SAP Web AS Java 6.40 – a Reliable, Scalable, Efficient J2EE Application Server

1 New Architectural Solutions Provide for Reliable Operations of the System

1.1 The Java Instance Concept
1.1.1 Reuse of Real-life Proven Messaging and Locking Technologies
1.1.2 Store Cluster Configuration Data in a Database
1.1.3 Support Multiple Server Processes Within a Java Instance

2 New Clustering Architecture Guarantees High-level Scalability

2.1 6.40 “Star” Architecture vs. 6.20 Point-to-point Connected “Ring” Architecture
2.1.1 The Facts

2.2 Communication Types Between Cluster Elements

2.3 Minimal Downtime to Scale Your Cluster

3 Java Instance Life-cycle Management Through the Startup and Control Framework

3.1 Features
3.2 Architecture

4 High Availability Solutions
4.1 No Single Point of Failure
4.2 Session Fail over Mechanism

5 Remote Debugging

6 Summary
This article describes the most basic features of SAP Web Application Server Java in the new version 6.40. It mainly concentrates on changes and describes them on a technical level for technical personal. Basically it describes the cluster architecture and the derived features of scalability, stability and performance improvements. Further we will dive into Java instance lifecycle management and how this is accomplished with Web Application Server Java and how this helps for high availability solutions despite other facts also mentioned.

Additionally there is an insight on how remote debugging works with the new system.

1 New Architectural Solutions Provide for Reliable Operations of the System

1.1 The Java Instance Concept

SAP Web AS Java 6.40 adopts the concept of an instance as an administrative unit that groups together components of an SAP system (known from the SAP R/3 systems). Well integrated into the Java Startup and Control Framework, it provides reliable life-cycle management of the processes running within a Java instance.

The components that comprise a Java instance are:

- Central services – a message server to facilitate cluster elements communication, a locking server implementing the SAP locking concept, and a database as a central configuration data storage.
- Java dispatcher – a component responsible for client request dispatching to server processes.
- One or many server processes – provide the environment the J2EE applications run in.
1.1.1 Reuse of Real-life Proven Messaging and Locking Technologies

Using the Message Server (a solution used for years in the ABAP world) as a central infrastructure for data exchange (small datasets only) between the participating cluster elements provides another aspect of the SAP Web AS Java’s reliability. This lightweight solution provides the following services:

- Notification of events that arise in the cluster, for example, if a cluster node gets lost (because the instance failed or shut down), or if a service is started or stopped. Therefore, the message server is always aware of the cluster elements’ availability.
- Forwarding of messages and requests to all participants (broadcast).
- Provide cluster elements information to the SAP Web Dispatcher for load balancing purposes.
- Guaranteed message transmission. Messages that failed to be sent because of a possible message server crash are re-sent when the message server restart.
- Exchange of cache information in the cluster for communication efficiency.

The other central service that is part of the Java instance deals with managing lock operations over logical objects. The enqueue server that is based on the mature, real-life proven SAP Locking Concept implements this functionality. The genuine idea of this concept is to interface the locking management process from the database-specific implementation so that it can be extended and better controlled to fit the SAP’s requirements for reliable transactions. That is, problems that arise as a result of different types of logical units of work (for database systems and SAP transactions) are avoided. Implementation of the SAP Locking Concept also results in enhanced performance - time-consuming database changes are carried out asynchronously at the end of the logical units of work after the user has already moved on to the next logical unit of work.

This process then checks the lock table to determine whether the lock request collides with an existing lock. If this is the case, the request is rejected. Otherwise, the lock is set and an appropriate entry is made in the lock table.

The enqueue service is perfectly integrated into the Java Instance, and is used to handle locks for objects used by both internal server components and deployed applications. The SAP Web AS Java internal components use the enqueue server functions directly through the Locking Manager API, whereas the applications see only the on-top-of-it Locking Adapter Service API. The latter ensures that a J2EE application can only set or release locks for the current user using the application.

1.1.2 Store Cluster Configuration Data in a Database

The SAP Web AS Java uses a central database to store cluster configurations, applications data, as well as enterprise runtime byte code. From the cluster architecture perspective, the benefits of this approach are that the processes of central administration of the cluster, central deployment, central installation and upgrade become possible.

In addition to the default Max DB major databases like Oracle, DB2, and MS SQL Server are supported too.

The process of reading and modifying cluster configuration data from the database is handled by the SAP Web AS Java enterprise runtime, namely the Configuration Manager. It is able to manipulate the configurations’ data structure, as well as the configuration objects themselves. It uses the locking services provided by the Enqueue Server to manage concurrent access to shared objects, and the Open SQL for Java framework to connect and work with the RDBMS.

The SAP Web AS Java enterprise runtime starts in a separate virtual machine, using the bootstrap data – that is, information stored locally on the same physical machine and which is used to initialize the core of the dispatcher and the server processes. When the Configuration
Manager starts, it connects the SAP Web AS Java to the database and enables it to retrieve the data needed for starting the other modules.

Additionally to this, any deployment to the server works in a transactional mode so that applications, upgrades, or services are either deployed completely or not at all. Breaking the deployment process for whatever reason never will result in bad installations that make the server unusable.

### 1.1.3 Support Multiple Server Processes Within a Java Instance

It is possible to run more than one server process within a single Java instance. This way you can optimally utilize the hardware resources of the box the instance is running on (performing vertical scaling), as well as add new hardware boxes with new cluster elements (performing horizontal scaling) to increase linearly the load that the cluster can process.

Furthermore, this creates the option to shrink the risk of losing requests based on Java Virtual Machine (JVM) crashes. In general, a JVM crash means that a complete server goes down and takes every request running on this server with it. Providing additional servers and therefore dividing users sessions between those additional servers will at the same time reduce the risk of failing due to JVM crashes.

### 2 New Clustering Architecture Guarantees High-level Scalability

#### 2.1 6.40 “Star” Architecture vs. 6.20 Point-to-point Connected “Ring” Architecture

Analysis of the reasons behind limited scalability of the SAP J2EE Engine 6.20 cluster has brought the new “star” architecture of the SAP Web AS Java 6.40 cluster into life. Ever since, this has proved to be a step in the right direction towards achieving the first class ABAP-like cluster scalability.
The new 6.40 architecture is much more successful than the 6.20 version, mainly because cluster data synchronization is now more efficient and stable. Efficiency means less network overhead for synchronizing a single piece of data.

The pure math says – for the 6.20 point-to-point “ring” architecture, \( n^*(n-1)/2 \) connections are needed to synchronize of \( n \) cluster elements, compared with only \( n \) connections for the “star” architecture of 6.40. That is, adding new nodes to a “ring” cluster increases the number of connections nearly quadratically. This will heavily stress network connections in a large environment. That’s not an issue with a “star” cluster with linear growth of connections. The latter provides good opportunities to scale your cluster horizontally to meet unexpected increase in user demand in a flexible and time-efficient way. Adding a new hardware box with SAP Web AS Java running on it to the cluster grows processing power and hence increases the capacity of serving more parallel users. With the presence of an efficient load balancing algorithm on the Java dispatcher, the SAP Web AS Java achieves linear scalability.

The stability and consistency of the 6.40 cluster elements configuration is guaranteed by the central services – the robustness of the database to store the configuration data, the locking service to manage locks when modifying the data, and the efficiency of the message service that manages distribution of internal cluster messages and events between the cluster elements.

### 2.1.1 The Facts

The SAP Web AS Java cluster tests have proved its extended scalability power. The largest cluster setup up to now that was run during testing comprised 60 cluster elements with no applications deployed (only the services and managers running). With default applications deployed, the 6.40 cluster was scaled up to 50 elements. The tests showed no evidence that the uppermost limit of the cluster elements count was reached. It is believed that if a productive system is capable of scaling more than 40 elements, this means it has practically unlimited scalability.

### 2.2 Communication Types Between Cluster Elements

Cluster elements need to exchange a wide variety of messages between each other in order to provide for the normal operations of the whole cluster. The SAP Web AS Java has a dedicated enterprise runtime module, called the Cluster Manager, which addresses those communication efficiency issues.

The Cluster Manager implements three different types of communication between elements within a cluster group. This facilitates the variety of cases of exchanging internal messages. These are:

- **Message server communication**
  
  This type of communication is established through the message server used as a dispatcher when sending messages. This way of communication has one advantage – it has a failover function that avoids the loss of information. Whether messages are sent through the message server or through the lazy communication is determined on a Cluster Manager level. The message body size is checked against a threshold value and is sent through the message server only if the message is shorter than the threshold. The semantics behind this rule is to keep the message server a lightweight component that carries out the most critical cluster elements communication.

- **Lazy communication**
  
  This function is used for quickly exchanging large amount of information between two servers without using the message server as an intermediary. Instead, the information is transported through sockets that are opened on both servers.

- **Session communication**
This communication type is realized by a direct connection between dispatcher and server in one cluster group. It is used to process client requests faster. The session communication has only one-way operations – from dispatcher to server.

2.3 Minimal Downtime to Scale Your Cluster

Scalability itself as an option to increase the processing power of a cluster was never that important without providing the framework and the ease-of-use tools to add cluster elements to enable on-demand capacity planning and flexibility in deployment. With SAP Web AS Java, adding a new server element to your cluster is just a few clicks away. Using the Config Tool that is provided with the standard installation, you edit the property file of the corresponding instance in the database by adding the appropriate information (such as server name, ID, installation directory, etc.) about the new process. The procedure can be repeated if more than one new element is to be added to the cluster. The actual bootstrapping and starting of the new elements is a task performed by the Java Startup and Control Framework – the JControl program is restarted in order to run the bootstrap program against the changed instance properties. After the bootstrap, the JControl starts the necessary JLaunch processes to start all the elements within the current instance.

3 Java Instance Life-cycle Management through the Startup and Control Framework

The SAP Web AS Java provides a framework that integrates the life-cycle management of a Java instance into the existing SAP instance management solution. This framework makes it possible to inherit the tried-and-tested process management concepts from the ABAP stack. The Startup and Control Framework is one of the key solutions used in the SAP Web AS Java to ensure the reliability of the server infrastructure.

3.1 Features

- Strictly defines the sequence of starting the Java processes for the Java instance (bootstrapping, starting the separate Java dispatcher and server processes). Embedding processes to provide single-source of control.
- Serves as a “watchdog” – controls the Java processes that run in a Java instance and restarts them if necessary. This eliminates the possibility of hanging elements disturbing the operations of the cluster.
- Provides for remote control of all the Java instances in the cluster. Allows cluster-wide life-cycle management activities such as starting, restarting or shutting down the Java processes.
- Includes a Startup Framework client that can be easily integrated into other pieces of software that need control over the Java instance’s life-cycle (for example, integration into the SAP NetWeaver Developer Studio to enable the restart of the instance).
- Enables native functionality to be bound to the Java processes running within the Java instances.
- Includes functionality for remote debugging to boost supportability.

3.2 Architecture

The Java Startup and Control Framework consists of the JControl and JLaunch programs. The JControl is the master program for the Java instance and starts all necessary JLaunch programs in the correct order, and controls them later on.
The JLaunch program executes any Java program (similar to the standard java.exe program). It loads the JVM into its own address space and then the required cluster element. The program can receive notification from the JControl process via the named pipes to stop the cluster element and then terminates if JControl stops running. The JLaunch program executes also the bootstrap Java program that synchronizes the binaries from the database with the file system and that creates the property files describing the Java instance.

The following steps are involved in starting the Java instance:

- JControl is started (on Windows by the SAP start service; on UNIX platforms by the startsap script).
- JControl initializes the SAP signal handling to be able to handle signals received.
- JControl starts JLaunch with the bootstrap.properties file (1). This executes the following steps:
  - The first argument of JLaunch is the PID of the parent process (JControl). JLaunch starts a thread, which ends the JLaunch process, if the parent process, JControl, fails.
  - Creates JVM arguments and initializes hosting of the VM.
  - Loads the VM into its own process, initializes the VM and starts the bootstrap program.
  - The bootstrap program synchronizes the binary data of the Java database with the local file system (2).
  - The bootstrap program reads the J2EE instance description from the Java database and writes the file instance.property (3). The file instance.property contains the description and the arguments of the Java cluster elements that are to be started.
- JControl reads and creates a list of the Java cluster elements to be started (4).
- JControl starts a JLaunch process for each cluster element (5). This executes the following steps:
  - The first argument of JLaunch is the PID of the parent process (JControl). JLaunch starts a thread, which ends the JLaunch process, if the parent process, JControl, fails.
  - Creates JVM arguments and initializes hosting of the VM.
  - Loads the VM into its own process, initializes VM and starts the Java cluster element. This executes the following steps:
    - Starts the “offline” configuration manager to read the properties for the Java Enterprise runtime from the database and to save them in various hash tables (6).
    - Stops the “offline” configuration manager and starts the Java Enterprise runtime with the saved properties.
    - Starts the service framework and the services.

4 High Availability Solutions

To ensure continuous availability of cluster operations and avoid the unplanned downtime of the system, the SAP Web AS Java cluster provides high availability solutions for the possible single points of failure, as well as a transparent session failover mechanism.
4.1 No Single Point of Failure

Potential single points of failure in the SAP Web AS Java cluster are the elements that provide the central services, namely the database, the enqueue server and message server.

The message server is a very lightweight solution, so instant restart is the way to get it up and running again. The restart is initiated automatically by the Startup and Control Framework if the message server crashes.

The enqueue server solves the high availability issues by providing server replication on a different host. In the case of a crashed enqueue server, the “stand-by” replica server takes control over the locking management process in the SAP Web AS Java cluster.

The database high availability solutions are provided by the corresponding database vendor. Reliability and effectiveness of this solution is critical for the reliability of the SAP Web AS Java cluster operations.

4.2 Session Fail over Mechanism

Providing fault tolerance on server processes means that a mechanism to preserve the state of the active sessions must be available. Without session failover, any information concerning the state of the user at the time of the crash is lost. The requests are just dispatched to another server process in the cluster where the same business objects are available.

The picture is different with the session failover in use. Stateful information is made persistent, which enables the state to be retrieved when the request is dispatched to an available server process. The session failover is completely transparent to the applications (nothing has to be done explicitly in the application’s code). The session state persistence is achieved through Java objects serialization mechanism.

The Session Failover Service on the SAP Web AS Java manages the serialization process. The main features of the session failover framework are:

- All server processes running within the SAP Web AS Java cluster can retrieve serialized session data if database persistence is used. For cluster setups on a single hardware box, a file system can also be used for storing serialized session data.
- Session Failover Service API to provide serialization for any kind of session data.
- Advanced serialization mechanism – this makes it possible to serialize non-serializable objects such as: references to naming contexts, references to enterprise bean interfaces, references to enterprise beans session contexts, and references to user transaction objects.

Finally, session failover must be used efficiently because preserving the session state in cases of server crashes reduces the performance of applications.

5 Remote Debugging

Another important aspect of clustering that impacts reliability and supportability is providing a solution for remote debugging. The purpose is to provide isolated cluster elements that carry out the debug operations without affecting the operations of the productive elements. In SAP Web AS Java 6.40, isolation takes place on two levels: first, a debug element is isolated from any load balancing algorithm that takes place on the Java dispatcher, and second, it is removed from the destinations list of addresses for broadcast messages that is kept by the message server. This way any productive request is prevented from being dispatched to the debug element, and no internal events that are caused by debugging operations can be distributed among the productive elements of the cluster.

Remote Debugging Isolation

A remote debugging client and a Startup Framework client have been integrated into the SAP NetWeaver Developer Studio to provide central, ease-of-use control over the remote debugging process on the SAP Web AS Java. For the developer, activating debugging (preparing a debug cluster element and isolating it from the productive elements in the cluster) means just a single mouse-click – everything else is done by the Java Startup and Control Framework.
Preparing and Activating Remote Debugging

To get more detailed about what goes on behind the scene, here is an outline of the process:

1. The Startup Framework client inside the SAP NetWeaver Developer Studio sends the "enable debugging" notification to the message server.
2. The message server forwards the notification to JControl.
3. JControl switches on the debug mode and starts the process.
4. The debug element uses the specified debug port from the instance.properties file.
5. Isolate the debug element from the cluster communication:
   a. The Startup Framework client sends the activation notification to the message server.
   b. The message server removes the element from the broadcast destination list. Once this step is complete, the debug element is no longer synchronized with the productive cluster elements.
   c. The message server sends the activation notification to the debug element. The debug element changes its own debug mode from "Off" to "On". All other cluster elements will receive a notification about this change for the debug element.
   d. The SAP NetWeaver Developer Studio starts the debugging session and connects to the debug port of the debug element. The Startup Framework client provides the port configuration transparently.
6. Start debugging the application

After finishing debugging, the debug element must be closed. This is again performed by the Java Startup and Control Framework, and is triggered by a "disable debugging" notification sent from the SAP NetWeaver Developer Studio to the message server. Apart from that,
JControl has a pre-configured debugging timeout (e.g. 10 mins). This is the maximum debugging time up from the debugging activation. The debug element is automatically stopped after this debugging timeout by the Startup Framework to prevent debug elements that are not synchronized with the cluster from running for a long time.

6 Summary

Although SAP started relatively late in the J2EE application server world, it definitely has the chance to build its robust Java "workhorse" which, together with the ABAP one, will move the whole NetWeaver foundation towards the future. Having the conceptual and the technological know-how from the R/3 world, and the ability to transfer it to the Java application server is a key indicator of success. So, if you think that the ABAP application server is the most reliable and flexible application server ever, you are probably in bad faith –SAP Web AS Java is not falling behind because it:

- Has a new cluster architecture that allows for extended scalability of the cluster
- Adopts the SAP instance concept as an administrative unit for cluster elements and incorporates it into the Startup and Control Framework to achieve reliability for cluster operations
- Provides high availability solutions for each cluster element type to eliminate single-points-of-failure.