Release: XI 2.0

XI
Exchange Infrastructure

XI Tuning

Subtitle

History

Version	Status	Date
1.5	Release	2004-03-18

Content

1 Introduction

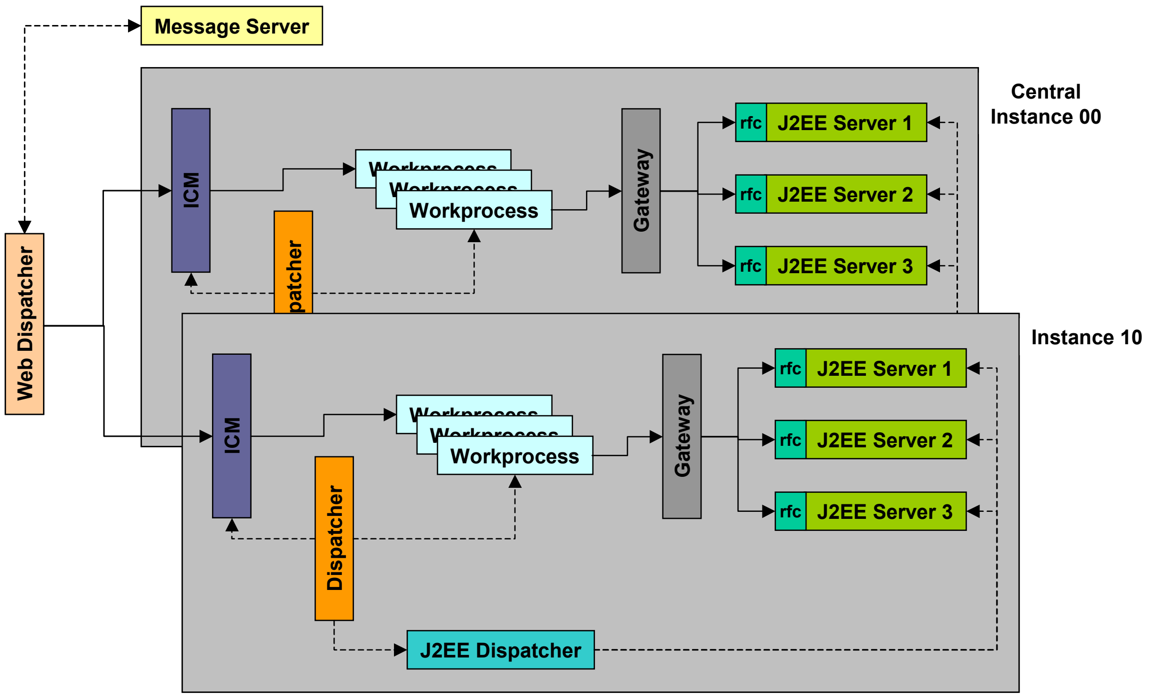
The purpose of this paper is to summarize configuration and tuning aspects for running SAP Exchange Infrastructure (XI). Experiences are based on performance tests and several internal projects. The area of Java VM tuning is of special importance for scenarios using the XI mapping service.

2 Glossary

ABAP Engine	Web Application Server (ABAP part)
GC	Garbage Collection (Java VM)
ICM	Internet Communication Manager (icman)
J2EE Engine	Web Application Server (J2EE part)
J2EE cluster	SAP J2EE Engine cluster
J2EE dispatcher	SAP J2EE Engine dispatcher
J2EE server	SAP J2EE Engine server instance
JCo	SAP Java Connector
OS	Operating System
RFC	Remote Function Call
rfcengine	rfcengine service of a SAP J2EE Engine server instance
SLD	System Landscape Directory
WebAS	Web Application Server
WebAS instance	Web Application Server instance (ABAP + J2EE)
XI	Exchange Infrastructure

3 WebAS architecture

The picture below gives a short overview of overall WebAS architecture within XI context



A single WebAS instance consists of an ABAP part (SAP Kernel 6.20) and a J2EE part (SAP J2EE Engine 6.20). Standalone ABAP or J2EE WebAS instances are not considered. Each J2EE Engine consists of a dispatcher and one or more servers (comprising a cluster). All WebAS instances are connected to a central instance. All J2EE clusters are connected to the central instance's J2EE cluster. This allows for central administration and deployment of J2EE applications.

Within XI, inbound and outbound HTTP requests are not handled by the J2EE Engine's HTTP service. J2EE applications (Integration Builder, Mapping Service, SLD) are called from the ABAP based Integration Engine via JCo interfaces (Rfc protocol). Each J2EE server runs a rfcengine service. Threads of the rfcengine connect to the central or to their local WebAS gateway.

4 Tuning areas

The main tuning areas identified are

- WebAS configuration
- Java VM tuning
- Database tuning
- Application tuning

4.1 WebAS configuration

An XI Integration Server consists of the following components

- WebAS instances (ABAP+J2EE)
- Web Dispatcher (optional)
- Technical Adapters (File, JDBC, JMS)
- Java Proxy Runtime

Configuration of WebAS instances is crucial for XI performance and throughput.

Relevant settings for WebAS instance configuration

- # of WebAS instances (ABAP+J2EE)
  influenced by # of CPUs available (or physical servers)
  and amount of inbound/outbound traffic
- # of dialog work processes per WebAS instance (ABAP)
  influenced by # of parallel requests to be processed per instance
- # of qRfc inbound/outbound queues (ABAP)
  influenced by # of parallel requests to be processed per instance
- # of J2EE servers per WebAS instance (J2EE)
  influenced by # of parallel mapping requests per instance
  and garbage collection overhead
- # of rfcengine threads per J2EE server - XI mapping service (J2EE)
  influenced by # of parallel mapping requests per instance
  and garbage collection overhead

Depending on the number of available CPUs on a server it makes sense to run multiple WebAS instances per Integration Server. Running 1 WebAS instance per 4 CPUs (or a CPU board) was a suitable approach during internal tests.

Example configuration

For a performance test on a large SMP server (72 CPUs) configuration was as follows:

16 WebAS instances (using 4 CPUs each)

30 dialog work processes per WebAS instance

3 J2EE servers per WebAS instance

10 rfcengine threads (mapping) per J2EE server

4.1.1 SAP profile parameters

In general profile parameters should be set according to Technology CoE parameter recommendations (see SAP note 103747). All other profile parameters should be left to default values.

Name	Value	Description
abap/arfcrstate_col_delete	x	Activates deletion of *ARFCRSTATE* records in background - forces to run report *RSTRFCEU* in batch periodically every 2-5 minutes (see note 539917)
gw/max_conn	2000	Sets max number of active connections (gateway)
gw/max_overflow_size	10000000	Sets size of local memory area for gateway
rdisp/appc_ca_blk_no	2000	Sets tcp/ip communication buffer size
rdisp/force_sched_after_commit	no	Disables automatic rollout of context after commit work
rdisp/max_comm._entries	2000	Sets max number of communication entries
rdisp/no_statistic	<empty>	Enables HTTP statistical records - important for performance analysis
rdisp/tm_max_no	2000	Sets max number of available connections (instance)
rdisp/rfc_max_own_login	90	Sets rfc quota for own logins
rdisp/rfc_max_own_used_wp	90	Sets rfc quota for own used work processes
rdisp/rfc_max_wait_time	5	Sets rfc max wait time after load check
rdisp/wp_ca_blk_no	1000	Sets work process communication block buffer size
ztta/max_memreq_MB	2000	Limit for a single memory allocation request (default 64) - depends on the max. document size

Each WebAS instance should be configured for massive use of parallel rfc (see note 74141). Rfc quotas can also be configured dynamically using report *RSARFCLD*.

Parameters refering to gateway activity may also need to be changed if large numbers of rfcengine threads are used (see note 384971).

4.1.2 Load balancing

For ensuring SMP scalability load balancing is a key feature. XI can make use of load balancing by using multiple WebAS instances and/or J2EE Engines.

4.1.2.1 HTTP inbound

The number of ICM processes (and therefore WebAS instances) is important for HTTP request processing. Besides processing HTTP inbound requests and forwarding to their respective handler work process, icman also serves HTTP client outbound requests (i.e. work processes sending HTTP requests using plain HTTP or SOAP outbound adapters).

If using several WebAS instances, there are 3 possible solutions for load balancing of inbound HTTP traffic:

Message server

Every client uses the message server http port for addressing (parameter *ms/http_port*). The disadvantage with this solution is that the message server sends a HTTP redirect to the client. The client then resends the request to the specified server which causes additional network load (especially with large messages).

Web Dispatcher

A Web Dispatcher is able to forward HTTP requests. It also gets current load information from message server for identifying instances with lower load. This causes no additional network load and is the preferred solution if using several WebAS instances.

Manual distribution

In this case load distribution is done by the client application which sends requests directly to different icman processes, e.g. use several file adapters in parallel sending to one specific icman process.

4.1.2.2 RFC inbound

Scenarios using the Rfc protocol (Rfc Adapter, IDoc) profit from RFC load balancing by logon groups which is a standard WebAS functionality. Rfc requests are forwarded by the WebAS Message server which collects information about WebAS instance load.

4.1.2.3 qRfc

The qRfc scheduler implements load distribution by rfc server groups and managing available work process resources (see *SMQR -> Goto -> QRFC Resources*). See SAP note 369007 for a short overview about scheduler configuration and functions.

4.1.2.4 Gateway

A server application can register several times at a WebAS gateway with the same Program ID. (Program ID is configured in *SM59* Rfc destinations - TCP/IP connections). The gateway process then uses request counters and a round-robin algorithm for distributing requests to registered server programs (this is called load distribution by multiple gateway registration).

Mapping service

Each rfcengine thread of a J2EE server registers at a specific WebAS gateway. The mapping service is called by a work process via rfc destination *AI_RUNTIME_JCOSERVER*. This ensures load distribution between rfcengine threads. One important thing to notice is that each J2EE server registers its bulk of rfcengine threads at service startup. This may lead to a sub-optimal distribution if using several J2EE servers.

Example

Gateway Register Table:

20 rfcengine threads of J2EE server 1

20 rfcengine threads of J2EE server 2

The gateway process dispatches the first 20 mapping requests to server 1, the next 20 to server 2. The gateway process doesn't know about the J2EE configuration. It only dispatches requests to registered server applications. To distribute parallel load to several J2EE servers there is only the possibility to lower the number of rfcengine threads per server (depending on the number of parallel mapping requests).

The number of rfcengine threads (number of processes) is configured for each rfc destination within the rfcengine service of a J2EE server (via Visual Administrator). There is also a tool called rfcEngineTool that can change these settings via commandline.

Use of Local Gateway

Another possible distribution option is the use of local gateways for the mapping service instead of one central gateway. To enable the use of an instance's local gateway just delete the gateway option of the rfc destination *AI_RUNTIME_JCOSERVER* in *SM59*. This ensures that more J2EE servers process mapping requests in parallel and prevents overload of the central gateway. Mapping requests are distributed by the local gateway to local J2EE servers.

Rfc Adapter

The Rfc Adapter can also make use of the multiple server registration feature. That means it is possible to set up several Rfc Adapters for inbound or outbound processing. The Rfc Adapter also uses thread pools for communication with XI (inbound, outbound) and with client and server application systems.

4.1.3 qRfc processing

EO scenarios

The number of queues available for qRfc processing determines the degree of parallelism on the Integration Server. The qRFC scheduler algorithm works more efficient with less queues and more entries per queue as one scheduling step processes one queue up to 60 secs (can be configured). Load is distributed by WebAS instances and work processes.

Speed of outbound processing may lead to a higher number of queues. As long as the Integration Engine runs the *CALL_ADAPTER* step, the corresponding outbound queue will be blocked (or even the inbound queue if XI config-parameter *EO_INBOUND_TO_OUTBOUND*=0).

EOIO scenarios

The number of queues is determined by the application in this case. The Queue-ID is used as serialization context. The only way to improve throughput here is to split up requests logically on application level and use different Queue-IDs. Therefore EOIO should only be used if absolutely necessary.

qRfc Scheduler Activation

During activation of the qRfc scheduler (inbound and outbound) the current implementation uses exclusive locks (select for update). Therefore exclusive lockwaits lead to high serialization costs for larger processing volumes. This has to be observed during performance analysis.

4.1.4 XI parameter configuration

Configuration parameters for the Integration Server can be set via *SXMB_ADM*.

Parameter EO_INBOUND_TO_OUTBOUND (category TUNING)

Determines if an outbound queue is used during outbound processing (only EO scenarios). By default this parameter is set to 1 (which means use outbound queue). An outbound queue is automatically used (regardless the actual parameter setting) if there is a message split because of multiple receivers or interfaces (detected during Receiver/Interface determination).

General recommendation is to use outbound queues as this allows to configure parallelism on receiver level. It also enables receiver specific handling of communication failures. Otherwise one failing receiver may also affect processing of requests to other receivers.

On the other hand using outbound queues causes an additional asynchronous step which needs additional cpu resources (about 30%).

Here are some hints for setting *EO_INBOUND_TO_OUTBOUND*.

Set *EO_INBOUND_TO_OUTBOUND*=1 if

- outbound calls take too long (as this blocks processing queues)
- receiver is not fast enough (also prevents receiver overload)
- expensive CBR (content based routing) is used (reason: if an error occurs the whole pipeline must be re-processed)

Set *EO_INBOUND_TO_OUTBOUND*=0 if

- only IDoc/Rfc outbound adapters are used (as tRfc layer is fast and can handle quotas)
- receiver is fast and can't get overloaded
- no expensive CBR (content based routing) is used

Parameters EO_INBOUND_PARALLEL, EO_OUTBOUND_PARALLEL (category TUNING)

Determine the number of inbound/outbound queues (only EO scenarios). The parameter value should correspond to available resources (number of work processes, CPUs).

As a rule of thumb the number of inbound queues (XBTI*) for a EO-scenario (without using out-bound queues) should be set to:

```
rdisp/wp_no_dia * 0.5
```

Parameter CCMS_MONITORING (category MONITOR)

Please denote that CCMS monitoring currently places a performance overhead of about 10-20%.

4.1.5 J2EE Engine configuration

4.1.5.1 Cluster Manager

Within release 6.20 the J2EE engine is able to handle up to 10 nodes in a cluster. If it is necessary to setup additional servers as load increases, independent clusters each having up to 10 nodes should be setup. Each cluster requires separate deployment and administration.

Cluster configuration can be changed by editing parameter ClusterHosts in file:

```
<j2ee-home>/cluster/dispatcher/managers/settings/ClusterManager.properties
```

Setting

```
ClusterHosts=              (empty)

RepeatToConnect=true
```

makes the dispatcher a primary cluster node which runs independently.

Depending on the type of mapping and the load it may make sense to install secondary J2EE servers in an existing cluster. Especially if garbage collection is a critical issue and the use of another VM is appropriate.

A secondary J2EE server should have the following settings in

```
<j2ee-home>/cluster/dispatcher/managers/settings/ClusterManager.properties:
ClusterHosts=localhost\:50020
DependentElement=true
RepeatToConnect=true
```

(Join Port of the primary J2EE server)

This ensures that these cluster elements use InQMy DB-Settings of the primary server (all configuration settings) and only startup after the primary element is up and running. Secondary J2EE nodes should never be started as first node (after dispatcher) as this clears the cluster's deployment information.

4.1.5.2 Memory Manager

J2EE Engine Memory manager should be set to run only rarely. Explicit garbage collection calls by an application can also be switched off using a Java VM parameter.

```
<j2ee-home>/cluster/server/managers/settings/MemoryManager.properties:
MemoryLevels={95, 96, 97, 98, 99}
SleepTimes={320000, 160000, 80000, 40000, 20000, 10000}
```

4.1.5.3 Service Manager

In case there are service timeouts during startup, general service timeout settings should be adjusted.

```
<j2ee-home>/cluster/server/managers/settings/ServiceManager.properties:
AdditionalLoadTimeout=300
CoreLoadTimeout=300
```

4.1.5.4 Thread Manager

Adjusting system and client thread pools (ThreadManager, SystemThreadManager) is not necessary for XI as only the mapping service is relevant for runtime performance. Thread number for the mapping service is controlled via rfcengine's process number setting for Rfc destination *AI_RUNTIME_JCOSERVER*.

4.2 Java VM tuning

Especially for scenarios that use non-trivial mappings Java VM tuning is one of the most crucial tuning steps. The HotSpot VM parameters for the SAP XI J2EE servers have to be set in file `cmdline.properties` of each server. Settings for the Adapter Engine and the Rfc Adapter have to be made in the shell startup file (*run_adapter*) or within Windows registry.

The Java VM up to and including version 1.3.1 does not have parallel garbage collection, so the impact of GC on a multiprocessor system grows relative to an otherwise parallel application. Therefore garbage collection may become a critical issue.

For example, assume an application spending only 1% of the time in GC on a uniprocessor; GC activity will lead to more than 20% loss in throughput at 32 processors. At 10%, not considered an outrageous amount of time in GC in uniprocessor applications, more than 75% of throughput is lost when scaling up.

The size of the VM heap and the sizes of the VM generations determine how often garabage collection occurs and what algorithm will be used. Note that XML parsing and XSLT mapping create large numbers of objects, which can typically be discarded very quickly after a message was processed (at least if there is no object caching). Optimally these objects should be collected during minor collections as this is less expensive.

The mapping service provided by application IntegrationServices on the J2EE engine is called via JCo from a SAP application server using RFC destination *AI_RUNTIME_JCOSERVER*. The J2EE dispatcher, the system and client threads are not relevant for the mapping process. The thread number of the J2EE rfcengine determines the degree of parallelism for the mapping service. With regard to garbage collection it may be important to adjust the number of rfcengine threads. If there is a higher load it may make more sense to use a second J2EE server.

Example parameters for a test configuration:

```
-server
-Xmx2040M
-Xms2040M
-XX:NewSize=256M
-XX:MaxNewSize=256M
-XX:+DisableExplicitGC
-XX:+PrintTenuringDistribution
-verbose:gc
```

Parameter Explanation

`-server`

This runs the VM in server mode, optimized for long-running server applications such as XI. Note that it needs to be the first flag on the command line. The effect of this setting differs between platforms. On the Windows platform this option should not be set.

`-Xmx2040M`

Defines the maximum heap size as set to 2 GB. On Solaris the Java VM could use up to almost 4 GB. Depending on the mapping it is necessary to use large heaps.

`-Xms2040M`

It is reasonable to avoid resizings of the heap by setting this parameter. Of course, using large numbers of VM to achieve scalability with large heaps at the same time requires large amounts of physical memory.

`-XX:NewSize=256M`

Determines the minimum size of the young generation. Larger messages will in generally need larger generation sizes since the number of objects generated per message normally grows with the message size.

`-XX:MaxNewSize=256M`

Same argument here as for the heap size: avoid resizings if possible.

`-XX:+DisableExplicitGC`

Some versions of the SAP J2EE Engine issue explicit GC calls via `System.gc()`. Since it is the general understanding that the VM knows best when it needs a GC, it is generally recommended (also by SAP) to disable explicit GCs.

`-XX:+PrintTenuringDistribution`

This is an interesting output option showing the generations in the survivor spaces during GC. It can be used to get a better understanding of GC behaviour.

`-verbose:gc`

This is the most important analysis option since GC tuning is a crucial step and, at the same time, impossible without this option. The output of this option is placed in the console logs of the SAP J2EE servers.

In real-life scenarios where all kinds of different messages (and different types of mappings) need to be handled you may have to find a compromise as VM settings cannot be changed dynamically at runtime.

Overall GC times for the J2EE application should be well below 5%. For general recommendations on Java VM parameters see SAP note 552522.

4.3 Database tuning

The database is of crucial importance for the overall system performance since XI persists the messages it transports several times.

I/O Distribution

Distributing I/O evenly across available I/O channels (controllers, disks) is a key factor for optimal database configuration. Therefore as many independent channels as possible should be used. Data areas should be separated from log areas (as well as from paging/swap areas).

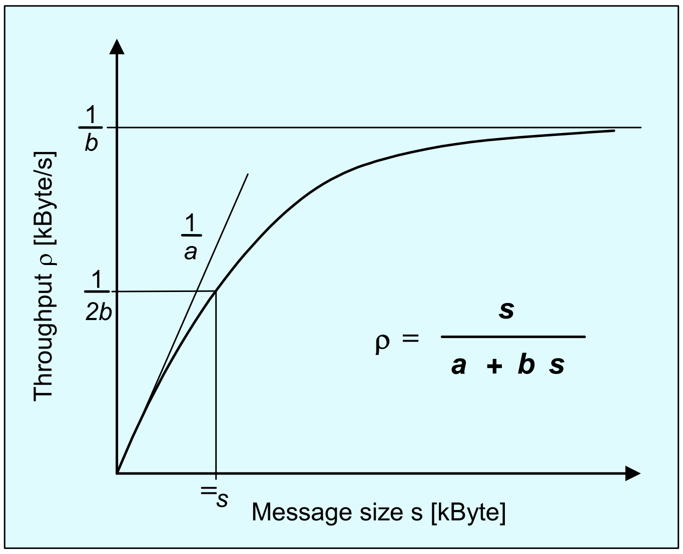
The XI application profile is characterized by

- high-volume processing on a small number of tables (about 10)
- some tables (qRfc) with high insert/delete frequency

4.4 Application tuning

4.4.1 Message size

Throughput in terms of Bytes per second is much higher for larger messages. This is mainly due to processing overhead of a single message (see figure). Assuming a linear model for the cost to process a single message of size s : $\tau = a + bs$, the throughput ρ increases linearly with message size for small messages and asymptotically reaches a maximum for large messages.



Additionally the Intergation Engine stores messages in a compressed format (compression rates for XML messages are normally in the range of 70-80%). Therefore applications should use larger messages by design (if possible).

4.4.2 XI Mappings

XI basically provides two types of mappings:

- XSLT mappings (processed by the inqmy xslt processor)
- Java mappings (Mapping Toolkit or customer development)

Resource consumption for the mapping depends on the complexity of the mappings and the source document sizes. In general XSLT mappings require more memory than mapping classes generated by the XI Mapping Toolkit.

As mappings are processed by the J2EE Engine, the maximum available Java heap may be a limiting factor for the maximum document size the XI mapping service is able to process. During tests it showed that processing of XSLT mappings consumes about the 15-fold of the source document size (using identity mapping). The maximum available Java heap is platform dependent (between 1,6 and 3,8 GB - currently only 32-bit Java VMs are supported). On the Windows platform the maximum document size for mappings using an identity mapping is about 90 MB.

Current max. heap sizes

OS	Max. heap (GB)
Solaris	3,8
HP-UX	3,8
AIX	2,5
Linux	2
Windows	1,6

5 Performance Analysis

5.1 Performance Measurement

Within XI 2.0 there is a performance measurement implemented based on timestamps. This can be activated within *SXMB_ADM* by setting parameter *MEASUREMENT_LEVEL* in category *PERF*. A value greater zero causes the Integration Engine to write a timestamp in the message at the beginning and the end of each processing step. These appear in the header attribute *PerformanceHeader* of an XI message. Additionally the parameter *MEASUREMENT_PERSIST* causes the timestamp data to be persisted in a separate table for evaluation.

Overview of XI 2.0 timestamp locations

Name	Location	Level	Rel.
INTEGRATION_ENGINE_HTTP_ENTRY	CL_XMS_HTTP_HANDLER=>HANDLE_REQUEST	1	SP1
PARSING_HTTP_ENTRY	CL_XMS_HTTP_HANDLER=>HANDLE_REQUEST	2	SP2
HTTP_ADAPTER	CL_HTTP_PLAIN_INBOUND=>HANDLE_REQUEST	1	SP0

IDOC_ADAPTER	IDX_INBOUND_XMB forms idx_inbound_idoc/call_message_broker	1	SP0
RFC_ADAPTER	CL_XMS_RFC_MESSAGE=>CREATE/CALL_XMS	1	SP0
INTEGRATION_ENGINE	CL_XMS_MAIN=>ENTER_XMS/CALL_UC_EXECUTE/ CALL_PIPELINE_SYNC	1	SP1
DB_ENTRY_QUEUING	CL_XMS_MAIN=>CALL_PIPELINE_ASYNC, SXMS_ASYNC_EXEC	1	SP0
DB_SPLITTER_QUEUING	CL_XMS_PLSRV_RECEIVER_SPLIT=>ENTER_PLSRV, SXMS_ASYNC_EXEC	1	SP0
PARSING_MESSAGE_HEADER*	CL_XMS_MESSAGE_HEADER=>readfromxmlsource	2	SP2

<PIPELINE SERVICE>	CL_XMS_MAIN=>CALL_PIPELINE_SYNC,
	CL_XMS_PLSRV_RECEIVER_SPLIT=>ENTER_PLSRV
		1	SP0

PERSIST_INSERT_MESSAGE*	CL_XMS_PERSIST=>INSERT_MSG/ GET_RESPONSE_FOR_MSG/READ_CURRENT_MSG/ READ_MSG/READ_MSG_ALL	2	SP2
PERSIST_ENTRY_BUILD_MESSAGE	CL_XMS_PERSIST=>READ_MSG_ALL	2	SP2
PERSIST_SPLITTER_BUILD_MESSAGE	CL_XMS_PERSIST=>READ_MSG_ALL	2	SP2
PROXY_OUTBOUND*	CL_PROXY_OUTBOUND=>PROCESS_CALL/ PROCESS_CALL_XMS	1	SP2
PROXY_INBOUND*	CL_PROXY_XMS_PLSRV=>ENTER_PLSRV	1	SP2

5.2 Statistical Records

Statistical records can be used for performance evaluation. XI processing generates normally the following statistical record types

- HTTP (for HTTP inbound processing)
- RFC (for asynchronous calls during qRfc processing)

For example one XI message processed in an EO scenario using IENGINE SOAP-HTTP for inbound and outbound processing (without using an outbound queue) generates the following statistical records output:

Task Type	#Steps	Av. Request Time	Av. CPU Time	Av. DB Time
HTTP	1	72	63	9
RFC	3	114,7	57	16

That means there is one HTTP record for inbound processing (in this example running ICF handler class CL_XMS_HTTP_HANDLER) and 3 RFC records for async processing steps. These async steps are characterized by the executed Rfc server function modules:

- QIWK_RUN	(qRfc scheduler activation)
- TRFC_QIN_ACTIVATE	(single qRfc queue activation)
- TRFC_QIN_DEST_SHIP	(single qRfc entry processing)

Function module SXMS_ASYNC_EXEC which actually does the message processing and calls the pipeline is run in step TRFC_QIN_DEST_SHIP.

Following is a single records overview. Depending on the queue fill level steps QIWK_RUN and TRFC_QIN_ACTIVATE are not executed for each incoming message. On the other hand these request types are not very expensive. During load tests a factor of 2,7 between number of messages (HTTP inbound records) and RFC records was reached.

Type	Step	Req.Time	CPUTime	DBTime	Mem.kB	Trans.kB
HTTP	CL_XMS_HTTP_HANDLER	71	63	7	2037	3,5
RFC	QIWK_RUN	41	31	18	2183	7,4
RFC	TRFC_QIN_ACTIVATE	176	16	12	1019	2,4
RFC	TRFC_QIN_DEST_SHIP	148	109	18	3056	66,1

XI Memory usage can also be determined by statistical records. Using SOAP-HTTP processing for inbound and outbound tests showed max. memory consumption per message as upper limit:

3 MB + 7 * xml-document-size

This formula does not consider any mapping.

5.3 CPU Usage Examples

Following are some test results of a scenario using SOAP-HTTP inbound and outbound processing. The message size is 4kB. No outbound queue was used.

Test results for single message processing based on stat records (30 msgs; time in ms):

Task Type	#Steps	CPU Time	DB Time
HTTP	30	1929	252
RFC	90	4671	1413
Overall	30	6600	1665
Single	1	220	55,5

DB request time on application level is about 20% of total request time.

Performance analysis using operating system tools for single message processing:

Process Type	#Msg	Process Time [ms]
disp+work		7094
icman		204
gwrd		234
kernel (DB)		953
Overall	30	8485
Single	1	282,8

In a mass load test CPU time for a single request increased to 240 ms (based on stat records). DB request times collected by stat records for a mass scenario are not that relevant, as there is an unspecified amount of wait time included. Process time measured using OS tools increased to 261 ms for WebAS processes and 35 ms for the database kernel (DB process time about 12% of total time).

6 Appendix

6.1 FAQs

Q : what to do if rfcengine reports error

```
Connect to SAP gateway failed ...
LOCATION    CPIC (TCP/IP) on local host
ERROR       max no of 100 conversations exceeded
```

A : set environment variable *CPIC_MAX_CONV* according to SAP note 314530 or reduce number of rfcengine threads for the J2EE server

Q : what to do if rfcengine reports error

```
Connect to SAP gateway failed ...
LOCATION    CPIC (TCP/IP) on local host
ERROR       max no of conversations exceeded
```

A : adjust SAP profile parameters *gw/max_sys*, *gw/max_conn* for the respective WebAS instance or reduce number of rfcengine threads for the J2EE server

6.2 References

[Sun2003] Performance Documentation for the Java HotSpot Virtual Machine.
Sun Microsystems.

URL: http://java.sun.com/docs/hotspot/