SOFTWARE AG PRODUCTS TUNING GUIDELINES FOR PERFORMANCE

Performance Team

TABLE OF CONTENTS

1. Introduction

This paper presents guidelines for various webMethods and IoT analytics product's configurations which could be followed to exercise optimum and acceptable performance SLAs when working with webMethods products.

Integration Server:

Integration Server is the core application server of webMethods and is used in enterprise level integrations. Adopting best practices when building the solution helps to avoid redundancy and does not limit the solutions to scale on ever growing business needs.

JMS:

Integration Server can connect to webMethods Broker, Universal Messaging or other JMS Providers using JMS Connection Aliases, JMS Triggers and a number of built-in JMS services for sending and receiving messages.

The documents does not factors in the general JMS provider specific attributes like persistent messages non-persistent impact on throughput. Trivially, persistent throughput messages will be limited by the storage speed. While throughput of non-persistent messages will be limited by memory capacity and network bandwidth. Further, JMS provider specific attributes which would impact performance is beyond the scope of this document.

MWS:

My webMethods Server (MWS) provides a user interface to webMethods products such as Optimize, Broker, Process Monitor, and Task Engine. MWS is also a container for Task Engine servlets and custom portlets; it uses an embedded Jetty server as its servlets environment and is capable of very high performance.

MWS instance runs as a multi-threaded process within a single Java Virtual Machine (JVM) which relies on operating system resources.

The users communicate with MWS through a web (HTTP) interface, whereas MWS accesses back end resources through TCP/IP sockets. MWS also requires the services of a database in which state and configuration information is stored. MWS communicates with other applications, such as Optimize and Integration Server, through web services over HTTP.

Universal Messaging (UM)

This document provides basic guidelines for using Universal Messaging-Integration Server publish scenarios in a production environment.

2. Disclaimer

This document lists just the general guidelines which may help in improving the performance of a typical webMethods installation. These guidelines may degrade performance in different environments and use cases. These guidelines can be used on trial and error basis only. Please contact Performance Team in Software AG for suggestions, help and clarifications.

3. Integration Server

3.1. Primary optimization

Optimization and review of the Flow and Java related services are important while implementing the processing algorithm in Integration Server.

3.1.1. Flow service optimization

These are some high-level considerations:

- Stabilization of any performance test should be conducted early
- When measuring performance; predictable, consistent and repeatable results are necessary
- Validation of any infrastructure in use and elimination of any variations between environments should be performed
- Any performance evaluation must include a well-defined objective
- Goals may target highest throughput, ability to scale, and reduced latency
- The requirements for a solution will often dictate acceptable settings (or rule out certain settings) for transaction handling, acknowledgement modes, and latency
- Testing of both happy path and failure processing scenarios should be conducted
- The performance tests must either include failure handling capabilities or the requirements must specifically exclude them
- The system performance often degenerates when failures occur and this should be measured
- Tune JMS trigger settings to support appropriate concurrency for the solution
- Concurrency is affected by threads, pools, queues used and number of Integration Server instances
- Concurrency normally has to increase to reduce latency
- Increasing concurrency without consideration or limit will often result in reduced performance or throughput
- Tune the processing implementation (that is service implementations) to reduce latency
- Increased latency will drive concurrency requirements up for the same target throughput
- Measurements should be made to assess latency at a high level before homing in on specific areas for incremental improvements
- Tuning of the implemented code is far more effective as a first approach than expending a lot of effort on operating system or JVM tuning
- Sending a batch of JMS messages
- The JMS specification v1.1 does not have ‘batch on send’ concept
- Apply appropriate design considerations for use of transactions
- Fully test XA transaction performance, that is use more than one separate XA resource
- All usual best practices regarding implementation of performance testing applies as well

Sl. No	Parameter Name	Default Value	New Value	Comments
1.	Pipeline value	Not dropped	Needs to be dropped until required	**If there is large pipeline data then it can increase network traffic**
2.	Document validation	pub.schema:validate	Should be performed only once	**validate-input or validate-output options should be checked only once**
3.	service caching	False	True	**This indicates the number of messages that can processed at a time**
4.	**Stateless**	**False**	**True**	**This reduces memory consumption as fewer sessions are kept on the server**

3.2. Tuning guidelines in Integration Server

3.2.1. Logging

I/O is many times more expensive than CPU/ memory operations. Keeping a log of key documents for auditing and recovery purposes can have a significant effect on performance. Logging adversely impacts performance because logging threads must be synchronized.

Based on the solution and business requirement, components which should be enabled must be configured to use database as destination and logging mode should be synchronous. This is based on the current design of Integration Server.

3.2.2. Thread pool

Integration Server maintains a variety of thread pools. The most important of these pools is the service thread pool. All client requests to invoke services are processed using threads from the service thread pool. If Integration Server does not have enough threads in the service thread pool, it will block the execution of a request. For instance, if 100 clients make a request to the Integration Server and the Integration Sever has only 75 available threads in the thread pool, 75 of the requests will execute immediately. The remaining 25 requests will wait until threads become available in the service thread pool.

In some cases, larger thread pools may increase throughput. However, larger thread pools do not always translate into higher throughput. When the Integration Server is permitted to execute a large number of threads concurrently, context switching can exert a noticeable effect on performance. The only way to determine the appropriate size of the thread pool is through testing. To identify the optimal pool size, increase the number of threads in the pool and measure the performance under each size. (Note that every time you allocate more threads to the server thread pool, you should also increase the heap size, as every thread needs some amount of memory for itself).

3.2.3. Heap size

Default heapsize may prove to be too small for enterprise level solution expected to scale over time. It is always recommended to set minimum and maximum value to equal value. Optimum value for the heap should be derived from performance tests, analyzing the memory utilization pattern and GC log. To begin with, 4GB minimum and maximum should be good.

3.2.4. JDBC pools

3.2.4.1. MaxPooledStatements

Setting the MaxPooledStatements connection option enables statement pooling. Enabling statement pooling allows the driver to reuse Prepared Statement objects. When Prepared Statements are closed, they are returned to the pool instead of being freed and the next Prepared Statement with the same SQL statement is retrieved from the pool rather than being instantiated and prepared against the server.

3.2.4.2. JDBC connections and login timeout

It is really important to have separate thread pool definitions for each of the components. For better performance minimum value should be set to some number and not zero.

To prevent Login Time out related problem which occurs because of multiple reasons, it is recommended to set LoginTimeout, if login time out is observed frequently.

LoginTimeout, in seconds, is the time driver that waits for connections to be established before returning the control to the application and throwing timeout exception. Default is 0, which means driver does not time out for a connection request.

Sample URL-

DB URL: jdbc:wm:oracle://DBhost:<Port>;serviceName=wmschema;LoginTimeout=120

To prevent this problem, Login Timeout parameter can be appended at the end of your DB URL for the JDBC Pool or JDBC Adapter. The Login Timeout value should be set based on network performance in the landscape.

3.2.5. Adapters

3.2.5.1. JDBC adapter

The Java object generated by webMethods to perform the Stored Procedure call is CallableStatement. The Callable Statements object is created every time the adapter service is invoked. The following properties can be configured to enable caching of the callable or prepared statements.

- ImplicitCachingEnabled=true.
- maxStatements=500.

Minimum and maximum pool size
Minimum and Maximum connections should be tuned such that they are able to serve the concurrent number of incoming request.

Block timeout
This field specifies the number of milliseconds that the Integration Server will wait to obtain a connection with the database before it times out and returns an error. For example, a pool with Maximum Pool Size of

20 is configured. In the event of 30 simultaneous requests for a connection, 10 requests will be waiting for a connection from the pool. If the Block Timeout is configured to 5000, the 10 requests will wait for a connection for 5 seconds before they time out and return an error. If the services using the connections require 10 seconds to complete and return connections to the pool, the pending requests will fail and return an error message stating that no connections are available. Block Timeout value that is too high may encounter problems during error conditions. If a request contains errors that delay the response, other requests will not be sent. This setting should be tuned in conjunction with the Maximum Pool Size to accommodate such bursts in processing.

Expire timeout
If connection pooling is enabled, this field specifies the number of milliseconds that an inactive connection can remain in the pool before it is closed and removed from the pool. The connection pool will remove inactive connections until the number of connections in the pool is equal to the Minimum Pool Size. The inactivity timer for a connection is reset when the connection is used by the adapter.

Setting Expire Timeout value too high may result in a number of unused inactive connections in the pool. This consumes local memory and a connection on your backend resource. This could have an adverse effect if resource has a limited number of connections. If Expire Timeout value is set too low, performance could degrade because of the increased activity of creating and closing connections. This setting should be tuned in conjunction with the Minimum Pool Size to avoid excessive opening/ closing of connections during normal processing.

3.2.5.2. MQ adapter

Wait interval
Wait interval is something like pull interval which listens to the queue after every time out irrespective of whether the queue is empty or full. Once it starts receiving messages, it will keep on pulling messages until the queue becomes empty. Once the queue becomes empty, it will wait for the “waits interval” time out to happen so that it can listen to the queue and check for the messages.

It is always better to have higher Wait Interval (increased wait interval time) when incoming message rate is high and real time. Wait interval can be set to smaller value as well but it consumes lot of CPU cycles when multiple listeners are configured and becomes overhead when number of cores on the box are 2 or less.

Pool size
The minimum number of connection objects that remain in the connection pool at all times. When the adapter creates the pool, it creates this number of connections.

The maximum number of connection objects that can exist in the connection pool. When the connection pool has reached its maximum number of connections, the adapter will reuse any inactive connections in the pool or if all connections are active, it will wait for a connection to become available.

3.2.5.3. SAP adapter

The best performance in the volume tests as well as in the scalability scenarios can be achieved using the following parameters.

watt.PartnerMgr.noMsgStorage
This parameter can be set to "true" or "false". If set to "true", then the message body of the incoming document will not be stored to disk, although a transaction will be created (or maintained) for the incoming document, and the transaction can be monitored later in the transaction list.

watt.PartnerMgr.xtn.store.fastAsyncMode
This parameter can be set to "true" or "false". If set to "true", then the information in the message store will be read or written asynchronously.

watt.PartnerMgr.xtn.store.timeToLive
The parameter can be used to set the "Time-To-Live" threshold value of the asynchronous read/ write cache. It is only effective when "fastAsyncMode" is set to true. The "Time-To-Live" value determines how many seconds a transaction will be kept in the internal cache until it gets purged.

watt.sap.xtn.cacheFlushPeriod
This parameter can be set in the server.cnf to improve the transaction store performance.

4. Process Engine

4.1. Cost of process step

After the business tasks are identified, the technical staff implements it in the process model. Optimal designing of business process is highly important for its performance and throughput (for both Business and BPM software). Research has been conducted to find the cost of a process step in a model. Let us assume a Business Process with 10 steps (excluding the document receive step). Each step is associated with Integration Server flow service which implements some logic.

Baseline numbers were taken with 10 steps process model. Step 2 and 3 were combined as one step and their business logic were also executed sequentially so the functionality is not affected but the business process has lost one step. Now there are 9 steps in the process model performing the same business logic. Throughput number for 9 step process model was recorded. Eventually more steps were combined together and throughput was measured at every step. Refer to the following result table:

Description	Number of steps	Throughput/ sec	% increase from baseline	% increase from previous step
1) 10 step process model with each step mapped to an Integration Server service.	10	65.15	<Baseline>	<Baseline>
2) Reduce the step to 9 by combining step 2 and 3	9	69.44	6.58%	6.58%
3) Reduce the step to 8 by combining step 4 and 5	8	77.52	18.99%	11.64%
4) Reduce the step to 7 by combining step 6 and 7	7	85.84	31.76%	10.73%
5) Reduce the step to 6 by combining step 8 and 9	6	94.79	45.50%	10.43%
6) Reduce the step to 5 by combining step 2,3,4 and 5	5	105.82	62.43%	11.64%
7) Reduce the step to 4 by combining step 6,7,8 and 9	4	119.76	83.82%	13.17%
Average increase per step				10.70%

4.2. Distributed database approach

Distributing Process Engine database is another approach for increasing the performance of BPM. In most of the real time scenarios, the process models are deployed in more than one Integration Server cluster and all Integration Servers point to the same database.

4.3. Effect of correlation

Correlation is an important feature in BPM which helps the incoming documents be processed by the correct instance of a business process.

When correlation is introduced in a process model, extra database calls are included in the execution flow to establish and lookup correlation. There is an additional transition delay of extra receive step which gets added here.

4.4. Impact of joins

AND join is more expensive than any other join types. As the number of steps to join increases, the time taken for AND join also increases.

COMPLEX joins cost depends on how many simple join types are used in it. The cost can be predicted as well. Consider the following two cases:

Split time is less when compared to join time.

There is no difference in performance between using simple AND join and COMPLEX performing AND operation.

Split time is directly proportional to the number of steps it splits into. Only the critical path time is accounted for split time and not all possible paths.

OR and XOR joins has almost the same impact in terms of performance.

OR and XOR joins takes only specific time, no matter how many steps are actually joining. This is because, out of all possible paths only one holds green signal. Thus, may it be 2 steps or 10 steps joining at a step with OR or XOR join type and one step condition is true, then it’s going to take only same time.

4.5. Effect of in-line sub process

Inline sub-process should be used only when it is absolutely necessary. It is not advised to use containers for visual/ UI purpose as it has negative impact on performance.

5. Trading Networks

5.1. Trading Networks performance guidelines

Overall performance for Trading Networks is very tightly bound to the RDBMS performance. In every test case, the database became the bottleneck. Sometimes the disk I/O is maxed out. In other instances, the CPU in the database server becomes the bottleneck. To optimize Trading Networks, focus first on database optimization. For internal Trading Networks document submissions, use wm.tn.doc.route:routeXML instead of wm.tn:receive. This will bypass the user check and is therefore slightly faster.

Within Processing Rules, execute Flow services in asynchronous mode where appropriate or practical.

The property "tn.task.threadpool.pct".if set, it will use that percentage of server threads for Trading Networks Job Manager and this is used for Trading Network outbound delivery of documents and not incoming (both for routing and guaranteed jobs).

There isn't any default value for watt.server.tspace.timeToLive. If a document is written to TSpace, it would get deleted in the following cases:

- If the timeToLive time is over from the creation time of the document
- The top level service has completed its execution

The document will get deleted from TSapce when the next document is created in TSapce. The value shouldn't be too large, or else the files will be in TSapce for a longer time and there may be undesirable results where TSpace gets filled up and no more large docs can be processed. Keep it to optimum value (say around 60 secs or 3 mins) which doesn't affect the processing of document and also there is regular cleanup of TSpace.

5.2. Activity log properties

tn.docType.EnableLogDocumentType: Specifies whether you want Trading Networks to log document type auditing information to the activity log when you manage document types from My webMethods Server.

tn.procRule.EnableLogProcessingRule: Specifies whether you want Trading Networks to log the processing rule auditing information to the activity log when you manage processing rules from my webMethods.

tn.profile.EnableLogProfileChanges: Specifies whether you want Trading Networks to log profile auditing information to the activity log when you manage profiles from My webMethods Server. This property also controls whether you want to log actions related to role-based access.

tn.tpa.EnableLogTPA: Specifies whether you want Trading Networks to log TPA auditing information to the activity log when you manage TPAs.

5.3. Archive properties

tn.archive.archiveAfterDays: Specifies the number of days so that Trading Networks archives documents older than the specified number of days. Specify a number from 0 through 730365. If you omit this property or specify 0 for this property, Trading Networks does not archive documents.

tn.archive.deleteAfterDays: Specifies the number of days so that Trading Networks deletes documents older than the specified number of days. Specify a number from 0 through 730365. If you omit this property or specify 0 for this property, Trading Networks does not delete documents.

5.4. Large document handling properties

tn.BigDocThreshold: The threshold size at which Trading Networks should consider a document to be large.

5.5. Other properties

tn.clean.routePipeline: Removes some of the contents of the pipeline when a document is routed. If set to true, this property removes all data from the pipeline except bizdoc, rule, TN_parms, and $tnReprocess. Setting this property to true can help prevent the pipeline from growing very large, which could affect system performance.

tn.db.fetchMaxRows: Specifies the maximum number of result rows that Trading Networks must retrieve from the database for each query. If you set the value as zero, then all the rows of the query result are retrieved from the database.

tn.receive.clearTNObjects: Specifies whether you want Trading Networks to drop pipeline variables that represent Trading Networks objects. Trading Networks drops the pipeline variables after it completes the execution of the wm.tn:receive entry point service. You specify the pipeline variables to drop using the tn.receive.clearKeys property. Dropping these objects reduces the time it takes to process each document and improves the overall performance of the server because the content handler does not have to format the objects for return to the client. Specify true if you want Trading Networks to drop the objects.

tn.store.pool.stmt: Specifies whether you want Trading Networks to cache SQL statements in memory rather than read them from WmTN/config/dbops.sql. Specify true to have Trading Networks cache SQL statements; specify false to have Trading Networks read SQL statements from a file.

5.6. Database queries properties

tn.query.maxrows: Specifies the default number of rows that the query services return. The query services are services in the wm.tn.query folder. An input variable to the query service allows specifying the maximum number of rows to be returned. If no input is provided to this variable, the query services use the value specified by the service property

6. Java Messaging Server (JMS)

These are general high-level considerations that should always be taken in to account and relate to the design of the overall solution:

- Stabilise any performance test environments early
  - When measuring, predictable, consistent and repeatable results are necessary
  - Run tests for longer and critically assess any unexplained variations observed
  - Finalise and document the JMS Provider's configurations
  - Validate any infrastructure in use and eliminate any variations between environments
- Document the final queue, transaction and processing designs by reference to well-defined requirements
  - Any performance evaluation must include a well-defined objective
  - Goals may target highest throughput, ability to scale, reduced latency and so on
  - The requirements for a solution will often dictate acceptable settings (or rule out certain settings) for transaction handling, acknowledgement modes, latency and so on
- Test both the 'happy path' and failure processing scenarios
  - The performance tests must either include failure handling capabilities or the requirements must specifically exclude them.
  - The system performance will often degenerate when failures occur and this should be measured.
  - Performance testing must include normal running with any safety systems in place (for JMS that might mean using persistent messages, but other impacts such as failure when using batches, or the need for processing to include recovery auditing increasing latency are significant).
- Tune JMS trigger settings to support appropriate concurrency for the solution
  - Concurrency is affected by threads, pools, queues used and number of IS instances
  - Concurrency normally has to increase to reduce latency
  - Increasing concurrency without consideration or limit will often result in reduced performance or throughput.
- Tune the processing implementation (i.e. service implementations) to reduce latency
  - Increased latency will drive concurrency requirements up for the same target throughput
  - Measurements should be made to assess latency at a high level before homing in on specific areas for incremental improvements
  - Tuning of the implemented code is normally far more effective as a first approach than expending a lot of effort on OS or JVM tuning.
- Use transactions to batch sending of JMS messages
  - The JMS specification v1.1 does not have 'batch on send' concept
  - Apply appropriate design considerations for use of transactions
  - Fully test XA transaction performance i.e. use more than one separate XA resource

All usual best practice regarding implementation of performance testing apply as well.

6.1. Transactions and Re-delivery

In JMS, a transaction organizes a message or group of messages into an atomic processing unit. Transactions in Integration server may be used for sending or receiving. When sending, the calls to the JMS send service must be wrapped by a call to startTransaction and a commit. Although there is no general guidance, a starting figure of 10 or 20 messages per batch is often appropriate.

6.1.1. Local or XA Transaction

The JMS interactions follow the JTA specification relating to the use of Local or XA transactions. Only one Local Transaction resource may be enlisted in a transaction context. Therefore, where JMS interactions need to involve more than one JMS Provider resource (varies depending on the JMS Provider used) these will need to be through XA Transaction connections.

Adoption of XA transaction may bring in a considerable and noticeable performance impact on the solution. Nature of the solution and JMS provider specific may impact the performance further. Hence, as part of design decisions it should be considered whether XA is really required as derived from solution requirement.

A viable alternative to XA transactions in many solutions is to be support guaranteed at least once delivery with duplicate detection mechanisms in place. Depending on solution requirements, this is sometimes appropriate, especially if duplicate detection capabilities are required for reasons other than batching.

6.1.2. Multiple JMS Connection Alias

The Integration Servers are often able to generate better throughput if there are multiple JMS Connections to the JMS Provider. This is particularly the case when transactions are being used, because transactions under JMS, limit some of the batching/ threading optimizations possible.

Multiple triggers consuming from or sending to one or more queues/ topics can be configured to use multiple JMS Connection Aliases even if connecting to the same JMS Provider instances. This is relatively trivial to implement because multiple triggers can be easily configured to invoke the same services.

This approach may be useful as it allows a transactional trigger to be used but makes it easy to create concurrency in the solution (simply by copying the triggers and optionally using different JMS connection aliases).

The exact benefit of using multiple JMS Connection Aliases and multiple triggers in this way would need to be investigated for each solution implemented. The same benefits may be achievable by using the trigger thread controls and multiple triggers using a single JMS Connection Alias and that approach should be tried first.

6.1.3. Trigger Concurrency

Throughput requirements are directly driven by the ability to exercise maximum concurrency at the trigger levels. Optimum values for the threads in Integration Server governing the concurrent volumes should be determined by iterative performance test mechanisms.

Higher threads does not necessarily mean higher throughput. Other factors like acknowledgement modes, target service latency invoked by the trigger thread, storage sub-system speed and so on also impact overall concurrency.

At the trigger levels, to maintain the throughput it should be balanced with the message processing overhead specially in cases of batch processing.

Operational laws of performance could be used to get a hint about the concurrency levels

Concurrent threads required =
(Throughput required * Latency of processing of message batch / batch size)

6.1.4. Connections to JMS provider

Triggers in default configuration operate on a single TCP connection to the JMS provider feeding the messages to the available threads. In high performance requirement scenarios, single connection leads to limiting point for concurrency with the property available in Integration Server under JMS connection alias known as “create new connection per trigger“. Desired connections to JMS provider could be established from the trigger properties section.

In the trigger property section through Software AG designer, connection count can be configured. Range is between 1 to 10 connections. It has been observed that starting with 5 to 8 connections is a good option to begin with.

6.1.5. Solution design decision-Processing overhead or Latency

Solution design plays critical role in maintaining the long term scalability goals as per business growth. In JMS based integrations, messages processing algorithm implementation plays vital role in attaining the required throughput. Small reduction in latency of service processing gives substantial gain in overall in the increasing throughput of the solution.

6.2. Limiting factors in JMS scenarios

It is equally important to understand each layer of solution to derive the confidence when optimum configuration from the JMS facilities is reached and when to start looking at other options outside the application boundary for improvements.

- Contention either at the solution level or the resource level degrades throughput. It should be observed that reading and writing to the same queue in real time might bring queue contention into the landscape.
- Unexplained variations in the performance tests run over long duration could hint resource saturations or the external resources like Database related contentions.
- JMS provider may fail to maintain the sustained publish and subscribe rate if there are too many client connections operating concurrently on the single queue.
- High latency requests (or even transient high latency due to system slowdowns) may generate more concurrency resulting in further degradation of the system. Concurrency should be constrained.
- The connection to the JMS Provider will normally be utilizing network bandwidth and this must be sufficient to move messages, payloads and support any overhead.
- Connected or dependent systems may influence throughput in an unpredictable fashion. For example, if production loads on a database vary and sometimes cause reduced throughput on a critical database for the solution (such as when a backup is in process or some large batch processing has started), this would directly affect throughput.
- The JMS Provider must be able to support writing to storage for persistent messages and the storage itself may become a limiting factor.

There are too many potential external factors to mention and others will exist. However, the solution design and the assessment of performance cannot be done without considering the other impacts that increasing threads and concurrency in the Integration Server will have especially on the JMS Provider and on the database in use.

6.3. Tuning Considerations

- **Number of Integration Server instances** – multiple Integration Server instances increases the number of consuming resources
- **Number of JMS Connection Aliases** – particularly when using transactions, the use of multiple separate JMS Connection Aliases allows the Integration Server to generate a more even loading on the system.
- **JMS Transaction Mode** – configured in the JMS Connection Alias, transactions may be used when sending to group multiple messages and reduce the commit load on the JMS Provider
- The desired transaction design will dictate which transaction model is appropriate.
- The use of transactions prevents the JMS Triggers using batches; therefore if transactions are to be used for consuming messages, the mechanism to obtain messages must be re-implemented to use on-demand JMS consumers under the control of an explicit transaction.

- If XA transactions are to be used across multiple Queue Managers, then the impact on throughput needs to be validated.
- **Create New Connection per Trigger** – this is enabled on the JMS Connection Alias and affects the consumption of messages by JMS triggers. It allows Integration Server to create separate groups of connections per trigger in order to increase the ability to deliver JMS messages to service threads.
- Initial recommendation is to enable this option and see the setting on the JMS triggers in Designer.
- **Producer Caching Mode** – this is enabled on the JMS Connection Alias and affects the sending of messages to the Queue Managers. It allows Integration Server to create one or more groups of persistent sessions which reduces the overhead associated with creating sessions when sending messages. The maximum, minimum and timeout values must also be specified.
    - Initial recommendation is to enable this option with at least the default session pool, having a minimum of zero and a maximum equal to the maximum number of threads that may be sending at any given time (typically the maximum number of trigger execution threads). The timeout should be set to 60,000 milliseconds.
    - Using zero as the minimum prevents stale or broken network connections appearing after long uptime periods without affecting average throughput in any significant manner. If the transient (and minimal) latency associated with recreating a new session after the timeout period is not acceptable, then this should be increased.
    - Separate groups of sessions may be allocated to one or more specified JMS destinations in order to be more selective about reserving sessions for sending particular messages. This is configured in the JMS Connection Alias as well.
    - Per destination connection pools may be selected depending on the solution design but there is no general guidance.
- **Number of JMS Triggers** – a single JMS trigger may be duplicated to provide additional bandwidth for consuming messages from a particular source.
    - Initial recommendation is one trigger which may be increased if performance testing shows this to be beneficial.
    - The Connection Count per JMS trigger has an imposed limit of 10 per trigger. Multiple triggers will also increase the ability to generate more than 10 connections if needed.
    - Multiple triggers generate concurrency in the management of connections to the Queue Managers. This is an appropriate way to increase the consuming resources in one Integration Server for one source of messages.
    - Using cut-and-paste is appropriate in Designer for both testing the instantaneous effect of more triggers (pasting an enabled trigger dynamically creates a new trigger in a test or development environment) and for creating multiple copies once the preferred configuration of a trigger has been determined.
- **JMS Trigger properties** – all set in Designer
    - **Processing Mode** – this should be concurrent to permit multiple threads where possible. Serial processing modes are uniquely constrained by processing latency and normally present more specific problems associated with improving efficiency.
    - **Max Execution Threads** – this controls the maximum number of service threads that will be launched by the trigger to process consumed messages. This may only be increased from one if the Processing Mode is concurrent.

        - The starting value should aim to generate the required number of execution threads over all available triggers and Integration Server instances in order to reach the required throughput (based on the latency for processing a single batch)
        - The calculation should be based on the processing latency and desired throughput shown in section **Error! Reference source not found.**, **Error! Reference source not found.**.
    - **Connection Count** – this controls the number of JMS Provider Connections being used to feed messages into the available service threads for the trigger. The value may range from 1 to 10 but cannot exceed the Max Execution Threads.
        - Recommended starting value is 5 but this will need to be modified based on actual performance measures.
- **Max Batch Messages** – this controls the number of messages obtained from the message queue and delivered to each service thread in one invocation for processing.

7. My webMethods Server (MWS)

7.1. General performance Notes

Searches: To optimize search performance, if you are a system administrator: Click Administration > Content > Search Admin, click Tools -> Optimize Indexes to merge all segments in index to improve performance.

CAF Applications: Because CAF applications depend on both the server and client browser, testing with different browsers to identify whether performance bottlenecks are at the server or due to client rendering should be conducted. Firefox generally performs better than Internet Explorer.

Minimize the number of controls per page to decrease render time. Another option is to use hide-able panels and asynchronous controls with both the Lazy Load and Two Pass properties set to true to reduce the number of controls rendered during the initial page load.

Simple controls have little negative impact on performance. Use as many simple controls on a single page as required while conforming to good usability principles helps performance. Simple controls include all read-only controls such as panels.

Loading large numbers of options in list controls increases render time. Generally, loading more than 100 items in simple lists, ‘select <item>’ and swap box controls should be avoided.

Minimize the number of list items displayed per page. Use pagination for tables and limit the page size to no more than 20 rows when possible. Use asynchronous tables for faster rendering. The recommended setting recommends sorting the table entries.

Minimize the size of sortable tables. Ensure that you have no more than 200 table rows when sorting is required. . When the number of table rows must be more than 200, implement sorting in the ISortableTableContentProvider implementation used by the table.

Use Tree controls only when it is really necessary. Using Async Tree with Lazy Load tree provider implementation and expand the tree to the 1st level only by default for faster rendering.

Copy the webm-taglib.tld file to the local package to improve the performance of the web application.

7.2. DB query roles

When Central Users are enabled, all MWS roles and users are visible to Integration Server when end-users access CAF applications and calls into Integration Server services; the user context is used to invoke the services (typically single-sign-on using SAML). Permissions on Integration Server services are configured and Integration Server will attempt to evaluate all ACLs the user and check for user role membership. For every DB Query role, there is a SQL query to be run. Therefore, for more number of roles, more SQL queries are generated. There is a cache for the queried information, but default cache eviction policy will flush the information during every user login. There is a configuration for Role Cache to ensure that the cache invalidation execute happens only when required.

The “config” file is located in the following jar:/common/lib/wm-mws-library.jar$/mws-config/cache.xml

Note: Change "roleCacheLifecycle" attribute of "Role Cache" to value "1" from default "0". If you are not using MWS roles for any permissions check in inside Integration Server, you can also set "defaultCacheTimeout" attribute to "-1" to indicate that the cache never expires.

7.3. MWS DB

Location: <MWS_Installation>\server\<Instance_Name>\config

MaxConnections – (100, 300, 600)

Significance - The max number of connections allowed being open to the database

MinConnections – (30, 60, 120)

Significance – 20% of max connection. Setting this value to too low will result in creating a new connection at runtime instead of using it from a pool of connections. Set the MinConnections to a small non-zero value.

7.4. Web XML

The session-timeout parameter is important in the web XML configuration. Session information is stored in the memory till a user logs out or is logged out automatically due to a time-out. If the session-timeout is long, users consume memory do not relinquish the memory even after moving away from the page, this memory is locked in the tenured space. A few applications can consume significant memory.

The default session-timeout is 30 minutes. Ensure that the value does not exceed a couple of hours. Long timeout values prevent you from knowing if the users are still using the system as their session will still show as active in the session monitor.

7.5. Glue

Glue has a single configuration file that is loaded at startup from your file system or classpath. This configuration file allows you to control settings for the Glue subsystems, such as thread pool sizes and buffer capacity.

threadPoolSize

Significance - When the runtime system needs a thread, it requests the thread pool for a thread to run the task. When the thread finishes the task, it adds itself back to the thread pool to be reused. The pool is initially empty and grows on demand up to a maximum value. Each request needs 1 thread to service the web service call.

maxOutboundKeepAlive

Significance - When a Glue client wants to communicate with a remote server, it requests the outbound connection pool for an HTTP connection to the server. When the request has been sent and the response has been received, the client leaves the connection open and gives it back to the pool to be reused. The pool is initially empty and grows on demand up to a default maximum of 30 outbound connections. When the maximum is reached, the least recently used connection is closed to make room for a new connection.

maxInboundKeepAlive

Significance - When a Glue server receives a connection request from a remote client, it grants the request, creates the inbound HTTP connection, and then allocates a thread from the thread pool to service the connection. If there are no available threads, the request is queued until a thread becomes available. After the request has been serviced, the HTTP server has to decide whether to close the connection or to keep it alive so that it can quickly process additional requests from the same client. By default, up to 50 inbound connections are kept alive at any time. Default value for the maximum number of inbound connections kept alive can be changed.

clientReadTimeout

Significance – Timeout in milliseconds for a client reading a response.

serverReadTimeout

Significance - Timeout in milliseconds for a server reading a request.

Acceptors

Significance - The number of thread dedicated to accepting incoming connections. This should be set to number of cores. It is assumed that you can dedicate half of the available cores for My webMethods Server.

Steps to edit Glue configuration file:

- Open glue-config.xml from following location: <MWS_Installation>\server\<Instance_Name> \deploy\portal.war\WEB-INF
- Edit the file and save.

7.6. Jetty

Jetty is a HTTP open server used within MWS. The Jetty configurations are defined in the Jetty.xml file. It's important not to over-allocate minimum number of threads. Setting the minThreads and maxThreads parameters to a very large number creates a large number of threads consuming resources. Further, garbage collection takes a long time when there are a large number of threads.

minThreads

The minimum number of unused threads to keep within the thread pool. A large number of unused threads will allow the server to respond to sudden increase in load with little latency. If there is an estimation of max and average load, the difference can be in between average and maximum.

maxThreads

Limit to the number of threads that can be allocated to connections for that HTTP listener. This option will limit the number of simultaneous users of the server as well as the maximum memory usage. The primary objective of the maxThread parameter is to protect the server from excess resource utilization from high connection or request rates.

Out of memory

Each accepted connection/ thread consumes memory and unlimited threads will eventually result in an OutOfMemoryException. Note that the memory allocated to the JVM can be increased to avoid this limit,

but at some level physical memory will be exceeded and the server performance will decline. Eventually, virtual memory will be exhausted as well.

Out of threads

Threads are normally implemented by the host operating system and are a finite resource that can be exhausted. The operating system can normally be tuned to increase this limit, but not indefinitely as system performance will eventually degrade.

Out of file descriptors

TCP/IP connections are implemented by most operating systems using file descriptors and are a finite resource that can be exhausted. The operating system can be tuned to increase this limit, but not indefinitely as system performance will eventually degrade.

7.7. SAML

SAML authentication can reduce performance because the following happens for every call between Integration Server and My webMethods Server.

For this example, consider there is on Integration Server instance named IS1 and two My webMethods Server instances named MWS1 and MWS2. When performing a SAML authentication:

- MWS1 initiates a call to IS1
- MWS1 creates a SAML assertion for the current user identity and stores it in memory on MWS1.
- Part of assertion is information pertains to what MWS server assertion was generated. MWS1 in this case.
- MWS1 invokes a web service on IS1, therefore passing this SAML assertion as password for Basic Auth.
- IS1 needs to validate assertion
- IS1 is configured with SAML provider URL, which is in the case of a cluster is a Load Balancer between MWS nodes
- IS1 calls an anonymous web service of SAML provider to validate an assertion it received
- This web service request is routed to MWS1 or MWS2 (round-robin)
- If SAML validate web service request is routed to MWS1, then MWS1 simply checks the presence of a valid assertion it has in its memory list and responds true back to IS1.
- If the SAML validated web service is routed to MWS1, then MWS1 checks the presence of a valid assertion it has in its memory list and responds back true to IS1.The assertion is removed from memory and becomes invalid.

If the SAML validated web service request is routed to MWS2, it does not have this assertion in its memory. MWS2 obtains the originator from the assertion (example, MWS1) and uses MWS Remote Command Invocation (this is an HTTP call) to call directly MWS1 to validate the assertion and then responds back to IS1 with positive or negative result. The parameters mentioned below have to be set when MWS are configured as a cluster. SAML in this use case adds additional overhead to web service

interactions between MWS and IS. This increases the authentication caching used by the Common Directory Services running in IS.

-Dauthenticate.cache.capacity=10000 (Suggested a value to equal number of users that will be able to invoke webService.)

-Dauthenticate.cache.timeout=3600 (The default is 2 minutes, but this info doesn't often change so increase to 1 hour or more).

7.8. Web service and task engine

Web Service processing threads on MWS are used to processing.

-Dmax.webservices.threads=10 (default).

For example, if these are task searches returning large results - the requests consume a large amount of memory.

7.9. LDAP

Set minimum size for the task cache to 10 times the default.

Role cache is set to ensure authentication is made invalid after a specified time, and not to invalidate user logins. The cache.xml file is used for user login invalidation.

7.10. Task deletion

Apart from creating index for T_TASK.PARENT_TASK_ID, you can disable Task Deletion Events by setting the following parameter to true.

Setting -Dtask.delete.event.suppressed to true rescues the number of database calls when deleting the task.

7.11. Other Parameters

7.11.1. Search Tasks

This option specifies the maximum number of concurrent threads allowed to execute task searches. While the default value is high, set this value conservatively to ensure that the maximum number of task search threads and maximum number of task update threads (default 30) is less than the maximum size of the My webMethods Server Data Source JDBC pool (default size is a maximum of 100 connections).

-Dtask.inbox.search.threads may need to be increased depending on your use case. Ensure that you don't set a value that is more than the maximum number of available JDBC connections as it may otherwise cause a deadlock.

7.11.2. Update Threads

This option specifies the maximum number of concurrent threads allowed to run task updates. The threads have to be increased based on the use cases. The number of threads available should be more than the maximum available JDBC connections as it may otherwise cause a deadlock.

-Dtask.update.threads=30

7.11.3. Processing Threads

-Dtask.max.processing.threads default value is 4.

This value is the number of threads that are used to process task events that may be low for real-world applications.

7.11.4. In-Memory event handling

-Dtask.event.lightweight=true|false

For version 8.2 and later, this value is set to true by default. Dtask.event.lightweight enables Task Engine to not use JMS queue for processing task events and all the events are processed on the same JVM instances.

The load on the database is minimized, thus speeding up processing. This parameter should only be used in a single node set-up. If this parameter is enabled in a cluster, there is risk that the same task is updated by multiple users.

8. Universal Messaging and Network Infrastructure

In Universal Messaging clusters, the state of each cluster node is updated regularly. A cluster requires a certain number of cluster nodes to work, to form an active/ active cluster, more than 50% of the servers (a quorum) in the cluster must be active and intercommunicating. Hence, any latency in the network would impact performance of message delivery. Ensuring minimal latency with stable networks and minimal or near zero interrupts would ensure smooth running of Universal Messaging with sustainable performance numbers.

Following are some of the best practices: If possible, firewalls or numerous hops through bridges should be avoided to have a stable network and optimal performance.

- There should be minimum network latency between Client(s) and a Universal Messaging server.
- LAN over WAN is preferred.
- Usage of HTTP/ HTTPS should be avoided for inter-realm communication. Using NIO socket protocols.
- HTTP(S) communication for Integration Server-Universal Messaging should only be used when firewalls are involved.
- In the event of Universal Messaging clustering, a dedicated inter-ream communication interface is preferred.
- Administration interface should also be separated from client - UM communication.
- To have a dedicated interface for communication between client-UM and communication between cluster nodes(inter-realm communication), see the following guidelines:
    - Check "Allow for Inter-Realm", "Enable NIO"
    - Uncheck "Advertise Interface", "Allow Client Communications"
    - Client-UM communication
    - Check "Advertise Interface", "Allow Client Communications", "Enable NIO"
    - Uncheck "Allow for Inter-Realm"
    - Admin Interface communication
    - Check "Allow Client Connections" , "Enable NIO"
    - Uncheck "Allow for Inter-Realm", "Advertise Interface"

8.1. Universal Messaging and SSL

For inter-realm communication, do not enable cert validation because all inter-realm communication goes through "Deffi-Hellman" key exchange at connection creation. For Universal Messaging clients, we recommend enabling certificate validation to use SSL. Unselecting the "Enable Cert Validation" option makes the Universal Messaging clients to not require SSL certificates when connecting to Universal Messaging server instances.

ABOUT SOFTWARE AG

Software AG offers the world's first Digital Business Platform. Recognized as a leader by the industry's top analyst firms, Software AG helps you combine existing systems on premises and in the cloud into a single platform to optimize your business and delight your customers. With Software AG, you can rapidly build and deploy Digital Business Applications to exploit real-time market opportunities. Get maximum value from big data, make better decisions with streaming analytics, achieve more with the Internet of Things, and respond faster to shifting regulations and threats with intelligent governance, risk and compliance. The world's top brands trust Software AG to help them rapidly innovate, differentiate and win in the digital world. Learn more at **www.SoftwareAG.com**.