High Availability in a J2EE Enterprise Application Environment

Udo Pletat

IBM Deutschland Entwicklung GmbH
Schönaicher Str. 220
71032 Böblingen, Germany
pletat@de.ibm.com

Abstract

Recent developments of middleware products enabling J2EE enterprise architectures include high availability features in, e.g., J2EE application servers and database systems. This is a step towards making the cornerstones of a J2EE enterprise infrastructure more autonomous, in the sense that their high availability becomes less dependent on dedicated system management software ensuring application and system availability. The article reports about a project exploring recent high-availability features of respective IBM products in an industry application context.

1 Introduction

The trend towards 24x7 operation of enterprise IT infrastructures increases the need to make this IT infrastructure highly available. Besides organizational guidelines how to operate an IT shop continuously, there are many approaches to automate system failovers not only in case of unexpected failures, but also for regular system maintenance or upgrades, see e.g., [1]. The term business resilience - i.e., continuous availability of the IT infrastructure without business relevant downtimes - captures the expectation of IT customers and the challenge for IT providers.

Recent developments of middleware products enabling J2EE enterprise architectures include high availability features in, e.g., J2EE servers and database systems, see [5] and [6]. This makes the cornerstones of a J2EE enterprise infrastructure more autonomous, in the sense that their high availability becomes less dependent on dedicated system management software.
The article reports about a technology assessment project exploring and evaluating recent high-availability features of respective IBM products (WebSphere Application Server[1] and DB2[2]) in an industry application context. The project emphasized obtaining reference results with respect to

- simplifications that can be achieved for setting up a high-availability infrastructure,

[1] WebSphere Application Server is a registered trademark of IBM Corporation
[2] DB2 is a registered trademark of IBM Corporation

- comparing system failover times with high availability configurations relying entirely on dedicated system management software, and
- integration of modern J2EE components of an enterprise architecture with approaches that are operational for 10 years or more.

2 Integration architecture for legacy and J2EE applications

The high-availability setup being discussed reflects an enterprise application integration architecture where legacy application systems are connected to more modern J2EE based applications running on a respective J2EE application server, see also [2]. Figure 1 below shows a typical structure of such an integration architecture.

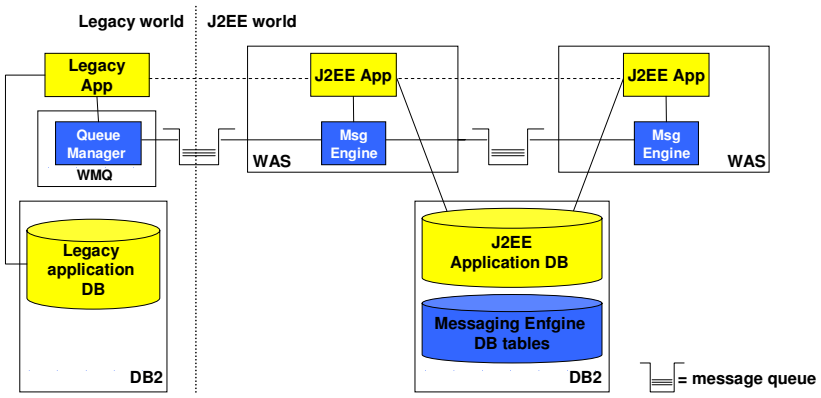

Figure 1: Legacy and J2EE application integration architecture

The fundamental components of such EAI structure are

- a (set of) legacy application(s) attached to a JMS-free message queuing infrastructure, e.g., based on message queuing systems like WebSphere MQ[3];
- the J2EE applications are hosted on respective application servers. Modern servers like WAS 6 include full JMS provider functionality capable to interact with respective non-JMS message queuing systems;
- the J2EE applications need to access databases, e.g., based on DB2. The database system holds the application data and serves as the persistent message store maintaining respective tables of the messaging engine[4].

The key objective of setting up a high-availability infrastructure for the J2EE part of such an enterprise integration architecture is a deployment of the J2EE part of the above enterprise application to a redundant hardware topology, where each J2EE application server is clustered with the cluster typically containing a primary and a standby machine; the same applies to the database server. In order to achieve short

[3] WebSphere MQ is a registered trademark of IBM Corporation

[4] The database view is slightly simplified for the time being, but there is no conceptual difference whether to have a single DB shared by multiple J2EE servers or each J2EE server accessing its 'own' DB or having criss-cross database access. Typically, one may assume that the legacy applications run against their own database.

failover times, a so-called 'hot standby' approach is mandatory; in such a configuration the primary server performs all normal operation while the standby server is ready to take over, but idles until the failover becomes necessary.

3 High-availability scenarios and requirements

Typical high-availability scenarios to be covered are

- network disconnections between the legacy applications and J2EE application cluster or between the J2EE cluster components,
- failure of a J2EE application server, and
- database server failure.

The kernel requirements for a high-availability system setup are usually

- ease of setup and operation and
- short failover times with (close to) no human intervention.

The 'classical approach' to cope with these high-availability requirements is to use shared disks holding the relevant data of the middleware component to be made highly available and two server nodes - primary and standby – accessing the component's data on the shared disk, see also [13]. In case of a failover, high availability management software like Tivoli System Automation (TSA)[5] or High Availability and Cluster Multi-Processing (HACMP)[6] (see [9] and [10]) starts the failed middleware component on the standby machine. This failover process makes sure that all relevant file systems, IP addresses, etc. for that component are switched over to the standby node, see left hand part of Figure 2 below illustrating the shared disk concept for the database failover.

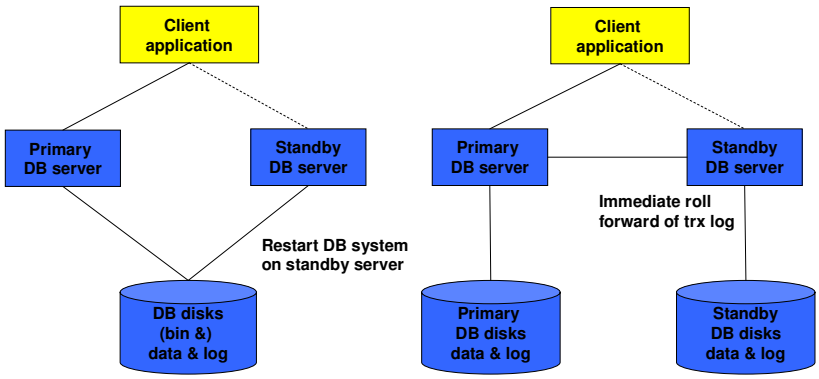

Figure 2: Shared disk plus restart versus transaction log replication

The right hand part of Figure 2 above shows how data replication features in a product like DB2 8.2 can be used for making the database system highly available:

- in-transaction data replication through log shipping from the primary to the standby database server assures that the standby database keeps track with the DB operations executed against the primary database;

[5] Tivoli System Automation is a registered Trademark of IBM Corporation

[6] High Availability and Cluster Multi-Processing is a registered Trademark of IBM Corporation

- automatic client re-route assuring that the database failover is basically transparent to client applications.

The transaction log forwarding approach - also known as 'eager data replication', see [11] or [12] - combines aspects of high-availability and data replication for disaster recovery scenarios. Looking a bit closer at the options for a database high-availability setup, Table 1 summarizes some arguments for/against the two main approaches.

Table 1: Shared disk versus replication approach to database high availability

	Pro	Con
Shared disk	- no performance penalty in normal operation	- dependence on storage area network - database corruption due to failure may require database restore - more complicated system management software configuration
Data replication	- simpler system management software setup - database restores after primary database failure are less likely	- performance penalty in normal operation

For the database side the overall failover time, i.e., the time between the occurrence of a failure and the point in time when the applications resume processing after the component failover is complete, is not so much influenced by the database startup times. However, in a complicated storage area network setup for high volume databases, the disk switching may consume a significant portion of the failover time.

On the WebSphere side, the main argument for the hot standby approach is indeed the shorter failover time because there is no need for restarting the application server and the deployed applications on the standby node. Application server start time plus application loading time can be significant, so the availability of the standby application server with all applications being started there – but remaining passive until a failover occurs – is an essential decision criterion.

High-availability features of a product like WebSphere Application Server Version 6 include

- setup of clusters consisting of redundant application server nodes with failover policies like '1 of N' (meaning that application server runs on 1 out of N available server machines), 'static' (meaning that application server has to run on a dedicated server machine), plus user-definable policies to let also external high-availability management software interact with WebSphere Application Server,
- support of so-called *singleton services* that run only once in a WAS high-availability cluster; typical singletons are the messaging engine and the transaction manager;
- improved failover times compared to using a shared disk approach plus typical high-availability management software.

These new features reduce the need for using dedicated high-availability management software and move J2EE server high-availability away from being 'only' based on workload management features where - in a J2EE server cluster - the workload of a failed server will be taken over by other servers in the cluster.

4 High Availability using recent WebSphere and DB2 technology

In the previous section we have outlined some key features of recent middleware products and now we sketch how these features can be exploited for setting up a highly available J2EE server infrastructure where

- high availability requirements of the J2EE server can be satisfied with the WAS internal high availability features[7]. The key element being the hot-standby application server which can takeover immediately without any need for server re-starts and replaying transactions other than those that were 'in-flight' at the point in time the primary server failed.
- high availability features of the database for the J2EE server(s) are fulfilled by using the high availability and disaster recovery feature of DB2. For database failover, this means that we also arrive at a hot-standby setting where at most the in-flight transactions have to be replayed on the backup database server after the failover has occurred[8].

The next sections provide some details about the respective setup of WAS and DB2.

4.1 WAS System setup

The layout of the so-called WAS cell is illustrated in Figure 3 below. The setup for the study consists of a 5-node cell where the nodes on the left-hand side make up the so-called primary tower while the two nodes on the right-hand side make up the backup tower, see Figure 3 below. The transaction log files for each J2EE server pair (primary and backup) reside on a NFS Version 4 enabled file system, which can be attached either in NAS-style (Network Attached Storage through a respective file server – which in our case also hosts the deployment manager administrating the WAS cell configuration) or SAN-style (Storage Area Network via directly attached disks).

WebSphere controls the failover of the application server itself plus the transaction manager and messaging engine. Using the so-called 'connection proximity' configuration allows to specify that an application can only participate in message traffic, if a messaging engine is hosted by the application server where the application is deployed. This allows for having all applications started both on the primary and the standby application server; the availability of the messaging engine on either of the cluster's application servers defines whether of the application

[7] When WAS fails over a connectivity loss between the WAS Messaging Engine and the WebSphere MQ Queue Manager cannot be recovered automatically by WAS alone, because WAS does not support an 'automatic client-re-route' feature as offered by DB2. Respective TSA scripting failing over the WAS IP address fills this current gap.

[8] Also here there is a little gap in the sense that the triggering the takeover of the backup database fully automatically requires a simple use of high-availability management software like TSA. This automates the decision on whether the primary database server is considered down so that a failover has to be initiated.

deployed on the primary or standby application server may process messages, thus making the applications 'quasi-singletons'.

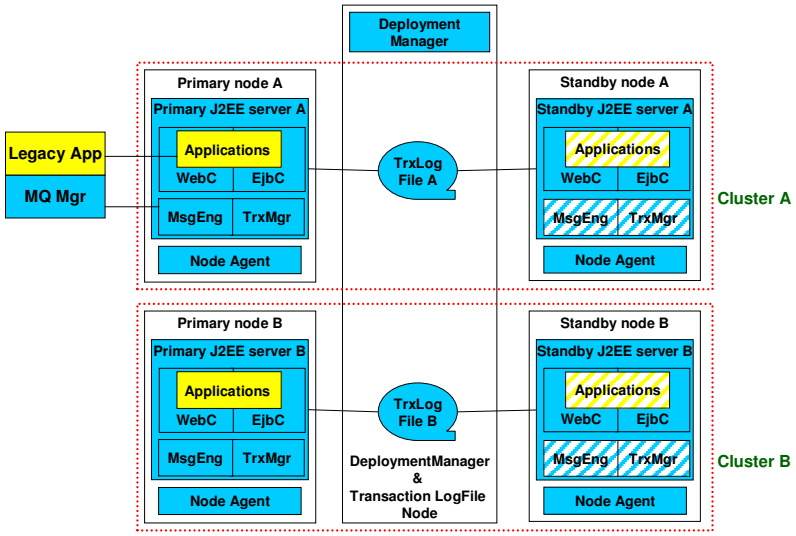

Figure 3: High availability layout of the J2EE server cell

4.2 DB2 System Setup

Exploiting the HADR (high-availability and disaster recovery) feature of DB2 leads to a quite straight-forward setup of the highly available database, as illustrated in the right hand part of Figure 2 above. The database system has to maintain both the J2EE application data and it also serves as the persistent store for the Messaging Engine. In most cases it is a recommended practice to separate the application data from the Messaging Engine data leading to at least two databases for which a primary and standby instance have to be configured for use with HADR. There are three reliability modes for transferring the transaction logs from the primary database node to the standby:

- *'synchronous'* - the original operation on the primary database succeeds when it receives a notification from the standby database that the transaction log entry has been written to the disk on the standby server;
- *'near synchronous'* - the original operation on the primary database succeeds when it receives a notification from the standby database that the transaction log entry has been written to the main memory on the standby server;
- *'asynchronous'* - the original operation on the primary database succeeds when it has transferred the log file entry to its TCP/IP system.

The three modes allow for balancing between performance and reliability requirements as the transfer of the transaction log data from the primary to the standby database server is part of the original operation executed on the primary server.

According to the database replication scheme proposed in **[12],** the HADR approach falls into the category of so-called *'eager replications'* and especially into the category *'primary copy, linear, voting'* (for the synchronization modes 'synchronous' and 'near synchronous') and *'primary copy, linear, non-voting'* (for the synchronization mode 'asynchronous'), respectively.

4.3 Making your applications HA-aware

Of course, applications do not become highly available by 'simply' making the middleware infrastructure to which they are deployed highly available. In the preceding sections we have 'only' described the prerequisites that have to exist anyway in order to prepare your IT infrastructure for making the applications highly available.
As outlined also in [1], the applications must be able absorb failing accesses to the middleware in case, e.g., the database becomes unavailable for a certain period of time. Typical strategies include

- enabling access retries;
- saving consistent intermediate state to disk so that in case of a failover and application restart on the backup node, the new instance of the application knows the state of the old one before that one 'died';
- capturing exceptions caused by the failovers appropriately, e.g., DB2 supports an automatic client re-route where applications will be notified about a database failover through respective exceptions;
- fnally, the transaction structure for processing incoming messages has a high impact on which high-availability provisions have to be taken in the application code. Especially the message processing may vary from using Message Driven Beans together with Container Managed Transaction (simple) to using plain JMS and the Java Transaction API (complex), see also [4] and [5].

The table below gives a quick overview on the various transaction management alternatives that are of interest when the message processing shall take place under transactional control in a J2EE enterprise application, see, e.g., [3] and [5].

Table 2: Transaction control alternatives

	Transaction control mechanism	**Message consumption**	**Failover behavior**
JMS	Java Transaction API	inside transaction	- transaction rolled back - message back to queue after rollback
	Java Transaction API	outside transaction	- transaction rolled back - message no longer on queue after rollback
MDB	Bean managed transactions with JTA transaction demarcation	outside transaction	- transaction rolled back - message no longer on queue after rollback
	Container managed transactions with transaction mode 'required' or 'notSupported'	inside transaction	- transaction rolled back - message back to queue after rollback

To exploit the transaction recovery services of the J2EE server after the failover is accomplished, it is a good practice to consume messages within the transaction scope – no matter whether using JMS or MDB. Whether to use JMS or MDB depends on the degree of flexibility of the transaction structure for processing messages required by the J2EE applications.

5 Results and experiences

The results of this advanced study exploring the high availability features of recent middleware products being the key elements of typical J2EE base enterprise IT architectures are both promising and creating new expectations.
On the positive side we find simplified setups of a highly-available J2EE infrastructure, including database access. In addition the so-called 'hot-standby' servers are becoming reality with shortened failover times, especially for J2EE application servers.
On the other hand, new expectations are also being created, e.g., by asking for improved failover policies both for WebSphere Application Server and DB2; finally, integration with upcoming trends of on-demand operating environments which improve the dynamic management of middleware infrastructures may lead to the next generation of high-availability approaches.

Acknowledgement
The author would like to thank Michael Haeberlen, Andrew James, and Sven Stueven for their contributions to bring the system described in the article to life.

References

[1] Blueprints for High-Availability, E. Marcus and H. Stern, Wiley Publishing Company, 2003
[2] Patterns of Enterprise Application Architecture, M. Fowler, Addison-Wesley Publishing Company, 2003
[3] Java Message Service, Version 1.1, Sun Microsystems Corporation, 2002
[4] Java Transaction API, Version 1.0.1b, Sun Microsystems Corporation, 2002
[5] Enterprise Java Beans, R. Monson-Haefel, O'Reilly Publishing Company, 2002
[6] WebSphere Application Server Version 6, Information Center, IBM Corporation, 2004
[7] DB2 Version 8.2, Data Recovery and High Availability Guide and Reference, IBM Corporation 2004
[8] IBM WebSphere Version 6: Performance, Scalability and High-Availability, IBM Redbook, IBM Corporation 2005 (draft)
[9] Tivoli Systems Automation for Multi-Platforms, Guide and Reference, Version 1.2, IBM Corporation 2004
[10] High Availability and Cluster Multi-Processing for AIX, Version 5.1, IBM Corporation, 2003
[11] The Dangers of Replication and a Solution, J. N. Gray, P. Helland, D. Sasha and P. O'Neil, Proceedings 1996 ACM SIGMOD International Conference on Management of Data, pp 173-182, Montreal, June 1996
[12] Database Replication Techniques: A Three Dimension Comparison. M. Wiesmann, F. Pedone, A. Schiper, B. Kemme, and G. Alonso, Proceedings 19th IEEE Symposium on Reliable Distributed Systems (SRDS 2000), Nuernberg, Germany, October 2000
[13] IBM WebSphere Version 5.1: Performance, Scalability and High-Availability, IBM Redbook, IBM Corporation 2004