INFORMATION SYSTEMS IN MANAGEMENT

Information Systems in Management (2016) Vol. 5 (4) 463−472

OVERVIEW AND IMPLEMENTING SQL SERVER HIGH AVAILABILITY SOLUTIONS

MICHAŁ BLEJA

Faculty of Mathematics and Computer Science, University of Lodz

The paper presents the concepts related to the design of high availability solutions for Microsoft SQL (MS SQL) Server database servers. MS SQL Server from version 2012 offers a new capability called *AlwaysOn* which is aimed at reducing downtime of servers or databases during a failure. It includes several mechanisms such as Failover Cluster Instances, Availability Groups, Database mirroring, Log shipping. The paper presents the implementation of some of these concepts in our prototype. We also compare these mechanisms focusing on their aptitude in contemporary information systems.

Keywords: high availability, windows clusters, availability group, failover cluster instance, database mirroring, log shipping

1. Introduction

High availability is implemented in most contemporary database management systems such as Microsoft SQL Server [1, 2], MySQL [3], Oracle [4], PostgreSQL [5]. Database administrators use various mechanisms to increase fault tolerance, especially for mission critical database servers. High availability can be implemented on different levels including the hardware level and the software level. One of the most important hardware-based technique is called Redundant Array of Independent Disks (RAID) [2, 6]. It is aimed at distributing data over multiple disks which guarantees automatic data recovery after the occurrence of hard disk crashes. RAID provides several techniques such as striping (RAID 0),

mirroring (RAID 1), stripping with parity (RAID 5), etc. It can be also implemented on the software side.

The research presented in this paper concerns concepts related to implementing high availability by using MS SQL Server AlwaysOn capabilities. We deal with mechanisms such as AlwaysOn Failover Cluster Instances, AlwaysOn Availability Groups, Database Mirroring, Log Shipping. The first approach provides redundancy at the database server level. A single copy of MS SQL Server is installed on several nodes which forms a Windows Server Failover Clustering (WSFC) cluster. Such the copy is called a failover cluster instance (FCI). The last three approaches act on the database level. They allow to maintain one or more standby databases for a production database. The above mechanism allow database administrators (DBAs) to cope with main recovery objectives: recovery time objective (RTO) and recovery point objective (RPO). The first one determines how long the system can be down and should be minimized. The second one refers to the point in time in the past to which data should be recovered in the case of failure. Implementing high availability is crucial to meet enterprise expectations because they often require no data loss and no downtime when the system crashes.

2. Database level high availability

There are three commonly applied mechanisms for ensuring high availability on the database level: Log shipping, Database mirroring, AlwaysOn Availability Groups. They provide software based failover solution by replicating data between database servers. It is also common practice to deploy not only mechanism for a given database. In particular, log shipping and database mirroring can act on the same database to provide high availability and disaster recovery.

2.1. Log Shipping

In SQL Server versions prior to 2012, DBAs utilized log shipping and database mirroring to maintain standby databases on remote servers for their primary databases. In contrast to Always on Availability Groups, these two approaches do not offer readable secondary databases which would be very close to their primary counterparts. Besides Availability Groups support multiple secondary servers to which data can be replicated in both asynchronous and synchronous manner. Deploying log shipping involves the following steps [7, 8]:

- Switch the primary database to the full recovery model and perform its full backup.
- Restore the full backup of the primary database on the secondary server.

- Execute the procedure *sp_add_log_shipping_primary_database* on the primary server. The procedure returns the job identifier which is responsible for backup of the log file. DBA must schedule this job according to the recovery point objective.
- Run *sp_add_log_shipping_secondary_database* on the secondary server with parameters describing the primary server. The procedure returns two job identifiers. The first job will copy the log backups of the primary server to the second server. The second job will regularly restore these log backups.
- Run *sp_add_log_shipping_primary_secondary_database* on the primary server providing information about the secondary server.

The secondary server can act as a report server if the log backups are restored with the *standby* option on the secondary one. However such report server is available only during the intervals between restore logs. Besides subsequent log backups cannot be restored if there are active connections to the secondary server. It can lead to significant differences between the primary and the secondary database. In consequence during a failover more transaction log backups must be applied to the secondary server before it becomes the primary server.

2.2. Database mirroring

Database mirroring [7, 9] is another mechanism for increasing database availability. It assumes maintaining two copies of a database which are hosted by various SQL Server instances. One instance called *primary server* stores a database available for users. The other instance acts as the mirror server and accepts all the transactions previously applied to the primary server. Database mirroring can be implemented in two various modes: synchronous mode and asynchronous mode. The first one called also high-safety mode assumes that transactions are committed on the primary server once the mirror server receives and commits them. In the second mode clients who applies transactions to the primary server do not wait for confirmation from the mirror server. Clients receive acknowledgement only from the primary server as soon as it sends transaction log records to the mirror server. For that reason this mode is called high performance mode. It can result in gap between instances especially when the primary server workload is high.

Establishing database mirroring sessions between two SQL Server instances involves the following steps [7, 9]:

- Restore the proper backups (the most recent full and differential backups, unbroken sequence of log backups) of the primary database on the mirror server.
- Create an endpoint on each server instance if it does not exist.
- Configure security on each server instance (Windows based authentication or certificate based authentication).
- Configure the primary instance and the mirror instance as partners.

```
--Restore the most recent full backup of demoDB on MSSQL02
restore database demoDB
  from disk='\\backup_server\demoDB_full_backup.bak' with norecovery
--Restore the most recent differential backup of demoDB on MSSQL02
restore database demoDB
  from disk='\\backup_server\demoDB_diff_backup.bak' with norecovery
--Restore the log backups taken after the most recent full backup or
--differential backup
restore database demoDB
 from disk='\\backup_server\demoDB_log_backup_1.bak' with norecovery
...
restore database demoDB
 from disk='\\backup_server\demoDB_log_backup_n.bak' with norecovery
--Implement security on MSSQL01 (Run in the master database)
create master key encryption by password = fhfd#kk!JkL'
create certificate MSSQL01_cert
  with subject = 'MSSQL01_ certificate'
--Create an endpoint on MSSQL01
create endpoint mirroring state=started as tcp (listenr_port=51033)
  for database_mirroring (authentication = certificate MSSQL01_cert
    encryption=disabled, role=all)
backup certificate MSSQL01_cert to file='c:\cert\MSSQL01_cert.cer';
--Copy the file MSSQL01_cert.cer from MSSQL01 to MSSQL02
--Implement security on MSSQL02 (Run in the master database)
create master key encryption by password = 'jyr#kk!Opl'
create certificate MSSQL02_cert
  with subject = 'MSSQL02_ certificate'
--Create an endpoint on MSSQL02
create endpoint mirroring state=started as tcp (listenr_port=51033)
  for database_mirroring (authentication = certificate MSSQL02_cert
    encryption=disabled, role=all)
backup certificate MSSQL02_cert to file=c:\cert\MSSQL02_cert.cer';
--Copy the file MSSQL02_cert.cer from MSSQL02 to MSSQL01
--Configure MSSQL01 for inbound connections (Run in master)
create login MSSQL02_login with password = fgfsGv#!h';
create user MSSQL02_user from login MSSQL02_login;
create certificate MSSQL02_cert authorization MSSQL02_user
  from file = 'c:\cert\MSQL02_cert.cer';
grant connect on endpoint::mirroring to MSSQL02_user;
--Configure MSSQL02 for inbound connections (Run in maste)
create login MSSQL01_login with password = fjdOGv#!h';
create user MSSQL01_user from login MSSQL01_login;
create certificate MSSQL01_cert authorization MSSQL01_user
  from file = 'c:\cert\MSQL01_cert.cer';
--Configure the mirroring partners
--Run on MSSQL02 and MSSQL01 correspondingly
alter database demoDB set partner = 'TCP://MSSQL01:51033';
alter database demoDB set partner = 'TCP://MSSQL02:51033';
```

Figure 1. Implementing database mirroring

For instance, it is assumed that the primary and secondary servers are called *MSSQL01* and *MSSQL02*, respectively, and the database is called *demoDB*. Then Fig. 1 presents the T-SQL code which is adequate to establish database mirroring session.
Database mirroring supports also automatic failover if the following conditions are satisfied:

- It is configured in the high-safety mode.
- It involves a third SQL Server instance called witness.

2.3. Always on Availability Groups

Availability Groups have been introduced in SQL Server 2012 [2]. An availability group consists of a set of user databases (called primary replica) and one to four corresponding secondary replicas. Replicas are hosted by SQL Server instances which are installed on the Windows Server Failover Clustering (WSFC) cluster nodes. Each instance can be Failover Cluster Instance (FCI) or non-FCI. Hence implementing a WSFC cluster is crucial for deploying AlwaysOn Availability Groups. The general architecture is presented in Fig. 2.

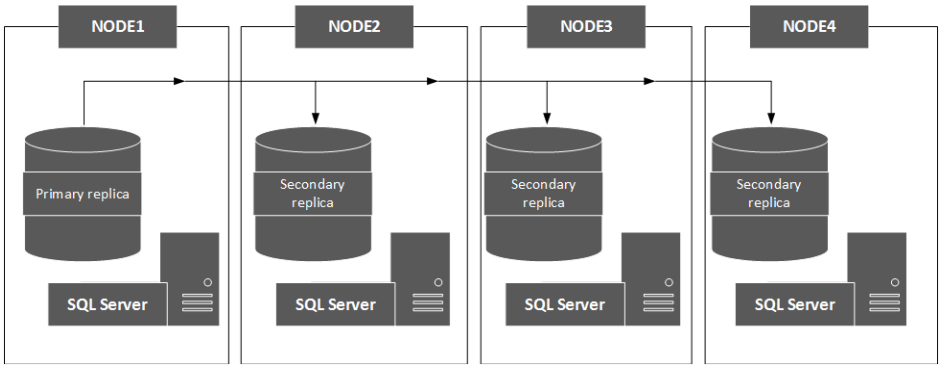

Figure 2. Architecture of AlwaysOn Availability Groups

Transactions applied to the primary replica are sent to each secondary replica. Each secondary database is synchronized individually with each primary database. One of secondary replicas is usually configured to accept read only connection to its databases. Backup operations of databases stored in secondary replicas are also supported.
The following steps are required to configure an AlwaysOn Availability Group [10]:

- Enable the AlwaysOn Availability Groups feature on each server instance (it can be done using PowerShell command: *Enable-SqlAlwaysOn -Path SQLSERVER:\SQL\Computer_Name\Instance_Name*).
- Create and configure an availability group listener.

- Connect to an instance which is dedicated to host a primary replica and create an availability group.
- Join a secondary replica to the availability group.
- Prepare each secondary database by restoring the proper backups of the databases belonging to the primary replica.
- Initialize data synchronization by adding secondary databases to the availability group.

3. Implementing SQL Server AlwaysOn Failover Cluster Instances

The approach is based on a Windows Server Failover Clustering (WSFC) cluster [11]. Such the cluster consists of several independent servers called *nodes* which work together to ensure availability of the MS SQL Server instance after a crash. Cluster nodes can be physical or virtual servers. The cluster configuration steps are the same both for physical and virtual nodes. We have implemented our prototype on the Windows Server 2012 with the Hyper-V role enabled. It involves the following components:

- Windows Server 2012 domain controller (cluster nodes belong to the same Active Directory domain).
- Shared storage - it is implemented using the Microsoft iSCSI Target service.
- Two cluster nodes with Windows Server 2012.

3.1. Creating Domain Controller

Microsoft Active Directory Domain Services are the most important component of any networks built on Windows server operating systems. They act as storage systems for network objects such as services, computers, users, printers, etc. A domain controller is a Windows server with Active Directory Domain Services (AD DS) installed. It deals with authentication and authorization requests within a Windows Server domain. The following steps are required to create and configure a domain controller:

- Create a new virtual machine with Windows Server 2012.
- Assign a network adapter with IP address 172.20.0.10 to this server – it allows clients connect to the cluster.
- Install the Active Directory Domain Services server role.
- Promote the server to a domain controller (We set the root domain name to *wmii.pl*).
- Create a domain account (called *wmii\clu_admin*) with the proper permissions to manage the cluster under this security context.
- Create two standard domain accounts (called *wmii\sql_srv* and *wmii\sql_agent*) for MS SQL Server services.

3.2. Creating iSCSI Target Server

iSCSI stands for Internet Small Computer System Interface. It allows servers to connect to a remote storage systems through a network. Targets are logical entities which involve logical unit numbers (LUNs) created on iSCSI devices. Each target has an IP address and some security settings. Servers use iSCSI initiators to communicate with LUNs assigned to targets. The following steps are required to create and configure an iSCSI target server and its clients:

- Create a new virtual machine with Windows Server 2012 (The server is called *target*).
- Join the *target* server to the *wmii.pl* domain.
- Assign a network adapter with IP address 172.20.1.10 to this server – it allows servers connect to the remote storage system.
- Install the iSCSI Target Server role.
- Create a new iSCSI target which contains the virtual hard disk to store MS SQL Server database files.

3.3. Creating cluster nodes

The following steps are required to create and configure the first cluster node (The second node should be configured in analogous manner):

- Create a new virtual machine with Windows Server 2012 (The server is called *node1*)
- Assign three network adapters to this server (Network adapter 1: 172.20.0.20, Network adapter 2: 172.20.1.20 – this network is dedicated for communication with the iSCSI target server, Network adapter 3: 172.20.2.20 – the heartbeat signal between nodes is sent using this network adapter)
- Join the *node1* server to the *wmii.pl* domain.
- Run iSCSI Initiator and configure the connection to the iSCSI Target server hosted on *target.wmii.pl*
- Initialize the disk (bring it online and create a single partition)
- Install the Failover Clustering feature on the *node1.wmii.pl* server.
- Run Failover Cluster Manager and create a cluster consisting of two nodes (*node1.wmii.pl* and *node2.wmii.pl*)
- Provide the cluster name (*mssql_cluster*) and the IP address for it (172.20.0.100)
- Change quorum configuration from *Node and Disk Majority* to *Node and File Share Majority*
    - Create a shared folder called *mssql_cluster* on *target.wmii.pl* and assign the proper share and NTFS permissions to it
    - Configure a file share witness in Failover Cluster Manager for *mssql_cluster.wmii.pl*
    - Provide the file share path (*target**mssql_cluster*).

The cluster architecture is presented in Fig. 3.

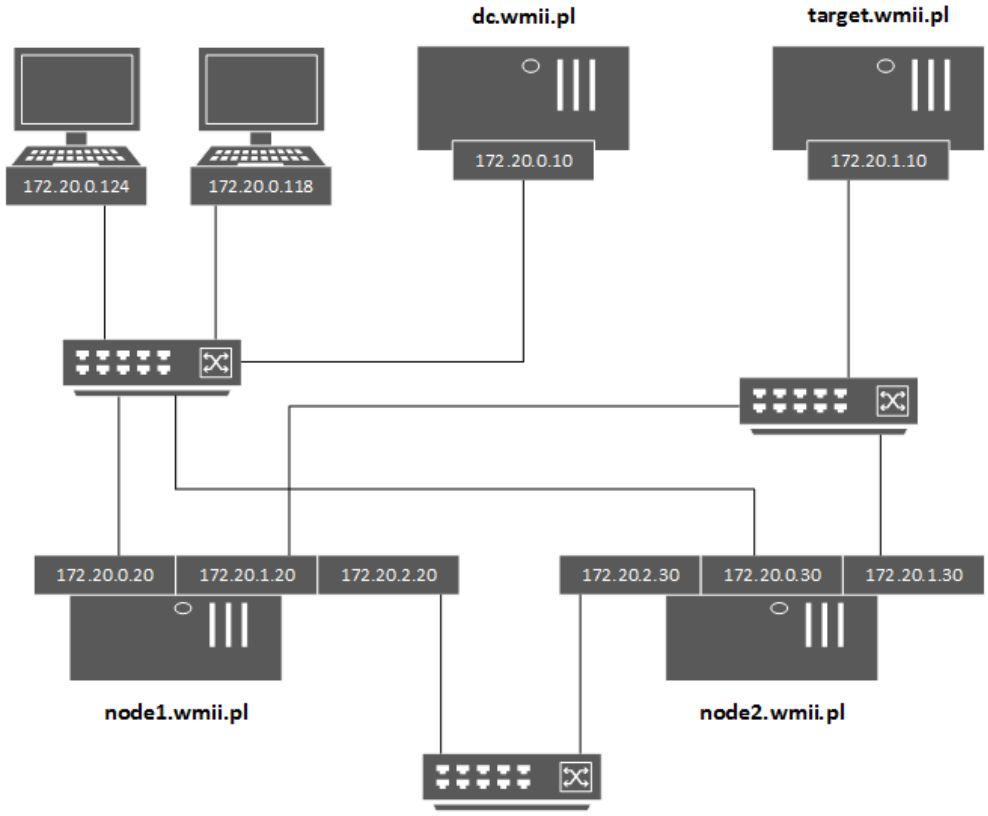

Figure 3. Architecture of a Windows Server Failover Clustering cluster

3.4. MS SQL Server Failover Cluster Installation

The following steps are required to install a SQL Server failover cluster:

- Run SQL Server Installation Center on *node1.wmii.pl.*
- Choose New SQL Server failover cluster installation.
- Install Database Engine Services and Management Tools.
- Set service account for the database engine service to *wmii\sql_srv*.
- Set service account for the agent service to *wmii\sql_agent*.
- Set network name for the MS SQL Server instance to *mssql* and set its IP address to 172.20.0.80.
- Run SQL Server Setup on *node2.wmii.pl* and choose Add node to a SQL Server failover cluster.

4. Conclusions

In this paper we have presented the overview of the high availability solutions for MS SQL Server. Mechanisms such as AlwaysOn Failover Cluster Instances, AlwaysOn Availability Groups, Database mirroring, Log shipping have been implemented and verified in our prototype environment based on a Windows Server Failover Clustering cluster. Log shipping and database mirroring provide high availability only at the database level. They have to be implemented separately for each database. In most cases a DBA work is required to make a secondary server available for user connections. Besides these techniques often require transferring logins and the passwords between MS SQL Server instances. Besides some data loss is possible in the case of failure. Database mirroring allows to eliminate such situations if its session is configured in the high safety mode. It can deteriorate performance because users receive conformations from the principal server if their transactions are committed on both partners. Database mirroring support automatic failover if its session involves the third server called witness and is configured in the synchronous mode. It is worth emphasizing that Database mirroring and Log shipping do not require shared storage and heart beat network. They are not based on a WSFC cluster.

AlwaysOn Availability Groups also implement high availability on the database level. The fact that an availability group can involve several databases is what gives it the advantage over database mirroring. Besides AlwaysOn Availability Groups support up to five availability replicas (one primary replica and one to four corresponding secondary replica). Each secondary replica can be configured to accept read only connections. In contrast to database mirroring availability replicas must be implemented in MS SQL Server instances which are stored on the cluster nodes. AlwaysOn Availabilty Groups provide three types of failover: automatic, manual and forced (with possible data loss). Automatic failover does not require the witness server like in case of a database mirroring session. Besides availability groups provide encryption and compression capabilities. Backups of secondary database are also supported.

AlwaysOn Failover Cluster Instances guarantee protection at the instance level. Failover is performed automatically in the case of hardware failures, operating system failures, services failures, etc. In contrast to the above solutions no configuration of clients and application is required during a failure. However FCI requires shared storage which can be implemented using iSCSI, Fibre Channel, server message block (SMB) file shares.

REFERENCES

[1] Books Online for SQL Server 2012 (2011): High Availability Solutions (SQL Server), Microsoft Corporation

[2] Bolton C. (2013) *Professional SQL Server 2012 Internals and Troubleshooting*, John Wiley & Sons, Inc.

[3] MySQL 5.7 Reference Manual (2016): High Availability and Scalability, http://dev.mysql.com/doc/refman/5.7/en/ha-overview.html

[4] Oracle Database High Availability Overview 11g Release 2 (2013), Oracle

[5] PostgreSQL 9.4.5 Documentation (2016): Chapter 25. High Availability, Load Balancing, and Replication, The PostgreSQL Global Development Group

[6] Worden D. (2004) *Storage Networks*, Apress

[7] Hotek M. (2009) *Microsoft SQL Server 2008 - Implementation and Maintenance*, Microsoft Press

[8] Books Online for SQL Server 2012 (2011): Configure Log Shipping (SQL Server), Microsoft Corporation

[9] Books Online for SQL Server 2012 (2011): Database Mirroring (SQL Server), Microsoft Corporation

[10] Books Online for SQL Server 2012 (2011): Overview of AlwaysOn Availability Groups (SQL Server), Microsoft Corporation

[11] Books Online for SQL Server 2012 (2011): Failover Cluster Overview, Microsoft Corporation

[12] Books Online for SQL Server 2012 (2011): AlwaysOn Failover Cluster Instances (SQL Server), Microsoft Corporation