# MDW Performance Tuning on ServiceMix & Fuse

# TODO:

* DB Cleanup Scheduled Jobs

# Database Connection Pool Size

* Your database connection pool size is determined by the MDW property **mdw.database.poolsize**. In order to arrive at a suitable value for this property, some app-specific things need to be considered.
* First thing to consider is your **mdw.threadpool.max\_threads** setting, since each one of those threads will generally need one DB connection(the exception to this rule would be threads working a processes running at Performance Level 9, see “Performance Levels” section below)
* Second thing to consider is that Service Processes invoked via a web service call, execute within a separate pool (the Jetty thread pool, default pool size is 200), and so those process instances will also require a DB connection.
* Third, you must also consider that Web requests (again, using the Jetty thread pool), will also briefly require a DB connection in order to at least persist the incoming request in the DB, among other things depending on the nature of the request.
* While threads can, and do, share the DB connections in the pool, not having a readily available connection when needed will slow the system down considerably (flows cannot make progress while waiting for a DB connection), and might even result in a deadlock situation (in rare situations) requiring the instance to be bounced.
* Keeping all these things in mind will help you in determining an appropriate **mdw.database.poolsize** for your application. Ideally, you would set this value equal to the size of the jetty thread pool plus MDW thread pool (e.g. Jetty(200) + MDW(400) = 600 DB pool size), but realistically, each connection requires server resources so this might not be feasible considering the number of MDW instances your application has. Just keep in mind that having a max\_threads setting of 500 won’t generally do you much good if you set your database.poolsize to 10, since your 500 threads will have to take turns using the 10 DB connections.
* An additional setting you want to look at is **mdw.database.poolMaxIdle**. You don’t want this set too low based on your load because that will make the system waste a lot of time constantly closing and re-opening DB connections, which takes a lot more time to do than re-using an open connection. A connection is considered “idle” if it is returned to the connection pool and there isn’t a thread already waiting in queue for a connection (i.e. there is NO grace period before the connection is considered to be idle). A value less than 5 would be too low in almost all production scenarios.

# ActiveMQ Broker Configuration

* **\*\*IMPORTANT NOTE\*\*** Please note that all the provided template broker config files and the config parameters defined in these files (e.g. mdw-activemq) are suitable for **development environments ONLY**. It is the responsibility of the various app teams to review these config values and adjust them accordingly for Test, UAT and Production environments.
* For example, when running with ServiceMix/Fuse on a development environment, the cookbook provides a sample ActiveMQ configuration here, [**http://cshare.ad.qintra.com/sites/MDW/Developer%20Resources/Environment/Fuse/ActiveMQ/mdw-activemq-blueprint.xml**](http://cshare.ad.qintra.com/sites/MDW/Developer%20Resources/Environment/Fuse/ActiveMQ/mdw-activemq-blueprint.xml).
* When deploying in a test, UAT, Performance Test or Production environment, the contents of the mdw-activemq-blueprint.xml should be reviewed. A sample for these types of environments is available here: <http://cshare.ad.qintra.com/sites/MDW/Developer%20Resources/Environment/Fuse/ActiveMQ/mdw-activemq-blueprint-redhat.xml>
* At a minimum the following config values should be reviewed before moving to a Test or Production environment:
  + Any specifically defined “memoryLimit” parameters
  + <systemUsage> memory parameters

TODO – add example for Tomcat application-context.xml

# ActiveMQ JMS File Store Corruption Bug and Workaround

* This issue appears to only apply to applications running on ServiceMix. This is an instance-specific problem.
* Suspected cause of this issue is a forced kill (i.e. kill -9) of the ServiceMix server.
* The symptoms of this issue include:
  + The error message “**java.lang.IllegalStateException: PageFile is not loaded**” appears in the karaf.log file for the instance suffering from the corruption.
  + Another clue, although not necessarily proof of the issue, is the error message indicating that “no thread is available” may appear in the log files (karaf.log and/or mdw.log). If you see this error in your logs, you will want to look for the “PageFile” error mentioned above to confirm.
* Once you have confirmed one or more of your ServiceMix instances is suffering from JMS file store corruption, you may take the following steps to resolve the issue.
  + Stop the affected ServiceMix instances.
  + Delete the activemq directory under your {instance\_home}/data directory
  + Restart the ServiceMix instance
  + Once the instance is started, the activemq directory and its contents should be recreated.

# Non-Service Processes

* When a process is launched, MDW puts a message on its internal JMS queue, which uses ActiveMQ with the [VM transport](http://activemq.apache.org/vm-transport-reference.html).  According to the ActiveMQ documentation, "Using the VM transport to connect to an in-JVM broker is the fastest and most efficient transport you can use." To execute the process flow, the MDW JMS Listener receives the launch message from the queue and processes it using the MDW CommonThreadPool, which is built on the [JavaEE Concurrency API](http://docs.oracle.com/javase/6/docs/api/java/util/concurrent/ThreadPoolExecutor.html).
* To avoid exhausting the MDW thread pool, part of your performance tuning exercise should include drastically increasing the pool size from its default value of 10 through the MDW property **mdw.threadpool.max\_threads** (for high-volume systems a reasonable starting number is 100). The tradeoff with this is application memory footprint.  So for maximum throughput the optimal setting for thread pool size is the highest value that avoids excessive memory consumption.
* For versions of MDW prior to 5.5.28 (this includes MDW 5.2 as well), any processing that cannot be done due to thread exhaustion is lost and NOT retried, so it is imperative that the MDW CommonThreadPool is never exhausted. In order to prevent this from happening, it is recommended that in addition to setting the MDW property **mdw.threadpool.max\_threads** to a high enough number to handle your expected volume, you also overwrite the defaults of 2 other properties:
  + **mdw.threadpool.core\_threads** (Default is 5): This property should be set to 1 less than your **mdw.threadpool.max\_threads** property value.
  + **mdw.threadpool.queue\_size** (Default is 20): This property should be set to a value big enough to allow for queuing up ALL possible work requests that your system could possibly encounter (could be in the thousands). For comparison, the Jetty thread pool CTL standard configuration (handles incoming web requests and processing of MDW Service type processes) has a queue size of 6000.
* Setting the core\_threads property to be max\_threads – 1 allows for your system to make use of your desired number of threads if needed without having to first fill up the thread queue. The reason for this is that then you have no safety net if the queue is ever full and you have reached the max\_threads setting. This way, your system will create threads as needed up to your core\_threads set value, and then store additional work requests in the queue (your safety net that under normal circumstances would never be full) to be processed as threads become available. Do NOT set your core\_threads to be the same as your max\_threads in any version of MDW prior to 5.5.28 since this will cause MDW to think you have no threads available or queue space whenever all your threads are busy, even if your queue is empty.
* When the VM is overloaded and there are no available threads for processing, the MDW JMS Listener will wait for a configurable interval before trying to process another message. This interval is controlled by the following property: **mdw.jms.listener.poll.interval**.  However, the best solution is to make sure that the thread pool never gets exhausted in the first place.  Besides scaling up your number of ServiceMix instances, you can introduce a few optimizations through ActiveMQ configuration:
  + Prevent ActiveMQ Request Marshalling - Theoretically the VM transport allows you to avoid the overhead of serializing each request.  However, due to a bug in ActiveMQ 5.5 (https://issues.apache.org/jira/browse/AMQ-3488), MDW forces marshalling by adding "marshal=true" to the default local connector URL.  If you're able to upgrade ActiveMQ to 5.6 then you can override this marshalling by setting the MDW property **mdw.activemq.local.connector.url**.
  + Prevent Creating a Separate Transport Thread - If you don't require Async delivery of your JMS messages you can configure the VM transport to suppress creation of a separate worker thread by adding "async=false" to your connector URL (http://activemq.apache.org/uri-protocols.html).
  + Employ ActiveMQ Thread Pooling - As cited in the e-mail Tommy sent, you can tell ActiveMQ to use an internal thread pool to control dispatching of messages by adding "-Dorg.apache.activemq.UseDedicatedTaskRunner" to ACTIVEMQ\_OPTS.  Also in the link in Tommy's email is mention of the option to set a destination policy on the queue with "optimizedDispatch=true" to avoid spawning a separate session thread for delivering the message to the listener.
  + Enable Async Index Writing in KahaDB - To minimize file I/O bottlenecs with the ActiveMQ persistent store you can set "enableIndexWriteAsync=true" in your broker configuration (http://activemq.apache.org/kahadb.html).
* When JMS messages have been dropped MDW reprocesses

# Service Processes

* If a service process is launched through a Web Service, the initial thread pool for the master process is configured through the Jetty servlet container:

<http://docs.codehaus.org/display/JETTY/Newbie+Guide+to+Jetty#NewbieGuidetoJetty-ThreadPooloptional>

* Here's some information on optimizing Jetty performance (see the section on "Optimizing Threads"):

<http://cshare.ad.qintra.com/sites/MDW/Developer%20Resources/Environment/ServiceMix/JettyOptimizationGuide.html>

* Even within a service process, if an InvokeHeterogeneousProcessActivity is executed with Force Parallel Execution configured, this spawns a separate thread for each subprocess instance using the MDW CommonThreadPool, which uses Java thread pooling:

<http://docs.oracle.com/javase/6/docs/api/java/util/concurrent/ThreadPoolExecutor.html>

The Jetty thread that spawned the Heterogeneous process launch will block until all the Subprocesses are completed.

* Also, here's a good article on Thread Pooling in general (see the section on "Tuning the pool size"):

<http://www.ibm.com/developerworks/library/j-jtp0730/index.html> load does not make sense in this case. The MDW JMS Listener receives messages from the queue and processes them using the MDW CommonThreadPool, which is built on the JavaEE Concurrency API.

# Performance Levels

* 1: All process, activity, transition, and variable instances as well as documents are persisted in database. No in-memory cache is used.
* 3: All process, activity, transition, and variable instances as well as documents are persisted in database, but variable instances and documents are cached for each engine processing session to speed up read accesses.
* 5: This is for service processes only. All process, activity, and transition instances are persisted in database; variable instances and documents are created in memory cache only, not persisted in database.
* 9: All process, activity, transition and variable instances as well as documents are stored in memory cache only. No database persistence is used. For service processes, one memory cache is used for each top level invocation to a service process; for regular processes, a shared memory cache is used for all executions at this performance level.

# Timer Delay Threshold

A very important factor in MDW processing is that **mdw.timer.ThresholdForDelay MUST NOT be set to zero**.  This forces all delays to be scheduled in the database, including the 1-2 second delays that are scheduled internally.  If these rely on database polling to trigger the events then this can be a serious detriment to performance and can lead to backlogs under load.  The minimum value for this threshold should be on the order of 5 minutes.  Some teams have mistakenly set this to zero to enable timer wait functionality, but instead scheduler support should be enabled in ActiveMQ as described in this readme on our SharePoint site: http://cshare.ad.qintra.com/sites/MDW/Developer%20Resources/Environment/ServiceMix/Example%20ActiveMQ%20Config/readme.txt.

**MDW Internal Events Recovery**

The UnscheduledEventsMonitor periodically checks for internal events that have not been processed, and will process those that are older than a designated age.  The maximum number of events that will be processed per cycle are specified in the batch size.  The property names for controlling this behavior are summarized below.

mdw.unscheduled.events.check.delay seconds (default = 90)

mdw.unscheduled.events.check.interval seconds (default = 300)

mdw.unscheduled.events.max.batch.size (default = 1000)

mdw.unscheduled.events.min.age  seconds (default = 3600, min = 300) (**WARNING:** This value should never be set lower than the value specified for **mdw.timer.ThresholdForDelay**. Keep in mind that the former is represented in seconds, while the latter is in minutes. For example, mdw.unscheduled.events.min.age should not be set to 300 if mdw.timer.ThresholdForDelayis set to 60 (3600 seconds))

**NOTE:** For high volume/load applications,it could be beneficial in certain cases for the **mdw.unscheduled.events.max.batch.size** value to be set to a number that is lower than the **mdw.threadpool.max\_threads** value so as to NOT exceed the number of available threads.