which was processing the most complex query:

```
select r_object_id, object_name, owner_name, acl_name
from dm_sysobject, dmr_content where folder('/Temp', DESCEND)
and any parent_id = r_object_id and full_content_size >'1024'
and full_content_size < '8192';
```

A few examples of quries which were marked as query_L:

```
select object_name, owner_name, r_creation_date
from dm_sysobject where acl_name = 'test_acl';
select r_object_id, object_name, r_creation_date
from dm_document where owner_name='Stepan Staniek';
```

- Excluding the heavy query, the top 3 most-time consuming DocWeb services (in terms of averages) were:
  - checkIn - 525 ms
  - createDocumentcreateDocument - 390 ms
  - query_L - 350 ms
- In total, 2 failures occurred; both of them occurred in getPropertiesOfLatestVersion. The detailed logging level was not enabled to capture the error messages though.
- 351 kB downloaded every second using DocWeb Services.

As can be seen above, the user load scenario is working as expected and it simulates the real "work load" for the Documentum Content servers. In order to verify the fail over functionalities the scenario above was launched once again and following actions were triggered on the Content Servers (Node1 * Node2).

- Action #1: 13:45:43; TEST repository on Node1 was shut down
- Action #2: 13:57:37; TEST repository on Node1 was activated
- Action #3: 14:08:00; ACS on Node1 was shut down
- Action #4: 14:21:37 - 14:22:04; ACS on Node1 activated

The ACS instance on content server Node1 was shut down or activated by shutting down or restarting the Java Method Server on the same content server.

4.2.2 Observations

- When the TEST repository was shut down on content server Node1, the first document download (downloadContent) failure occurred 23 seconds later as a result of the URL returned from the getContentUrl service pointing to Node1. From then onwards, the same failure occurred in the next 7 minutes. After that, the hostname returned from the getContentUrl service was only Node2 and therefore the failure no longer appeared.
- During the 7-minute interval 180 downloads failed and 200 downloads succeeded.
- It seems that the traffic transferred from Node1 to Node2 after the repository shutdown was not fully transferred back when the repository was activated again. However, some of the load was diverted back to Node1, reaching the maximum in about 7-8 minutes from the repository re-activation.
- Shutting down the ACS instance on content server Node1 caused downloadContent (document download) requests to fail immediately.
  The reason for the failure was the same as in the case of shutting down the TEST repository. Unlike in the case of TEST shutdown, document download failures during the ACS shutdown occurred longer - for about 9 minutes before download requests were diverted only to Node2.
- During the 9-minute interval 355 download requests failed and 380 requests succeeded.
- After the ACS shutdown on Node1, the load was fully transferred to Node2 in 9 minutes. This corresponds to the fact that download requests were sent only to Node2 in 9 minutes from shutting down the ACS instance.
- When ACS was re-activated, the load was transferred back to Node1.

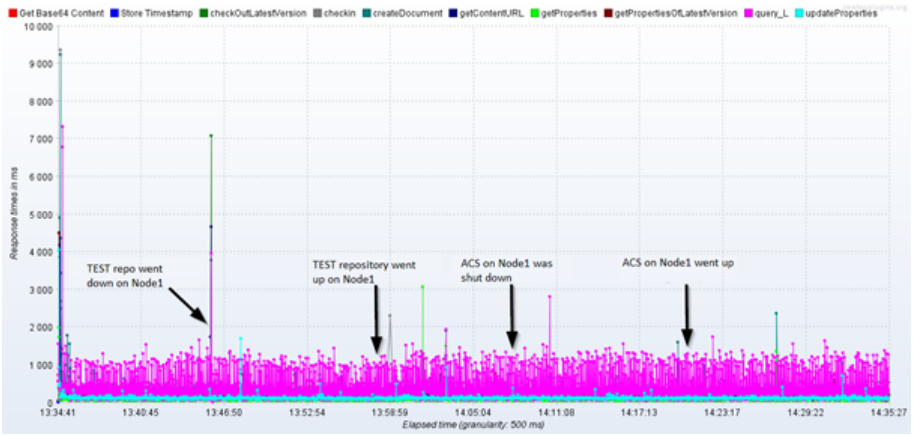


Figure 4.2: JMeter output of requests by all services include. Timezone GMT+0.

The course of DocWeb service response times during the 100-user load test sending 24000 requests/h (the heavy query and downloadContent excluded).

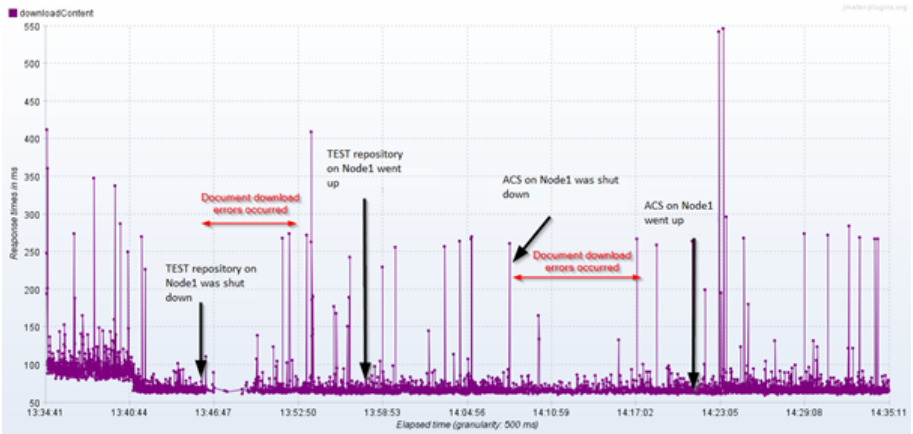


Figure 4.3: JMeter output of requests by DownloadContent service. Timezone GMT+0.

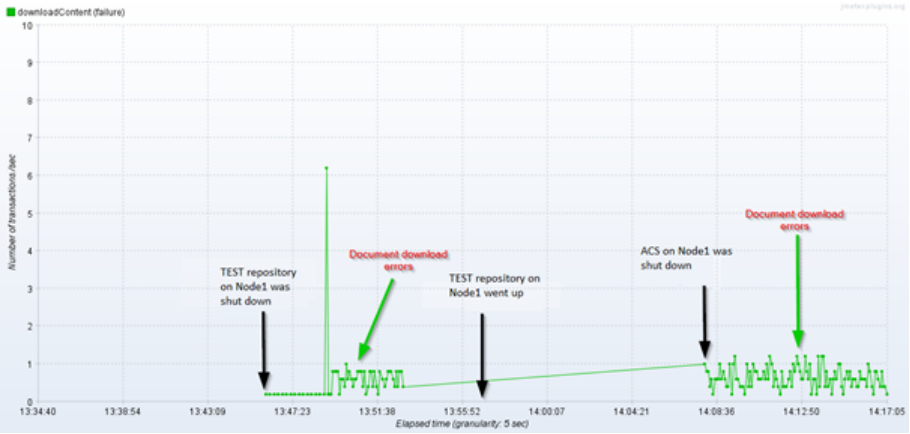


Figure 4.4: JMeter output of failure requests by DownloadContent service. Timezone GMT+0.

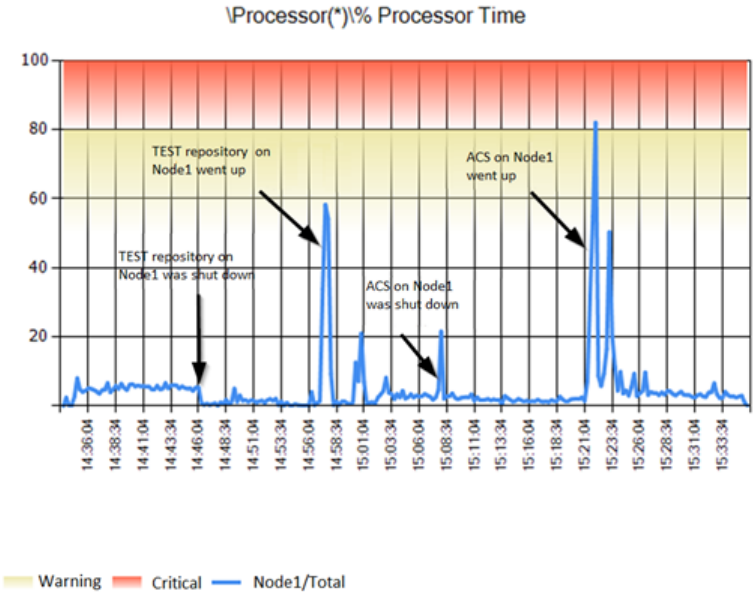


Figure 4.5: CPU usage on Node1 during the 100-user failover test sending 24000 requests/h. Timezone GMT+1.

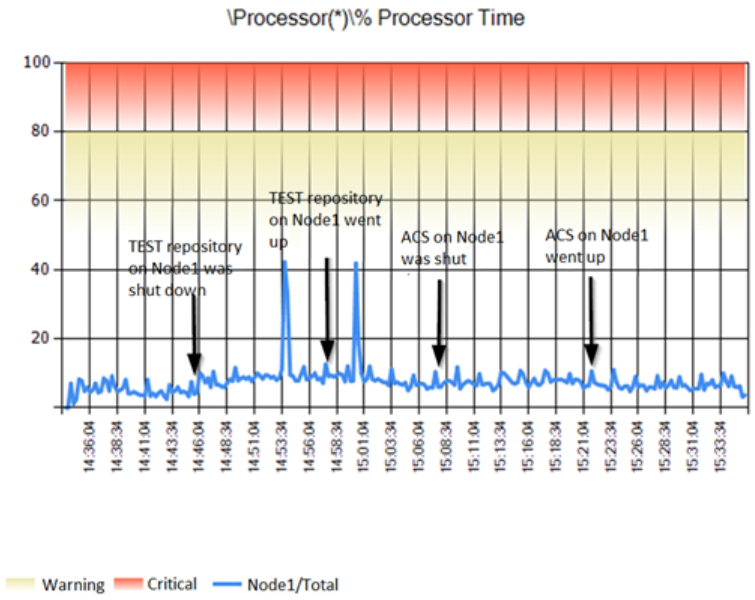


Figure 4.6: CPU usage on Node2 during the 100-user failover test sending 24000 requests/h. Timezone GMT+1.

- The failover test showed that after shutting down the TEST repository on content server Node1 one Server Communication Failure occurred while processing the complex query and it took about 7 minutes before the getContentUrl service of the DocWeb application returned URLs in which the hostname pointed to content server Node2 only. As a result, download failures kept occurring during the 7-minute time interval after any attempt to download a document from Node1. The ratio of document download failures to document download successes was 180 to 200.
- Reactivating the TEST repository on Node1 did not lead to transferring some of the load from Node2 to Node1 immediately. It took up to 8 minutes before Node1 received some of the load back (see Figure 4.5 and Figure 4.6).
- Similarly as during the TEST shutdown, turning off the ACS instance on content server Node1 caused ACS document download requests to fail for about 9 minutes before the getContentUrl service of the DocWeb application returned URLs in which the hostname pointed to content server Node2 only. The ratio of download failures to download successes was 355 to 380.
- Regular spikes about 30 minutes apart observed in CPU usage on the content servers during the load tests.

Conclusion

The goal of my thesis was to describe, implement, verify and demonstrate possibilities of High Availability configuration which a product Documentum Content Server offers. In the theoretical part I described briefly very complex Documentum architecture in a way to get at least a basic picture of how the Documentum Content is composed to be able to understand further Documentum High Availability configuration as well as I described High Availability configuration itself, its benefits and ways of how it can be configured. More technical details regarding to configuring additional content server to enable HA was provided in chapter 2.2. Even though JMS HA configuration wasn't tested as the behavior of it is pretty straightforward some basic information were provided as well in chapter 2.3 and 2.4.

In the practical part I installed and prepared a Documentum environment described in chapter 3, table 3.1. The installation itself is not described in the thesis as all the details are stated in attached installation guide, moreover, I didn't see any extra errors on which I would spend much time to fix them. The important part for the testing was to adjust the content server in high availability mode which I described in chapter 3.2 and confirmed in sections 3.3 and 3.4.

The practical part is consisted from two parts. In the first one I tested a behavior of load balancing based on the configuration of external application which I created in Java. The simple java code returns a server name on which a session created by the code landed. From the results is visible that all three configurations I tested are working reliably and I did not notice any issues. The results I got are based on 300 requests I sent to the content servers and it seems that the configuration which distributes sessions in the best way is sequential. Of course that we have to take into the account that this testing was done on CS running on version 6.5SP3 so it is not applicable in general but only for this specific version which was released in July 2008 (SP3 sometime later). Despite the fact that this version is quite old, it is still heavily used by many companies.

In the second part of the practical part I tested fail over functionality and summarized the results. In order to be able to simulate real load I created a test where I simulated a 100-user load test sending 24000 service requests every hour (using JMeter). At first I left the test running successfully to get a picture of how the system is behaving and stated results in the table 4.2 which helped me to understand how the system is behaving and I could see the performance. Then I re-ran the test and called shut down for the TEST repository and after some time it again and then I ran the same scenario for the ACS server.

After the TEST repository was shut down on content server Node1, the first document download failure occurred 23 seconds later as a result of the URL returned from the getContentUrl service pointing to Node1. From then onwards, the same failure occurred in the next 7 minutes so in fact it took 7 minutes to get all sessions fully transferred on the running node which caused that 180 download requests failed and 200 succeeded. When the repository on Node1 was started back, it took almost 8 minutes to fully divert the load back to Node1. After ACS server was shut down on Node 1, it caused that downloadContent (document download) requests to fail immediately. Unlike in the case of TEST shutdown, document download failures during the ACS shutdown occurred longer – for about 9 minutes before download requests were diverted only to Node2. During the 9-minute interval 355 download requests failed and 380 re- quests succeeded. When ACS was re-activated, the load was transferred back to Node1. Based on the results I can say that the fail-over is working but the reaction times are not perfect. It seems that the default configuration to be used is not sufficient and it would be worth to investigation the issue further. Despite the fact that this version is still being used by many companies, it is already out of support so there is no way of how to contact vendor to investigate the issue further. It would be worth to do the same testing for the newest version 7.2 to see of the fail over performance in a way of recovering from crash was improved.

Bibliography

[1] Corporation, E.: *EMC®Documentum®Content Server OEM Edition 6.5 SP3 - Installation Guide*. EMC®Corporation, 2008.

[2] Corporation, E.: *EMC®Documentum®Content Server - Load balance and failover*. EMC®Corporation, 2012.

[3] Documentation - ShareLaTeX, Online LaTeX editor: ShareLaTeX Documentation. `https://cs.sharelatex.com/learn/`, 2013, online; accessed 13 January 2016.

[4] Kumar, P.: *Documentum Content Management Foundations: EMC Proven Professional Certification Exam E20-120 Study Guide*. From Technologies to Solutions, Packt Pub., 2007, ISBN 9781847192417. Dostupné z: `https://books.google.cz/books?id=asBywd7hq2kC`

[5] Lisová, M.: Implementation of the ECM system in a small company with a focus on technical documentation. 2014. Dostupné z: `https://insis.vse.cz/zp/portal_zp.pl?podrobnosti_zp=45438`

[6] Corporation, E.: *EMC®Documentum®Content Server 7.2 Administration and Configuration Guide*. EMC®Corporation, 2016.

[7] Corporation, E.: *EMC®Documentum®Content Server 6.5 Distributed Configuration Guide*. EMC®Corporation, 2008.

[8] Corporation, E.: *EMC®Documentum®Foundation Classes 7.2 Environment and System Requirements Guide*. EMC®Corporation, 2016.

[9] Corporation, E.: *EMC®Documentum®Foundation Classes 7.2 Installation Guide*. EMC®Corporation, 2016.

[10] Corporation, E.: *EMC®Documentum®Foundation Classes 7.2 Development Guide*. EMC®Corporation, 2016.

[11] Roth, M.: *A Beginner's Guide to Developing Documentum Desktop Applications: Techniques and Solutions Using Visual Basic and the DFC*. iUniverse, 2005, ISBN 9780595339686. Dostupné z: `https://books.google.cz/books?id=sP-v6YirZ7EC`

[12] Zhu, W.; Bacalzo, R.; Edeen, E.; aj.: *Federated Content Management: Accessing Content from Disparate Repositories with IBM Content Federation Services and IBM Content Integrator*. IBM redbooks, IBM Redbooks, 2010, ISBN 9780738433974. Dostupné z: `https://books.google.cz/books?id=XVPAAgAAQBAJ`

[13] Corporation, E.: *EMC®Documentum®Content Server 7.2 - Installation Guide*. EMC®Corporation, 2016.

[14] Corporation, E.: *EMC®Documentum®Content Server 7.2 Distributed Configuration Guide*. EMC®Corporation, 2016.

Appendix A

Acronyms

ECM Enterprise Content Management

DMS Document management system

DFC Documentum Foundation Classes

RDBMS Relational database management system

JMS Relational database management system

ACS Accelerated ContentServer

RCS remote Content Server

CS Content Server

CFS CFSConfiguration

WDK Documentum Web Development Kit

Appendix B

Contents of enclosed CD

- `bibliography` ..... Used materials in thesis
- `readme.txt` ..... the file with CD contents description
- `src` ..... the directory of source codes
- `thesis` ..... the directory of LaTeX source codes of the thesis
  - `thesis.pdf` ..... the thesis text in PDF format
  - `thesis.images` ..... the thesis images
  - `thesis.latex` ..... the zip file of LaTeX source code of the thesis