Important Information

SOME TIBCO SOFTWARE EMBEDS OR BUNDLES OTHER TIBCO SOFTWARE. USE OF SUCH EMBEDDED OR BUNDLED TIBCO SOFTWARE IS SOLELY TO ENABLE THE FUNCTIONALITY (OR PROVIDE LIMITED ADD-ON FUNCTIONALITY) OF THE LICENSED TIBCO SOFTWARE. THE EMBEDDED OR BUNDLED SOFTWARE IS NOT LICENSED TO BE USED OR ACCESSED BY ANY OTHER TIBCO SOFTWARE OR FOR ANY OTHER PURPOSE.

USE OF TIBCO SOFTWARE AND THIS DOCUMENT IS SUBJECT TO THE TERMS AND CONDITIONS OF A LICENSE AGREEMENT FOUND IN EITHER A SEPARATELY EXECUTED SOFTWARE LICENSE AGREEMENT, OR, IF THERE IS NO SUCH SEPARATE AGREEMENT, THE CLICKWRAP END USER LICENSE AGREEMENT WHICH IS DISPLAYED DURING DOWNLOAD OR INSTALLATION OF THE SOFTWARE (AND WHICH IS DUPLICATED IN THE LICENSE FILE) OR IF THERE IS NO SUCH SOFTWARE LICENSE AGREEMENT OR CLICKWRAP END USER LICENSE AGREEMENT, THE LICENSE(S) LOCATED IN THE “LICENSE” FILE(S) OF THE SOFTWARE. USE OF THIS DOCUMENT IS SUBJECT TO THOSE TERMS AND CONDITIONS, AND YOUR USE HEREOF SHALL CONSTITUTE ACCEPTANCE OF AND AN AGREEMENT TO BE BOUND BY THE SAME.

This document contains confidential information that is subject to U.S. and international copyright laws and treaties. No part of this document may be reproduced in any form without the written authorization of TIBCO Software Inc.

TIBCO, TIBCO ActiveMatrix BusinessWorks, TIBCO Rendezvous, TIBCO Enterprise Message Service, TIBCO Business Studio, TIBCO Enterprise Administrator, TIBCO ActiveSpaces, TIBCO Runtime Agent, TIBCO Designer, and Two-Second Advantage are either registered trademarks or trademarks of TIBCO Software Inc. in the United States and/or other countries.

Enterprise Java Beans (EJB), Java Platform Enterprise Edition (Java EE), Java 2 Platform Enterprise Edition (J2EE), and all Java-based trademarks and logos are trademarks or registered trademarks of Oracle Corporation in the United States and other countries.

All other product and company names and marks mentioned in this document are the property of their respective owners and are mentioned for identification purposes only.

THIS SOFTWARE MAY BE AVAILABLE ON MULTIPLE OPERATING SYSTEMS. HOWEVER, NOT ALL OPERATING SYSTEM PLATFORMS FOR A SPECIFIC SOFTWARE VERSION ARE RELEASED AT THE SAME TIME. SEE THE README FILE FOR THE AVAILABILITY OF THIS SOFTWARE VERSION ON A SPECIFIC OPERATING SYSTEM PLATFORM.

THIS DOCUMENT IS PROVIDED “AS IS” WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NON-INFRINGEMENT.

THIS DOCUMENT COULD INCLUDE TECHNICAL INACCURACIES OR TYPOGRAPHICAL ERRORS. CHANGES ARE PERIODICALLY ADDED TO THE INFORMATION HEREIN; THESE CHANGES WILL BE INCORPORATED IN NEW EDITIONS OF THIS DOCUMENT. TIBCO SOFTWARE INC. MAY MAKE IMPROVEMENTS AND/OR CHANGES IN THE PRODUCT(S) AND/OR THE PROGRAM(S) DESCRIBED IN THIS DOCUMENT AT ANY TIME.

THE CONTENTS OF THIS DOCUMENT MAY BE MODIFIED AND/OR QUALIFIED, DIRECTLY OR INDIRECTLY, BY OTHER DOCUMENTATION WHICH ACCOMPANIES THIS SOFTWARE, INCLUDING BUT NOT LIMITED TO ANY RELEASE NOTES AND “READ ME” FILES.

Copyright © 2001-2016 TIBCO Software Inc. All rights reserved.

TIBCO Software Inc. Confidential Information
Contents

Figures ............................................................................................ 5
TIBCO Documentation and Support Services .......................................................... 6
Overview ............................................................................................. 7
BusinessWorks 6.x Architecture ....................................................................... 8
Performance Benchmark Fundamentals ................................................................. 9
  Interpreting Benchmarks ................................................................................. 10
  Misleading Experiments ................................................................................. 10
  Test Client Limitations ................................................................................. 10
  Points to Remember ...................................................................................... 10
Benchmarking and Testing Performance ............................................................... 12
  Performance Benchmarking Process ............................................................... 12
  Performance Benchmarking Criteria ............................................................... 12
  Performance Testing Tools and Techniques ....................................................... 13
  Collecting Performance Data ......................................................................... 13
  Deploying Performance Testing Framework ..................................................... 13
  Developing Performance Testing Plans ........................................................... 14
  Build a Baseline Test .................................................................................... 14
  Compare Baseline to Targets ........................................................................ 14
  Build Stability Test ....................................................................................... 14
  Develop Incremental Tests ........................................................................... 14
  Develop Peak Rate Tests .............................................................................. 15
  Develop Steady State Tests .......................................................................... 15
  Develop Hardware and Resource Plan ............................................................ 15
  Develop Component Deployment Plan .......................................................... 15
  Monitoring and Analyzing ActiveMatrix BusinessWorks 6.x Components ...... 15
    JVisualVM .................................................................................................. 15
    Monitoring Threads and Taking Thread Dump Using JVisualVM ................. 16
    Understanding Thread Dumps .................................................................... 20
  Identifying Potential Improvement Areas ....................................................... 20
  Implementing Specific Enhancements .............................................................. 20
  Comparing Results ....................................................................................... 21
  Testing and Debugging Performance Issues .................................................... 21
Setting AppNode JVM Parameters ................................................................. 22
  JVM Parameters .......................................................................................... 22
    Heap Space ............................................................................................... 22
    Perm Gen Memory ..................................................................................... 22
    Heap Dump On Out of Memory Error .......................................................... 22
  Hierarchy of TRA files in ActiveMatrix BusinessWorks Home ....................... 22

TIBCO ActiveMatrix BusinessWorks™ Performance Benchmarking and Tuning Guide
# Figures

<table>
<thead>
<tr>
<th>Figure Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Message Flow Architecture</td>
<td>8</td>
</tr>
<tr>
<td>Basic Performance Curve</td>
<td>9</td>
</tr>
<tr>
<td>Hierarchy of TRA files</td>
<td>22</td>
</tr>
<tr>
<td>Scenario (1) HTTP Performance</td>
<td>35</td>
</tr>
<tr>
<td>Scenario (2) SOAP (HTTP) Performance</td>
<td>35</td>
</tr>
<tr>
<td>Performance of HTTP vs HTTPs</td>
<td>37</td>
</tr>
<tr>
<td>Performance of SOAP (HTTP) vs SOAP (HTTPs)</td>
<td>38</td>
</tr>
<tr>
<td>HTTP Connector</td>
<td>39</td>
</tr>
<tr>
<td>HTTP Connector Advanced Configuration</td>
<td>40</td>
</tr>
<tr>
<td>HTTP Client Resource</td>
<td>40</td>
</tr>
<tr>
<td>HTTP Client Thread Pool</td>
<td>41</td>
</tr>
<tr>
<td>JDBC Connection Resource</td>
<td>41</td>
</tr>
<tr>
<td>TCP Connection Resource</td>
<td>42</td>
</tr>
<tr>
<td>JMS Palette</td>
<td>42</td>
</tr>
</tbody>
</table>
TIBCO Documentation and Support Services

Documentation for this and other TIBCO products is available on the TIBCO Documentation site. This site is updated more frequently than any documentation that might be included with the product. To ensure that you are accessing the latest available help topics, please visit:

https://docs.tibco.com

Product-Specific Documentation

Documentation for TIBCO products is not bundled with the software. Instead, it is available on the TIBCO Documentation site. To directly access documentation for this product, double-click the following file:

TIBCO_HOME/release_notes/TIB_BW_version_docinfo.html

The following documents for this product can be found on the TIBCO Documentation site:

- Concepts
- Installation
- Getting Started
- Application Development
- Administration
- Bindings and Palettes Reference
- Samples
- Error Codes
- API Reference
- Migration

How to Contact TIBCO Support

For comments or problems with this manual or the software it addresses, contact TIBCO Support:

- For an overview of TIBCO Support, and information about getting started with TIBCO Support, visit this site:
  http://www.tibco.com/services/support

- If you already have a valid maintenance or support contract, visit this site:
  https://support.tibco.com

  Entry to this site requires a user name and password. If you do not have a user name, you can request one.

How to Join TIBCO Community

TIBCO Community is an online destination for TIBCO customers, partners, and resident experts. It is a place to share and access the collective experience of the TIBCO community. TIBCO Community offers forums, blogs, and access to a variety of resources. To register, go to the following web address:

https://community.tibco.com
Overview

Powered by a next-generation foundation that includes an Eclipse-based design-time, a powerful process engine and a modular OSGI-based run-time, TIBCO ActiveMatrix BusinessWorks™ 6.x enables developers create new services, orchestrate business processes and integrate applications in the shortest time. See, Concepts guide for additional details.

Performance plays a very important role in terms of stability, scalability, throughput, latency and resource utilization. With a view to achieve optimal performance of the ActiveMatrix BusinessWorks application, it is important to understand the various levels at which the tuning methods and best practices can be applied to the components.

The intent of the Performance Tuning Guide is to provide guidelines with respect to performance benchmarking, tuning methodologies and best practices. This document must be used along with other product documentation and project-specific information to achieve the desired performance results. The goal is to assist in tuning and optimizing the runtime for most common scenarios.

This document describes architectural concepts related to performance tuning for ActiveMatrix BusinessWorks 6.x. The document includes the different tuning parameters, steps required to configure the parameters, and design techniques for better performance. However, you must focus on real customer use cases to understand the issue and the associated solution.

The performance tuning and configurations in this document are provided for reference only. They can be reproduced only in the exact environment and under workload conditions that existed when the tests were done. The numbers in the document are based on the tests conducted in the performance lab for ActiveMatrix BusinessWorks 6.3.0 and may vary according to the components installed, the workload, the type and complexity of different scenarios, hardware and software configuration, and so on. The performance tuning and configurations should be used only as a guideline, and after validating the customer requirements and environment. TIBCO does not guarantee their accuracy.
BusinessWorks 6.x Architecture

The most important component in TIBCO ActiveMatrix BusinessWorks is the ActiveMatrix BusinessWorks Engine. The purpose of the engine is to handle a continuous stream of thousands of processes, each with dozens of activities, in an operating environment with finite resources. Resources include the memory, CPU threads, and connections.

The engine also performs the following additional functions:

- XML data transformation and validation
- XPath transitions and flow control
- Connection and session management with recovery and retries
- Engine crash and job recovery
- Exception management and logging
- Management and monitoring services
- Ability to persist or checkpoint and then resume using the checkpoint database
- Fault tolerance and load balancing

Message Flow Architecture
Performance Benchmark Fundamentals

The goal of performance measurement is to understand the performance capabilities and limitations of a system. Every system has limitations, and benchmarks characterize the system in a way that you can understand these limitations.

Benchmarks can be complicated if the system capabilities and limitations vary depending on the demands placed on the system. They also vary based on the resources that are available to the system, for example, CPU, memory, network bandwidth, and disk bandwidth. The set of benchmark measurements must be carefully designed so that the impact of these factors can be understood clearly.

Basic Performance Curve

In the above example, the X axis characterizes the input rate, and the Y axis represents the output rate. The system is exposed to load at a controlled rate and is in a steady-state for some period of time. After an initial stabilization period, both the input and output rates are measured, providing one data point for the performance curve. This example assumes that all other factors are being held constant.

The shape of the performance curve tells us a lot about the system under test. If each input results in a single output, then over the normal operating range, the output rate will exactly match the input rate and within statistical bounds. If each input results in more than one output, or if you are measuring data rates instead of input and output rates, there may be a scale factor relating the two values. However, over the normal operating range there should be a linear relationship between the two rates – that is the curve is a straight line.

The input rate that marks the end of this linear region marks the operating capacity of the system. This may mark the true limits of the system design, or it may indicate that some type of resource limit has been hit. This could be the result of the available physical memory or the bandwidth available on the NIC card or the available CPU cycles getting exhausted. It is important to determine the nature of the limit, as this may indicate ways to alter the environment and increase capacity either by tuning the system or adding resources.
Beyond the operating capacity, further increase in the input rate exceeds the capacity of the system to perform work. Once this occurs, increasing the input rate will no longer produce the same level of increase in the output. The inputs are increasing faster than the system is producing output. If the input rate continues to increase it will reach a point where the output rate begins to decline. The system is taking resources away from completing work and applying them to accepting the inputs.

Operating under load is inherently unstable. Inputs arrive faster than work is being completed, and this results in inputs piling up somewhere. This buffer of memory, disk, messaging system is finite in capacity. At full capacity the system will fail. Thus systems can, at best, operate in the overload region for short periods of time.

Interpreting Benchmarks

Each benchmark measurement provides a single data point on the performance curve. In order to meaningfully interpret that benchmark you need to understand where you are on the curve. Failure to understand your position on the curve can lead to significant misinterpretation of data.

Misleading Experiments

One of the most commonly run performance tests is when a large but fixed number of inputs are applied at the fastest possible rate, often by placing them all in an input queue and then turning the system on. The output rate is often misinterpreted as the system capacity.

However, if you look at the performance curve, it is likely that the system is actually operating far into the overload region with an output rate significantly below the operating capacity. Such tests characterize the system under overload circumstances, but they do not accurately reflect the capabilities of the system. This is especially true when it is possible to further configure or tune the system to limit the input rate so that it cannot exceed the operating capacity.

Another type of testing involves running tests at the low-end of the performance spectrum. While these experiments may be sufficient to establish the slope of the normal operation curve, they give no insight into the actual capacity of the system. They can often lead to false conclusions when comparing designs. Measurements at the low end of the performance curve will show only the increased resource utilization.

Test Client Limitations

When the apparent capacity of a system is reached without having exhausted any of the available resources, it is necessary to also consider whether the limiting factor might be the test client rather than the system under test.

A test client with a limited number of threads may not be capable of providing inputs or receiving outputs at the rate required to drive the system to its full capacity. Ideally, the test client will be configurable through its own parameters. In some cases, it may be necessary to run multiple copies of the test client, each on a different machine, in order to drive the system under test to capacity.

Points to Remember

Keep the following points in mind while performing the benchmarking and tuning exercise.

- Always document the test design in sufficient detail to allow others to accurately reproduce your results.
- Always range demand until the operating capacity of the system under test has been reached. Further increases in input rate do not result in proportional increases in output rate.
- Always document measured or estimated resource availability and consumption.
- Once an apparent operational limit has been reached, investigate to determine whether a true resource constraint has been reached. Consider adding resources such as adding memory, CPU or changing to a higher network bandwidth.
• If an operational limit has been reached without exhausting available resources:
  – Consider whether tuning the system under test might further increase the operational capacity.
  – Consider whether the design or configuration of the test harness might be the true limiting factor in the experiment.
Benchmarking and Testing Performance

This section outlines the steps required to successfully evaluate and tune a TIBCO BusinessWorks 6.x environment.

Performance Benchmarking Process

This document must be used as a general guideline and is not representative of any comprehensive tuning that may need to be done for each use case. Additional or fewer steps may be required, depending on individual factors and contextual requirements. Such tuning will require multiple iterations.

One of the fundamental requirements before performing any kind of tuning exercise is to first carefully eliminate all such external factors that can be a potential cause of any performance issues.

Performance Analysis and Tuning is an iterative process consisting of following:

- Establish performance benchmarking criteria
- Review TIBCO BusinessWorks 6.x performance architecture
- Establish performance best practices guidelines
- Identify and review tuneable parameters for the use case

Performance Benchmarking Criteria

The first step when measuring performance is to identify the Service Level Agreement. Performance targets are determined by user response time and message processing requirements.

Examples of performance requirements include:

- Engine throughput or number of messages processed per second
- Processing speed or average process instance duration and latency
- Web response time. The response and request time.
- Resource utilization
- Concurrent request, sleep time, registered and if applicable, the concurrent users.

Defining the minimum, desired, and peak targets for each requirement helps identifying the type of data to collect and to evaluate the test results.

In addition to these normal load expectations, abnormal error-recovery scenarios under unusually high loads should also be considered. For example, the ActiveMatrix BusinessWorks process might be receiving or polling messages from a TIBCO Enterprise Message Service queue, and the above targets reflect the normal flow of messages through the queue.

However, if communication to the TIBCO Enterprise Message Service server has been disrupted for an extended period of time, or if the ActiveMatrix BusinessWorks Engine shuts down, a much higher load may be experienced when communication is re-established or when the engine restarts.

These scenarios must be addressed when considering the engine performance under load, and to ensure that the throughput does not deteriorate below the target in such situations.

Business requirements also control the decision to use reliable or certified messaging. Certified messaging has an impact on performance.
Performance Testing Tools and Techniques

Once you have established appropriate goals for performance benchmarking, it is necessary to define the performance testing and monitoring framework.

This step significantly varies for each project based on application design and deployment requirements. However, it is important to establish key performance monitoring and benchmark measurement techniques and tools. Monitoring each component requires different techniques. It is important to monitor the application including the CPU, memory and logs, the hardware resources, network and performance metrics using load generation tools.

To monitor application resources like CPU, memory, thread dumps and GC, you can use JVisualVM. More details on JVisualVM and thread dumps are provided in the later sections.

Heap dumps can be generated from JVisualVM and analyzed using memory analyzer tools. For errors during load testing the application logs can be monitored. To monitor OS resources, you can use various OS-dependent tools such as PerfMon, TIBCO Hawk™ and various OS dependent utilities like, Top, prstat, iostat and vmstat. In addition, various systems trace and log files should also be monitored.

Collecting Performance Data

Begin with creating a set of processes for testing purposes. These can be actual processes that will be used in production or more basic processes that represent the production scenarios. The granularity and scope of your performance requirements should determine how processes are used for performance testing.

Configure the operating system tool to measure memory, disk, and CPU usage during all tests. Identify the TIBCO Activematrix BusinessWorks metrics that can measure conformance with requirements. A general strategy is to begin with summary metrics, and then progress to detailed metrics as areas for improvement are identified.

However, if specific performance goals have been defined, you can tailor data collection to provide only required information. Understand where process instance lifetime is spent and collect detailed process instance metrics while processing a few representative process instances. Calculate total throughput, collect summary metrics while generating a typical number of incoming messages of typical size and frequency.

Conduct separate tests for minimum, desired, and peak target numbers. Wherever possible, restrict other local network, operating system, and engine activities during this time. If average metrics are used, restrict the test window to ensure that data collected is relevant.

Deploying Performance Testing Framework

Developing a framework depends on performance goals established earlier and business requirements.

Deploy adequate hardware for running ActiveMatrix BusinessWorks software and testing its performance. Install TIBCO ActiveMatrix BusinessWorks software, along with any optional external software for measuring application performance. Verify that your operating system includes a tool for measuring system resources, such as memory, physical disk, and CPU usage.

Once you have established the key performance matrix and determined the tools and techniques to measure the components, the next step is to establish an appropriate framework for testing performance.

Frequently, a customer would require building a script using third-party performance testing tools such as HP Load Runner, SilkPerformer or JMeter. These tools are frequently used to build extensible framework to invoke HTTP, SOAP, REST and JMS messages.
Developing Performance Testing Plans

Developing a performance testing plan involves building an appropriate set of tests to meet business objectives.

This section provides series of tests that can be planned based on overall objectives.

Build a Baseline Test

For initial performance tests consider deploying the test scenario on a single AppNode.

Test the performance of the AppNode with varying payload and workload depending on the requirement. After basic testing, consider adding more AppNodes. Repeat the tests for testing scalability with added AppNodes.

Collect all the performance metrics during the benchmark tests like CPU, memory, I/O, network, latency, throughput, and GC during the tests.

Perform tests for minimum, desired, and peak numbers, that are identified as required. Capture and store output for the test runs. When the baseline tests are complete, stop the performance data collection utility, stop sending messages, and then stop the AppNode.

Compare Baseline to Targets

Compare baseline test results to performance requirements. If requirements are met, begin testing the next use case. If the requirements are not met, continue with the remaining steps.

Build Stability Test

Frequently, many performance issues can be identified in the stability test, where the application is deployed under lower load conditions, such as five to ten concurrent users with a pre-established think time.

This test focuses on end-to-end successful transactions, and does not aim at measuring the overall response time. Since the test system involves various components, it is important familiarize ourselves with the following:

- Tuning parameter at each component level
- Trace file and techniques to increase trace level
- Log files at each component level
- Error files at each component level
- Monitor database resources, if applicable
- Monitor any incomplete TIBCO BusinessWorks jobs
- Worst performing activities, that is CPU time and Elapsed Time

The test must be completed end-to-end, and the application developer should fix any issues associated with the run. The overall percentage of error and warning should also be noted.

Develop Incremental Tests

Define tests that measure different criteria including error rate in a steady incremental load; for example, 50 users, 100 users, 250 users, and so on.

The user should also define the payload for testing the deployed services. The application should be tested considering the peak load in the production environment.
Develop Peak Rate Tests

Overall business objectives can be different for each project. To measure the system against an extreme number of users and not fail, a peak load test can be designed to determine whether the system can respond to the desired maximum number of users and requests without degradation in response time. This depends on the hardware available in the environment. If the CPU and memory resources are not sufficient, then increasing the numbers will further degrade the performance.

Develop Steady State Tests

This test can be run when the business objective requires providing well established quality of service for the business users, such as the number of concurrent users with the least amount of response time. The steady state keeps steady capacity or steady number of requests per minute, even if the number of concurrent users keep increasing.

This test focuses on maintaining partner quality of service and capacity, and the high number of users.

Develop Hardware and Resource Plan

The choice of a proper OS with the appropriate number of CPUs is one of the most important decisions during this testing phase.

Many operating systems perform significantly different under different types of load. The test plan should consider different operating systems, number of CPUs, and other hardware resources.

Develop Component Deployment Plan

The test plan should extend results obtained in the Develop Hardware and Resource Plan test, and design for the optimal production deployment.

In this phase of the test, optimal production deployment is planned based on test results obtained in the previous test.

This can be achieved by increasing the number of instances of components on a single server or multiple servers, and using different load balancing and fault tolerance techniques to achieve optimal production objectives.

Monitoring and Analyzing ActiveMatrix BusinessWorks 6.x Components

There are various ways and tools to monitor and analyze the ActiveMatrix BusinessWorks components, AppNode and bwagent. Some of them are described in this section.

JVisualVM

JVisualVM, that is shipped with Java SDK is a tool that provides a visual interface for viewing detailed information about ActiveMatrix BusinessWorks applications while they are running on the AppNode JVM.

JVisualVM organizes JVM data that is retrieved by the Java Development Kit (JDK) tools and presents the information in a way that allows data on multiple ActiveMatrix BusinessWorks applications, both local and applications that are running on remote hosts to be quickly viewed. It can monitor both local and remote AppNodes and agents. It can be attached locally by using the PID of the AppNode or agent or remotely by enabling JMX on the JVM.

Users can monitor CPU, memory, classes, threads, monitor thread current state, running, sleeping, wait and monitor. JVisualVM displays thread view in real time.
Monitoring Threads and Taking Thread Dump Using JVisualVM

You can use JVisualVM to monitor threads and take thread dumps for AppNode and bwagent.

Procedure

1. Enable JMX on the AppNode or bwagent by adding the JMX properties in the AppNode or the agent TRA files for remote monitoring.
2. Start `jvisualvm.exe` from `JDK_HOME/version/bin`.
3. Connect to the AppNode or bwagent by using the PID or remotely. To connect remotely, select Remote in the Applications tab and right click to Add Remote Host. Provide the remote host name in the window.

4. Add JMX connection to the remote host as displayed in the figures below.
5. Right click on the remote JMX connection for the AppNode or bwgent and select **Open**.

6. The AppNode or agent CPU, memory, classes and threads can be monitored in the **Monitor** tab. The memory chart also provides the maximum memory settings of the JVM. The user can perform a manual GC and obtain the heap dump too.
7. Thread states can be monitored and the thread dump can be obtained from the **Threads** tab.
8. You can use JVisualVM to configure an option to generate heapdump if the AppNode or bwgent runs out of memory. Right click the JMX connection and select Enable Heap Dump on OOME as show in the figure below.

9. JVisualVM provides CPU and memory profiling capabilities. By default, the profiling tool is not running until you are ready to profile the application. You can choose from the following profiling options:

1. CPU Profiling - Select **CPU Profiling** to profile and analyze the performance of the application in terms of throughput, scalability or response time.

2. Memory Profiling - Select **Memory Profiling** to analyze the memory usage of the application. The results display the objects allocated by the application and the class allocating those objects.

   When you start a profiling session, JVisualVM attaches to the local or remote AppNode or agent and starts collecting profiling data.
When the profiling results are available they are displayed in the **Profiler** tab.

JVisualVM has the following plugins for java implementation:

a. A sampling profiler - Statistical and light weight
b. An instrumental profiler - Heavier

**Understanding Thread Dumps**

Keep in mind the following points while working with thread dumps.

- A thread dump displays the thread name, thread id (tid), which is address of thread structure in memory, id of the native thread (nid) which correlates to the OS thread id, thread priority, state (runnable, waiting, blocked, and so on), source code line and method calls.
- Waiting on monitor and waiting for monitor entry - It is very important to understand the difference between the Waiting on monitor state and waiting for monitor entry state. Waiting on monitor means sleep or wait on an object for a specific period or until notified by another thread. Waiting for monitor means to wait to lock an object since some other thread may be holding the lock, which can happen in a synchronized code.
- **IBM Thread Dump Analyzer** can be used for further analysis. For more details, refer to the link [https://www.ibm.com/developerworks/community/groups/service/html/communityview?communityUuid=2245aa39-fa5c-4475-b891-14c205f7333c](https://www.ibm.com/developerworks/community/groups/service/html/communityview?communityUuid=2245aa39-fa5c-4475-b891-14c205f7333c)

**Identifying Potential Improvement Areas**

If performance requirements are not met and improvement is required, some modifications may be needed.

Modifications may be required in the following areas:

- Adding hardware resources
- Modifying JVM parameters
- Increasing engine threads
- Running multiple AppNodes
- Reducing message size, message burst size or message frequency
- Modifying process design or activity settings
- Tune additional engine parameters and analyze impact
- Tune shared resource level thread pools, connections or any other settings.

Using symptoms specific to your environment, such as memory usage patterns or error messages, you can create a prioritized list of modifications.

**Implementing Specific Enhancements**

Scaling involves adding hardware resources or engines. Tuning involves changes to the AppNode config files or scripts and changes to the process design. When making any type of change, it is crucial to keep other factors stable so that the effect can be correctly measured.

For example, a sample list of modifications might include:

- Allocate additional memory for the engine JVM
- Increase the number of engine threads
- Enable flow control

These changes can be implemented in an incremental way. The reasonable approach is to implement the first modification, and then test to measure the effect of this change. After a satisfactory value is
determined, implement the next modification and measure the effect of the first two changes combined, and so on.

In this example, all the modifications are related to the same resource and memory usage. If the engine requires more memory, then increasing the number of threads in JVM would be ineffective. However, some scenarios might require measuring the effect of each modification separately, by backing out one change before implementing another.

**Comparing Results**

After implementing modifications and repeating the test scenarios compare the adjusted results to the baseline test results.

Exporting performance data to a third-party application to compare test results can simplify this step. If performance improves because of the modifications, compare the adjusted results to the performance requirements. If the requirements are met, begin testing the next use case. If the requirements are not met, repeat the step Develop Incremental Tests and step Develop Peak Rate Tests for additional enhancements.

**Testing and Debugging Performance Issues**

TIBCO Enterprise Administrator™ and TIBCO Hawk can be used to test and debug the performance issues.
Setting AppNode JVM Parameters

This section describes the JVM parameters for the AppNode.

JVM Parameters

This section specifies some of the JVM parameters and their default values for the AppNode.

Heap Space

JVM parameters that can be configured for the AppNode heap are minimum heap space (Xms) and maximum heap space (Xmx).

The default values for these parameters are -Xmx1024m -Xms128m.

Perm Gen Memory

The parameters to set the Perm Gen memory of the AppNode are -XX:PermSize=64m -XX:MaxPermSize=128m.

This parameter setting is applicable only for Java 7.

Heap Dump On Out of Memory Error

The parameter -XX:+HeapDumpOnOutOfMemoryError can be set to enable heap dump when the AppNode runs out of memory.

Hierarchy of TRA files in ActiveMatrix BusinessWorks Home

The hierarchy of the TRA files in the ActiveMatrix BusinessWorks configuration home is illustrated in the figure below:

Hierarchy of TRA files

```
bwcommon.tra

bwappnode.tra

bwappnode<AppName>.tra

bwappnode<AppnodeName>.tra
```
The JVM parameters can be changed in all of the TRA files depending on various factors. The factors are described in the later sections.

The diagram shows the top-down hierarchy from parent TRA to child TRA. Any change in the parent is inherited by its children, and any child can override what its parent has been configured to.

In the TRA files, file 1 is the parent of file 2, and file 2 is the parent of file 3, and file 3 is the parent of file 4. So, applying the precedence rules, if you configure a parameter in file 1 or file 2, that parameter is inherited by all its children unless the children override the configuration.

Setting JVM Parameters for the AppNode Manually

This section explains the location of the TRA files in the ActiveMatrix BusinessWorks home folder structure, the JVM parameter settings and the components that are applicable to these settings.

- The bwcommon.tra file located at `<BW_HOME>/bin/bwcommon.tra` is at the top of the hierarchy. By default, the `java.extended.properties` or JVM properties are not present in this file. However, you can add these properties explicitly and the properties will be applicable to all the AppNodes in all the AppSpaces.

- To set the JVM properties at the AppSpace level for all the AppNodes in a particular AppSpace, the JVM properties can be set in the `bwappnode-<AppSpaceName>.tra` file which is located at `<BW_HOME>/domains/<domainName>/AppSpaces/<AppSpaceName>/bwappnode-<AppSpaceName>.tra`

  This will set the JVM properties for all the AppNodes in that particular AppSpace.

- The user can also set or edit the JVM parameters for a particular AppNode in the `bwappnode-<AppNodeName>.tra` file directly. This file is located at `<BW_HOME>/bw/6.3/domains/<domainName>/AppNodes/<AppSpaceName>/<AppNodeName>/bin`

  This will set the JVM properties only for that particular AppNode.

It is important to understand how JVM parameters at different levels impact the AppNodes in the environment. This has been explained with the help of use cases in the sections below.

UseCase 1

When the user changes the JVM settings in either bwcommon.tra or bwappnode.tra, located in the in the `BW_HOME/bw/6.3/bin` folder the following changes will take place.

Existing AppNodes

- **Before setting the JVM Parameters** - If not specified explicitly, the default values of JVM parameters for each AppNode are applicable. These values are commented by default. These are the settings that an AppNode takes when created, and if the JVM parameters are already set at the AppSpace level, then those settings will take effect, refer UseCase 2,

- **After changing the JVM parameters and no restart of AppNodes** - No changes in the AppNode settings.

- **After changing the JVM parameters and AppNode restart** - The changes will be applied to all the AppNodes in all the AppSpaces in the environment. This change is required to be handled very carefully because, though it allows the user to maintain uniformity across all the AppNodes, if the same settings are not required for all AppNodes this usecase is not very useful.

  For example, if the max heap of an existing AppNode is not explicitly specified, then by default it is 1 GB. However, if the max heap in the files, bwcommon.tra or bwappnode.tra is changed to a higher value at some point of time, there will be a lot of memory allocated to the existing AppNodes when restarted which may not be a requirement for some AppNodes.

  Vice versa, if the memory allocated in the files, bwcommon.tra and bwappnode.tra is changed to a lower value like 512 MB, on restart the existing nodes will have max heap of 512 MB. This may not
be sufficient for an AppNode which has higher memory requirement. This can lead to instability in the environment along with mismanagement of resources especially in terms of memory.

**New AppNodes**

The changes will be applied to any new AppNodes that are created in that particular AppSpace. AppNodes created in any other AppSpace will not have any impact. Any new AppNode created after setting the JVM parameters will contain the new settings. If the memory requirement of the AppNode is not aligned with the settings in bwcommon.tra or bwappnode.tra, it can lead to various issues.

For example, if the max heap setting in the bwcommon.tra or bwappnode.tra is very low, a newly created AppNode may run into issues at startup if it has a higher requirement. Hence setting the JVM parameters in bwcommon.tra and bwappnode.tra is required to be done carefully, taking into account factors like specific AppNode requirements and anticipating the memory requirements of AppNodes to be created in future.

**Usecase 2**

When you are required to maintain the same settings for all the AppNodes in the same AppSpace, the JVM settings can be applied at the AppSpace level, in the bwappnode-<AppSpaceName>.tra file.

On applying the JVM settings, the following changes will take place.

**Existing AppNodes**

- **Before setting the JVM Parameters** - If not specified explicitly, the default values of JVM parameters for each AppNode are applicable. These values are commented by default. These are the settings that an AppNode takes when created, and if the JVM parameters are already set at the bwcommon.tra or bwappnode.tra level then those settings will take effect, refer Usecase 1.

- **After changing the JVM parameters and no restart of AppNodes** - No changes in the AppNode settings.

- **After changing the JVM parameters and AppSpace restart** - The changes will be applied to all the existing AppNodes in that particular AppSpace. The AppNodes in other AppSpaces will not be affected.

**New AppNodes**

The changes will be applied to any new AppNodes that are created in that particular AppSpace. AppNodes created in any other AppSpace will not have any impact.

In this particular use case, it is recommended that since changing the settings at the AppSpace level will affect all the AppNodes in the AppSpace, the requirements for all the AppNodes in that particular AppSpace need to be considered before making these changes.

**Usecase 3**

Changing the JVM settings in the file bwappnode-<AppNodeName>.tra gives you a micro level control over each AppNode.

This use case is useful when the requirements for each AppNode are different. You can manage the AppNodes independent of any changes in the other TRA files. When you want to change the JVM parameters for a particular AppNode in an AppSpace, you can change the properties directly, as mentioned in the section Setting JVM Parameters for the AppNode Manually.

The following changes will take place:

**Existing AppNodes**

- **Before setting the JVM Parameters** - If not specified explicitly, the default values of JVM parameters for each AppNode are applicable. These values are commented by default. These are the
settings that an AppNode takes when created, and if the JVM parameters are already set at the
bwcommon.tra or bwappnode.tra level, refer UseCase 1, or at the AppSpace level, refer UseCase 2,
then those settings will take effect.

- **After changing the JVM parameters and no restart of AppNodes** - No changes in the AppNode
  settings.

- **After changing the JVM parameters and AppNode restart** - The changes will be applied to the
  particular AppNode and there will be no impact on the other AppNodes.

**New AppNodes**

There will be no impact on the AppNodes created.
Best Practices

This section describes some of the most important observations and guidelines developed in field trials. These guidelines are based on numerous performance improvement projects and details of the new features introduced with TIBCO BusinessWorks 6.x.

- TIBCO ActiveMatrix BusinessWorks Engine Tuning Guidelines
- JVM Tuning Guidelines
- TIBCO ActiveMatrix BusinessWorks Transport and Resource Tuning Guidelines
This section provides high-level steps for TIBCO ActiveMatrix BusinessWorks performance tuning. Specific details are discussed in the subsequent sections.

The rule of thumb when tuning the ActiveMatrix BusinessWorks engine is that significant time should be allocated for selecting the JVM configurations and the necessary tuning parameters depending on the use cases, payload and workload requirements.

- Since jobs process on a separate Java thread, the number of engine threads controls how many jobs can run simultaneously. The number of threads that an engine will allocate is set in the AppNode config.ini.

- Measuring the available CPU and memory resources on your system under a typical processing load will help you to determine if the default value of eight threads is appropriate for your environment. For example, if engine throughput has reached a plateau, yet measurements show that CPU and memory are not fully utilized, increasing this value can improve throughput. Typically it is advisable to double the engine threads when tuning the engine if CPU resources are available.

- There are three benefits of CPU utilization:
  - Optimize engine thread.
  - Increase number of AppNodes. Your performance suite must cover different scenarios so that you can come up with the optimal configuration.
  - Tune thread pools, where available, for asynchronous activities.

- Typical numbers of engine threads range between eight and thirty-two. Specifying a value too low can cause lower engine throughput even though spare CPU resources exist. Specifying a value too high can cause CPU thrashing behavior.

- If the pace of the incoming processes still exceeds the number of threads available to run them, consider using flow control.

**ThreadCount (bw.engine.threadcount)**

The threadCount property specifies the number of threads that the ActiveMatrix BusinessWorks engine will allocate. The default value of engine threads is eight.

The jobs in memory are executed by the engine. The number of jobs that can be executed concurrently by the engine is limited to the maximum number of threads, indicated by this property. This property specifies the size of the job thread pool, and is applied to all the AppNodes in the AppSpace if set at the AppSpace level.

Threads execute a finite number of tasks or activities uninterruptedly and then yield to the next job that is ready. The thread count can be tuned to the optimum value by starting with a default value of eight threads and then doubling it up until maximum CPU is reached.

The CPU and memory resources should be measured under a typical processing load to determine if the default threadCount value is suitable for the environment. This value can be set either in the AppNode config.ini file or from the Admin UI as shown below.

- Property = `bw.engine.threadCount`
- Parameter location in the Admin UI = AppNodes > Select AppNode > Configure > General
- Default Value = 8
StepCount (bw.engine.stepcount)

The engine stepCount property determines the number of activities that are executed by an engine thread, without any interruption, before yielding the engine thread to another job that is ready in the job pool.

Exceptions to stepCount can occur when the job in a transaction, is blocked, or is waiting for an asynchronous activity to complete. When a job is in a transaction, the thread is not released until the transaction is complete, even when the stepCount is exceeded.

However, if a job is blocked or waiting for an asynchronous activity to complete, the thread can be yielded even when the stepCount has not been reached.

The default value of this property is -1. When the value is set to -1, the engine can determine the necessary stepCount value. A low stepCount value may degrade engine performance due to frequent thread switches depending on the scenario. Given the nature of the jobs and the number of activities it includes, a high step count value may result in uneven execution of available jobs.

This value can be set either in the AppNode config.ini file or from the Admin UI as shown below:

- Property = `bw.engine.stepCount`
- Parameter location in the Admin UI = AppNodes > Select AppSpace > Configure > General
- Default Value = -1

Flow Limit (bw.application.job.flowlimit)

This property specifies the ActiveMatrix BusinessWorks application's process starters or service bindings flow limit value. There is no default value.

Flow limit is useful when the engine needs to be throttled, as the property specifies the maximum number of jobs that can be started before suspending the process starter. This ensures that the incoming requests do not overwhelm the engine performance and the CPU and memory is preserved.

If the flow limit is set to a particular value at the application level, each of its process starters will contain the same value. For example, if you set the flow limit at application level as eight and the application has two process starters, the flow limit for each of these process starters will be eight.

If the user wants to set the value for one process starter only, they can set flow limit at the component level.

If the number of jobs being created exceeds the flow limit, the engine suspends the creation of new jobs but continues executing the jobs in memory. The engine resumes creating new jobs when sufficient resources are available. There is no default flow limit value and it is not enforced by the engine unless the flow limit property is specified for an application.

This value can be set either in the AppNode config.ini file or the Admin UI. In the Admin UI this property is required to be created as a user defined property.

- Property = `bw.application.job.flowlimit.<UsersBWApplicationName>`
  `[.<UsersBWApplicationVersion>][.<UsersBWComponentName>]`

  The parameters enclosed in square brackets [ ] are optional. For example:
  `[.<UsersBWComponentName>]`

- Parameter location in the Admin UI = AppNodes > Select AppNode > Configure > User Defined > Create
- Default Value = No default value
Page Threshold (bw.application.job.pagethreshold)

This property specifies the maximum number of jobs that can concurrently be loaded into memory.
The page threshold property specifies the job page threshold value for an application’s process starters
or service bindings and is applicable to all the AppNodes in an AppSpace.
It specifies the maximum number of jobs that can concurrently be loaded into memory, thus limiting
the number of running jobs in the memory. Once the threshold is reached, if there are jobs in memory
that are in a state where they could be paged out, the engine temporarily pages out the jobs to the
engine database. These jobs are moved back into memory when sufficient memory is available.
There is no default page threshold value and it is not enforced by the engine unless the page threshold
property is specified for an application.
The page threshold feature requires the engine persistent mode property, bw.engine.persistenceMode
set to datastore or group.
This property specifies the ActiveMatrix BusinessWorks Application’s process starters or servicee
bindings job page threshold value. This value can be set in the AppNode config.ini file.
● Property = bw.application.job.pageThreshold.<UsersBWApplicationName>
  [.<UsersBWApplicationVersion>][.<ComponentName>]
  Parameters enclosed in square brackets [ ] are optional. For example: [<ComponentName>]
● Default Value = No default value

OSGI Bundle Limit

This property specifies a limit on the number of jar files the framework will keep open at one time.
The minimum value allowed is ten. Any value less than ten will disable the bundle file limit, making
the number of jar files the framework keeps open unlimited. By default the value is 100. You can
increase the default value if you expect a lot of file system operations, or have many jar files in the
bundle class path.
This value can be set either in the AppNode config.ini file.

● Property = osgi.bundlefile.limit
● Default Value = 100

Application Statistics

The ActiveMatrix BusinessWorks engine collects three types of statistics for an application, application
job metrics, process statistics and execution statistics.
Enabling statistics adds a certain performance overhead.
You can disable application statistics collection by setting the publisher property
bw.engine.event.publisher.enabled to false in the AppNode config.ini or in the Admin UI.

Application Job Metrics

The application metrics collection or event statistics is enabled by default.
You can disable application metrics collection by setting the properties listed below to false in the
AppNode config.ini or in the Admin UI:

● Subscriber Property = bw.frwk.event.subscriber.metrics.enabled
● Parameter location = AppSpaces > Select AppSpace > Configure > General
● Default Value = true
**Process Statistics**

Process statistics collection for an application or individual processes can be enabled or disabled from the ActiveMatrix BusinessWorks Admin CLI by using the `enablestats` and `disablestats` commands respectively.

For more information on these commands, see the Administration Guide.

**Process Execution Statistics**

Using process execution statistics, you can collect information for individual processes and activities of an application like status, start time, elapsed time and so on.

You can use this information to identify time consuming activities and processes in an application.

The two types of statistics are process instance statistics and activity instance statistics.

**Activity Instance Statistics**

These are the steps to enable activity instance statistics.

**Procedure**

1. Upload and deploy the application to an AppNode.
2. Start the AppNode.
3. From the Admin console cd to the AppNode level and run the following command `enablestats activityinstance <ApplicationName> <ApplicationVersion>`
4. By default the statistics are collected in the CSV file format at `[BW_HOME]/bw/6.3/domains/perf/appnodes/<AppSpaceName>/<AppNodeName>/stats/activitystats.csv`

```
bwadmin[admin@perf/perf/perf]> enablestats activityinstance HTTPServer.application 1.8 TIBCO-BW-ADMIN-380413: Enabled statistics collection for Application [HTTPServer.application:1.0].
```

**Interpreting Activity Instance Statistics**

The following activity instance statistics are collected:

<table>
<thead>
<tr>
<th>Statistic</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Application Name</td>
<td>Name of the application.</td>
</tr>
<tr>
<td>Application Version</td>
<td>Version of the application.</td>
</tr>
<tr>
<td>Module Name</td>
<td>Name of the ActiveMatrix BusinessWorks module.</td>
</tr>
<tr>
<td>Activity Name</td>
<td>Name of the activity.</td>
</tr>
<tr>
<td>Process Name</td>
<td>Name of the process.</td>
</tr>
<tr>
<td>Process Instance ID</td>
<td>Instance ID of the process.</td>
</tr>
<tr>
<td>Statistic</td>
<td>Description</td>
</tr>
<tr>
<td>---------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Start Time</td>
<td>Process instance start time.</td>
</tr>
<tr>
<td>Eval Time</td>
<td>The Eval Time for an activity is the actual time (in milliseconds) required by the activity to complete.</td>
</tr>
<tr>
<td>Elapsed Time</td>
<td>Elapsed time of an activity is the time difference (in milliseconds) between start time and end time of the activity.</td>
</tr>
<tr>
<td>Status</td>
<td>Status of activity, for example: Completed, Faulted or Canceled.</td>
</tr>
</tbody>
</table>

**Process Instance Statistics**

These are the steps to enable process instance statistics.

**Procedure**

1. Upload and deploy the application to an AppNode.
2. Start the AppNode.
3. From the Admin console cd to the AppNode level and run the following command `enablestats processinstance <ApplicationName> <ApplicationVersion>`
4. By default the statistics are collected in the CSV file format at `[BW_HOME]/bw/6.3/domains/perf/appnodes/<AppSpaceName>/<AppNodeName>/stats/processstats.csv`

**Interpreting Activity Instance Statistics**

The following activity instance statistics are collected:

<table>
<thead>
<tr>
<th>Statistic</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Application Name</td>
<td>Name of the application.</td>
</tr>
<tr>
<td>Application Version</td>
<td>Version of the application.</td>
</tr>
<tr>
<td>Module Name</td>
<td>Name of the ActiveMatrix BusinessWorks module.</td>
</tr>
<tr>
<td>Statistic</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Component Process Name</td>
<td>Name of process configured to a component. If the process is a non in-lined sub process, this could be empty.</td>
</tr>
<tr>
<td>Job ID</td>
<td>Job ID of the process.</td>
</tr>
<tr>
<td>Parent Process Name</td>
<td>If the process is an in-lined sub process, the name of the parent process.</td>
</tr>
<tr>
<td>Parent Process ID</td>
<td>If the process is an in-lined sub process, the instance ID of the parent process.</td>
</tr>
<tr>
<td>Process Name</td>
<td>Name of the process.</td>
</tr>
<tr>
<td>Process Instance ID</td>
<td>Instance ID of the process.</td>
</tr>
<tr>
<td>Start Time</td>
<td>Process instance start time.</td>
</tr>
<tr>
<td>End Time</td>
<td>Process instance end time.</td>
</tr>
<tr>
<td>Elapsed Time</td>
<td>Elapsed time for a process is the total time taken by the process, including the elapsed time for all the activities executed for the process.</td>
</tr>
<tr>
<td>Eval Time</td>
<td>The Eval Time for a process instance is the total evaluation time (in milliseconds) for all the activities executed for the process instance.</td>
</tr>
<tr>
<td>Status</td>
<td>Status of process instance, for example: Completed or Faulted.</td>
</tr>
</tbody>
</table>

Since elapsed time is the time taken to execute an activity, including all the delays in acquiring resources like engine threads, JDBC connections, network, and so on, many times a higher elapsed time but acceptable eval time means incorrect engine tuning or a bad design in the process. This would mean the engine is starving for threads. If there are no real activities that are taking high eval times, it is a matter of tuning the ActiveMatrix BusinessWorks engine. Elapsed time will always be equal to, or more than the CPU time.
JVM Tuning Guidelines

Each TIBCO ActiveMatrix BusinessWorks engine runs as a multi-threaded Java server application. Processes and other objects used internally by TIBCO ActiveMatrix BusinessWorks are Java objects that consume memory while the engine is running.

Java provides some useful parameters for tuning memory usage. Ensure that you consider these factors when selecting a JVM.

Besides the JVM version and the vendor, the most relevant tuning parameters are:

- JVM heap size
- Garbage collection settings

Specifying JVM Heap Size

The default Java heap size, which varies according to platform, is a conservative estimate made by the developers of the particular type of Java being used.

When you calculate the amount of memory required for a AppNode, you should determine the largest heap size that can reside in physical memory. For best engine performance, paging to disk should be avoided.

The recommended heap size for a small workload is 1024 MB, for medium 2048MB, and for large workload 4GB or more. The recommendations will change based on the scenarios, payload and workload.

To set the JVM available memory, use the following parameters:

- The Initial JVM Size parameter sets the minimum amount of memory used.
- The maximum JVM Size sets the maximum amount of memory used.

See Setting Appnode JVM Parameters for more details.

It is recommended to set the minimum and maximum JVM size to same value.

The total amount of JVM memory required to operate a ActiveMatrix BusinessWorks engine should be the memory for each process plus the maximum number of processes that can be in the process pool. If flow control is enabled, the process pool can contain up to the Max Jobs value.

JVM Garbage Collection

Tuning garbage collection requires good understanding of garbage collection frequency, message size, and longevity,young, tenured, and perm.

Many JVMs have different levels of algorithms to suit application requirements. Hence, it is very difficult to provide general recommendations.

The option AggressiveHeap affects the memory mapping between the JVM and operating system, and the option ParallelGC allocates multiple threads to part of the garbage collection system.

The client should try to experiment with these options, but the results are not deterministic. It can be argued that a multi-process system of four to twelve CPUs, using ParallelGC can produce better results.

It is recommended that the application should not make direct explicit garbage collection. If the application does make a call to explicit garbage collection, to disable it, use verbose:gc -XX:+DisableExplicitGC. These properties can be appended to the java.extended.properties in the AppNode tra file.
Performance should be one of the criteria for using the appropriate transport. Besides performance, you should also consider interoperability, reliability, and security requirements.

Users have choices of using HTTP or JMS, SOAP over HTTP, SOAP over JMS and TIBCO ActiveEnterprise™ messages. The best practice of selecting a proper standard for the project is beyond the scope of this document. However, the following general results from a performance point of view are provided for your reference only.

You should use them only as general observations, and for initial understanding. Since each user has a different environment and different requirements, these results cannot be directly applied to their work. Instead, it is strongly recommended that you create a realistic scenario in the lab and then derive the appropriate results.

**HTTP Resource**

Performance characteristics of SOAP and REST are closely tied to the performance of the HTTP implementation in TIBCO ActiveMatrix BusinessWorks.

In some situations, you can alter the configuration of the HTTP server that receives incoming HTTP requests for TIBCO ActiveMatrix BusinessWorks. There are two thread pools that can be tuned or set to appropriate values for handling the incoming concurrent requests efficiently.

- **Acceptor Threads** – These are the Jetty server threads. Acceptor threads are HTTP socket threads for an HTTP Connector resource which accepts the incoming HTTP requests.
- **Queued Thread Pool** - The Queued Thread Pool (QTP) uses the default job queue configuration. The QTP threads accept the requests from the Acceptor Threads.

You can configure the following two properties to specify values for the Queued thread pool.

- **Minimum QTP Threads** - The minimum number of QTP threads available for incoming HTTP requests. The HTTP server creates the number of threads specified by this parameter when it starts up. The default value is ten.
- **Maximum QTP Threads** - The maximum number of threads available for incoming HTTP requests. The HTTP server will not create more than the number of threads specified by this parameter. The default value is seventy-five. This limit is useful for determining number of incoming requests that can be processed at a time. Setting a high number will create that many threads and drastically reduce the performance.

It is recommended that if you have a large number of incoming requests, you can change these values to handle more incoming requests concurrently. You can also increase these numbers only if you have a higher peak concurrent request requirement, and you have large enough hardware resources to meet the requests.

See Tuning Parameters for configuration details of these parameters.

**Sample Test**

Scenarios Under Test: Scenario 1 is a simple HTTP Receiver with HTTP Response and echo implementation.

Scenario 2 is a simple SOAP service with echo implementation. The settings when carrying out the tests were as follows:
Parameter | Value
--- | ---
Engine Threads | 32
Event Statistics | Disabled
Payload | 1 KB
Other Settings | Default

The graphs below illustrate the HTTP and SOAP (HTTP) service performance in terms of throughput scalability with increasing concurrent users for the given hardware. See Hardware Configuration for more details.

Scenario (1) HTTP Performance

![HTTP Performance Graph]

Scenario (2) SOAP (HTTP) Performance

![SOAP(HTTPS) Performance Graph]

HTTP Client Resource

There are two important tuning considerations related to the HTTP client:

- HTTP Client Thread Pool
- Connection Pooling
HTTP Client Thread Pool - If you are using HTTP or SOAP with HTTP Send Request or SOAP Invoke (Reference) activity, it is important to verify that the rate of request received on the HTTP server keeps up with the rate at which the client sends messages. This situation is likely to arise when there are very high numbers of concurrent requests being made.

Each Request and Response activity that uses the HTTP protocol, for example, Send HTTP Request or SOAP Invoke is associated with a unique thread pool.

Each request is executed in a separate thread, which belongs to the thread pool associated with the activity. The number of threads in the pool determines the maximum number of concurrent requests a request or response activity can execute. This is a cached thread pool with no default value.

Alternatively, you can create a separate client thread pool and define the core and max pool values. Set the value of this property to a reasonable number for your system. If you set the value too high, it may result in extra resources being allocated that are never used.

You may want to increase the value of the max pool size. However, this value should be increased only if the client side has a lot more backlogs compared to the receive side. It is recommended that you design a test framework that helps you monitor such behavior and determine optimal thread pool count.

Connection Pooling - In the absence of connection pooling, the client makes a call to the same server in a single threaded communication. By enabling connection pooling, multithreaded communication is enabled. Hence, by default the HTTP Client provides single threaded connection. Single threaded connection can be used for multiple sequential requests. However in cases where there are multiple concurrent requests, enabling connection pooling is required.

If connection pooling is enabled, it can improve performance as a pool of reusable shared connections will be maintained. Using the connection pooling feature you can additionally tune the number of connections that the client establishes with the service.

See Tuning Parameters for more details.

JMS Resource and JMS Transport

Explained below are some usage recommendations when using JMS resource or JMS transport.

<table>
<thead>
<tr>
<th>Usage Recommendations</th>
</tr>
</thead>
<tbody>
<tr>
<td>Transport such as JMS can be throttled by limiting the number of sessions. JMS does not deliver more messages until some of the sessions have been acknowledged.</td>
</tr>
<tr>
<td>The combination of TIBCO Enterprise Message Service features Explicit Acknowledge and FlowLimit also exists. In this case, location of ack in process makes no difference.</td>
</tr>
<tr>
<td>When using Client Ack, the JMS session cannot receive a new message until the current message is acknowledged.</td>
</tr>
<tr>
<td>Using TIBCO ActiveMatrix BusinessWorks you can configure multiple sessions to receive messages faster, and set number of sessions higher than the number of engine threads.</td>
</tr>
<tr>
<td>Acknowledge and confirm messages as soon as possible, to improve throughput.</td>
</tr>
<tr>
<td>By holding Client ack to the end of the process, you will block that session. This means you will slow down the rate at which TIBCO ActiveMatrix BusinessWorks pulls messages from the JMS server, which will have to hold messages for a longer period of time.</td>
</tr>
</tbody>
</table>
**Usage Recommendations**

With TIBCO Enterprise Message Service Explicit Ack, a single session is used to receive all messages. This mode allows for more efficient resource utilization, and provides even load distribution across multiple engines.

The best way to increase performance beyond the capability of a single engine is to distribute the load over multiple engines using a load-balanced transport such as JMS Queue, to distribute the work. External mechanisms exist to allow HTTP to be used for this purpose also.

Simple and Text JMS message types have the lowest processing overhead.

To design a long running process to fetch a message and process it, use **Get JMS Queue** message activity in a loop instead of **Wait For JMS Queue** message. In most cases, a JMS starter will be sufficient in this scenario.

If possible, choose NON_PERSISTENT as the delivery mode in replying to a JMS message.

---

**Impact of SSL on Performance**

When using HTTP over SSL, the overhead added by SSL on the overall request and response or throughput for a scenario is limited.

The graphs below illustrate the maximum throughput performance for scenarios with and without SSL for the given hardware. See Hardware Configuration for more details.

See HTTP Connector Resource for scenario and settings details.

**Performance of HTTP vs HTTPs**

![Performance of HTTP vs HTTPS](image)
Performance of SOAP (HTTP) vs SOAP (HTTPS)

![Graph showing comparison of SOAP (HTTP) vs SOAP (HTTPS) performance with throughput in requests per second.]

TIBCO ActiveMatrix BusinessWorks™ Performance Benchmarking and Tuning Guide
Tuning Parameters

This section explains the tuning parameters for the following connectors, connection resources and messaging servers:

- HTTP Connector Resource
- HTTP Client Resource
- JDBC Connection Resource
- TCP Connection Resource
- JMS Receiver, Get JMS Queue Message, Wait for JMS Request

HTTP Connector Resource

This section describes the tuning parameters for the **HTTP Connector** resource.

**Acceptor Threads**

These are the HTTP socket acceptor threads for the **HTTP Connector** resource, which pick up the HTTP requests. The default value is 1.

The value for these threads can be modified at design time in Business Studio as shown below:

Parameter location = **HTTP Connector** resource > Basic Configuration

Accept Queue Size

This is the number of connection requests to be queued before the operating system starts sending rejections. The value can be set to either 0 or -1. These values signify that the queue size is fifty or OS-specific.

These values can be modified at design time in Business Studio as shown in the figure above.

Parameter location = **HTTP Connector** resource > Basic Configuration

Queued Thread Pool

The Queued Thread Pool is the thread pool provided by Jetty that uses the default job queue configuration. The QTP threads accept requests from the Acceptor threads. The value for these threads can be modified at design time in Business Studio as shown below.

Parameter location = **HTTP Connector** resource > Advanced Configuration

The default values are:
- Minimum QTP threads = 10
- Maximum QTP threads = 75

**HTTP Connector Advanced Configuration**

| Parameter                  | Value   
|---------------------------|---------
| Header Buffer Size (B)    | 4096    
| Request Buffer Size (B)   | 8192    
| Response Buffer Size (B)  | 24576   
| Max Idle Time (ms)        | 200000  
| Low Resource/Max Idle Time (ms) | 0       
| Linger Time (ms)          | 0       
| Max Post Size             | 2097152 
| Max Save Post Size        | 4096    
| Minimum QTP Threads       | 10      

**HTTP Client Resource**

This section describes the tuning parameters for the **HTTP Client** resource.

**Maximum Total Connections**

This is the maximum number of concurrent HTTP connections allowed by the resource instance to be opened with the target service. This property is enabled only if connection pooling is enabled, that is the disable connection pooling parameter is unchecked.

For applications that create many long lived connections, increase the value of this parameter. The value for these threads can be modified at design time Business Studio as shown below:

Parameter location = **HTTP Client resource > HTTP Client**

Default value = 200

**Maximum Total Connections Per Host or Route**

This is the maximum number of concurrent HTTP connections allowed by the resource instance to be opened with the target service to the same host or route. This property is enabled only if connection pooling is enabled, that is the disable connection parameter is unchecked.

This value cannot be more than the maximum total connections. The value for these threads can be modified at design time in Business Studio as shown in the snapshot below. Every connection created here also counts into Maximum Total Connections.

Parameter location = **HTTP Client resource > HTTP Client**

Default value = 20

**HTTP Client Resource**

**Thread Pool Resource**
The user can optionally create this client thread pool which routes the messages to the target service. The thread pool resource can be created by either selecting a thread pool resource template or creating a new one as shown in the figure above.

The values for these threads can be modified at design time in Business Studio as shown below:

Parameter location = **Thread Pool Resource Template** > **Thread Pool**

The default values are:

- Core Pool Size = 5
- Max pool Size = 10

**HTTP Client Thread Pool**

![Thread Pool Resource](image)

**JDBC Connection Resource**

This section describes the tuning parameters for the **JDBC Connection** resource.

**Max Connections**

This parameter specifies the maximum number of database connections to allocate. The value can be modified at design time in Business Studio as shown below:

Parameter location = **JDBC Connection resource** > **JDBC Connection**

Default value = 10

**JDBC Connection Resource**

<table>
<thead>
<tr>
<th>JDBC Connection</th>
<th>Connection Type</th>
<th>Maximum Connections</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Direct</td>
<td>10</td>
</tr>
</tbody>
</table>

**TCP Connection Resource**

This section describes the tuning parameters for the **TCP Connection** resource.

**Maximum Connections**

This is the maximum number of simultaneous client sessions that can connect with the server. This parameter is enabled only if connection pooling is enabled. The value can be modified at design time in Business Studio as shown below:

Parameter location = **TCP Connection resource** > **TCP Connection**

Default value = 10

**Maximum Wait Time**

This is the maximum wait time in milliseconds to connect to the TCP server. This parameter is enabled only if connection pooling is enabled. The value can be modified at design time in Business Studio as shown below:

Parameter location = **TCP Connection resource** > **TCP Connection**

Default value = 10000 ms
TCP Connection Resource

Max Sessions [Client ACK Mode]

This is the maximum number of client sessions that can connect with the messaging server. This property is enabled only when the Client ACK mode is used. The value can be modified at design time in Business Studio as shown below:

Parameter location = JMS Activity > General
Default value = 1

JMS Receiver, Get JMS Queue Message and Wait for JMS Request

This section describes the tuning parameters for the JMS messaging servers.

Max Sessions [Client ACK Mode]

This is the maximum number of client sessions that can connect with the messaging server. This property is enabled only when the Client ACK mode is used. The value can be modified at design time in Business Studio as shown below:

Parameter location = JMS Activity > General
Default value = 1

JMS Palette
Performance Improvement Use Cases

Given the wide range of use cases and even more complex and demanding scenarios that the platform can address, the default configuration of ActiveMatrix BusinessWorks might require some adjustments to reach optimal performance.

Therefore ActiveMatrix BusinessWorks, as any other Enterprise-grade product requires tuning along with performance and load testing to achieve best results. The use cases discussed below are based on the tuning experience for large production implementations and POC scenarios.

Throughput Improvement for SOAP and REST Services

**Use Case**: The focus of the project was related to scalability and maximizing throughput for a given workload for SOAP and REST services deployed on single AppNode.

The aim was achieved by means of tuning certain ActiveMatrix BusinessWorks parameters. These tests were conducted on Amazon EC2 servers with the following configuration:

8 vCPUs - Intel Xeon E5-2680 v2 @ 2.80 GHz (25 MB Cache)

**Testing and Measurement**: The main testing was focused on the below aspects:

- Tests were conducted with the default engine and other parameters for a given workload.
- Throughput, that is transactions per second, latency and CPU utilization was measured.

**Solution For Scalability and Improving Throughput**: The solution for scaling the engine was as follows:

- Tuned the ActiveMatrix BusinessWorks engine thread count. Initially tested with default eight engine threads and then doubled the threads depending on the CPU utilization.
- Tuned the **HTTP Client** max total connections and max connections/host depending on the calls to the external services deployed and the engine thread value.
- Tuned the **HTTP Client** thread pool depending on the calls to external services deployed and the **HTTP Client** max total connections value.

**Results**

An increase in throughput (tps) and reduction in latency was achieved by following a step by step tuning approach as illustrated in the graphs below. The first graph illustrates the performance achieved by tuning the engine threads alone.

The second graph shows further improvement in performance after tuning the **HTTP Client** connections and threads.
Throughput and Scalability Improvement for SOAP and SOAP with SSL Services

**Use Case**
Given a ActiveMatrix BusinessWorks project which comprises mainly of SOAP services. The aim was to benchmark and achieve improvement in terms of throughput, scalability with maximum resource utilization. See [Hardware Configuration](#) for hardware details.

**Testing and Measurement**
The main testing was focused on the aspects listed below:

- The benchmarking process focuses on maximizing CPU and memory utilization. Benchmarking included executing load tests for a given workload.
- Total end-to-end business process completed per second was calculated

**Solution**

- The event statistics or application metrics feature is known to add some overhead in terms performance, hence the event statistics were disabled which improved the performance by about 30% to 40%. See [Application Statistics](#) for more details.
# Hardware Configuration

The hardware configuration used for tests conducted in the performance lab to establish benchmark results.

<table>
<thead>
<tr>
<th></th>
<th>Machine 1 (BW AppNode)</th>
<th>Machine 2 (Load Generator)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Server Class</td>
<td>Proliant HP BL460c G1 (C-Class)</td>
<td>Proliant HP BL460c G1 (C-Class)</td>
</tr>
<tr>
<td>Chip Vendor</td>
<td>Intel</td>
<td>Intel</td>
</tr>
<tr>
<td>CPU Model</td>
<td>Intel XEON</td>
<td>Intel XEON</td>
</tr>
<tr>
<td>Total Number Of Processors</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>Cores Per Processor</td>
<td>6</td>
<td>6</td>
</tr>
<tr>
<td>Threads Per Processor</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>Clock</td>
<td>1.6 GHz</td>
<td>1.6 GHz</td>
</tr>
<tr>
<td>Memory</td>
<td>32 GB</td>
<td>64 GB</td>
</tr>
<tr>
<td>OS</td>
<td>Linux 6.2 (Santiago)</td>
<td>Linux 6.2 (Santiago)</td>
</tr>
<tr>
<td>L2 Cache</td>
<td>1333 MHz</td>
<td>1333 MHz</td>
</tr>
<tr>
<td>Network</td>
<td>Gigabit</td>
<td>Gigabit</td>
</tr>
</tbody>
</table>