TIBCO ActiveMatrix BusinessWorks™

Performance Benchmark & Tuning Guide

Version 6.11.0 | December 2024

Contents

Changing Help Preferences

By default, documentation access from TIBCO Business Studio™ for BusinessWorks™ is online, through the TIBCO Product Documentation website that contains the latest version of the documentation. Check the website frequently for updates. To access the product documentation offline, download the documentation to a local directory or an internal web server and then change the help preferences in TIBCO Business Studio for BusinessWorks.

Before you begin

Before changing the help preferences to access documentation locally or from an internal web server, download the documentation.

1. Go to https://docs.tibco.com/

2. In the **Search** field, enter TIBCO ActiveMatrix BusinessWorks™ and press **Enter**.

3. Select the TIBCO ActiveMatrix BusinessWorks™ product from the list. This opens the product documentation page for the latest version.

4. Click **Download All**.

5. A compressed .zip file containing the latest documentation is downloaded to your web browser's default download location.

6. Copy the .zip file to a local directory or to an internal web server and unzip the file.

To point to a custom location:

Procedure

1. Perform one of the following steps in TIBCO Business Studio for BusinessWorks based on your operating system:

   - On Windows OS: Click **Window > Preferences**

   - On macOS: Click **TIBCO Business Studio > Preferences**.

2. In the Preferences dialog, click **BusinessWorks > Help**.

3. Click **Custom Location**, and then browse to the `html` directory in the folder where you extracted the documentation or provide the URL to the `html` directory on your

internal web server.

4. Click **Apply**, and then click **OK**.

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, ActiveMatrix BusinessWorks 6.x (hereinafter referred to as TIBCO BusinessWorks Container Edition) enables developers to create new services, orchestrate business processes and integrate applications in the shortest time. For more information, see the *ActiveMatrix BusinessWorks Concepts*.

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 *ActiveMatrix BusinessWorks Performance Benchmarking and 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. The document includes the different tuning parameters, steps required to configure the parameters, and design techniques for better performance. However, you must TIBCO FOCUS® on real customer use cases to understand the issue and the associated solution.

> **Note:** The performance tuning and configurations in this document is okprovided 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 6.6.1 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, after validating the customer requirements and environment. TIBCO does not guarantee their accuracy.

ActiveMatrix BusinessWorks Architecture

The most important component in ActiveMatrix BusinessWorks is BWEngine. The purpose of BWEngine 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 BWEngine 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
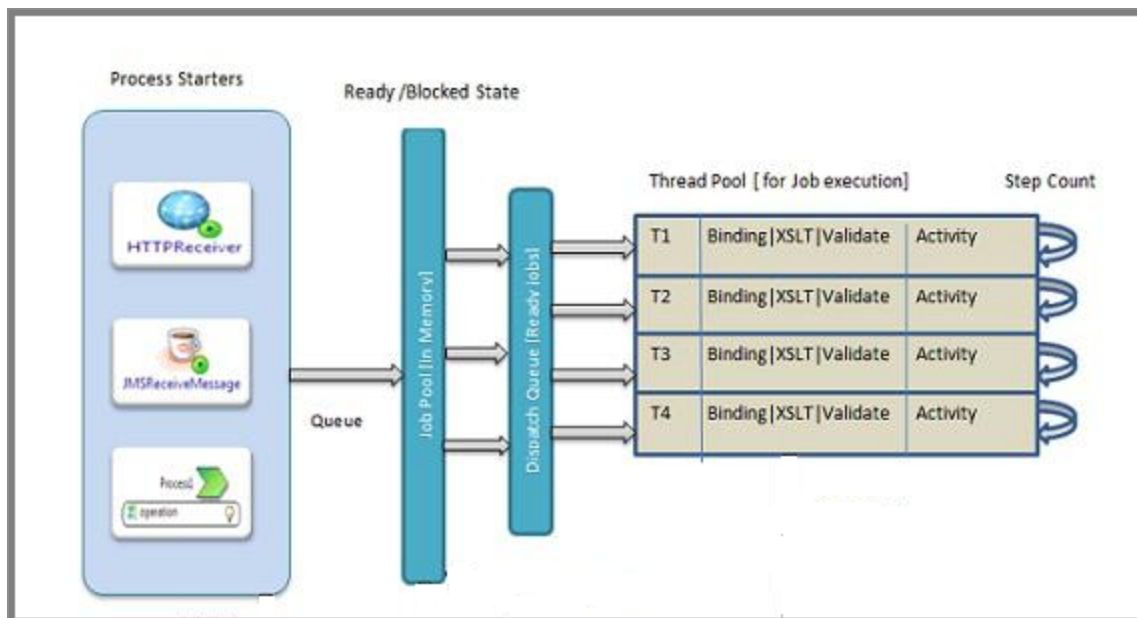
Performance Benchmark Fundamentals

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
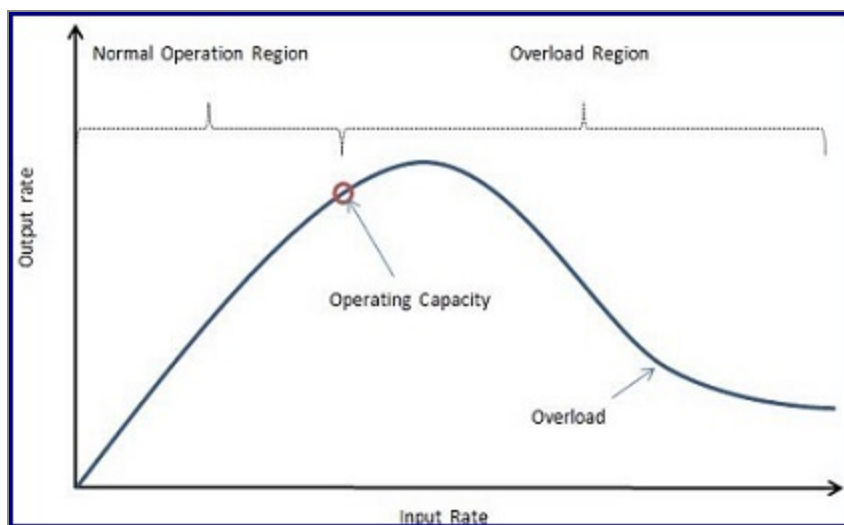
In the above example, the X axis characterizes the input rate, and the Y axis represents the output rate. The system is exposed to load at a controlled rate and is in a steady-state for some period of time. After an initial stabilization period, both the input and output rates are measured, providing one data point for the performance curve. This example assumes that all other factors are being held constant.
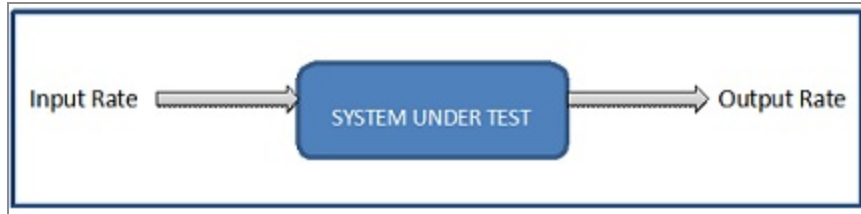
The shape of the performance curve tells us a lot about the system under test. If each input results in a single output, then over the normal operating range, the output rate exactly matches the input rate and within statistical bounds. If each input results in more than one output, or if you are measuring 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 must 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 does not 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 reaches 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 a load is inherently unstable. Inputs arrive faster than the work is getting completed, and this results in inputs piling up. This buffer for memory, disk, and messaging system is finite in capacity. At full capacity the system fails. Thus systems can, at best, operate in the overload mode 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 must 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 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 test 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 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 is 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 excercise.

- 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 ActiveMatrix BusinessWorks 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 requires multiple iterations.

One of the fundamental requirements before performing any kind of tuning exercise is to carefully eliminate all external factors that can potentially affect any performance issues.

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

- Establish performance benchmarking criteria

- Review ActiveMatrix BusinessWorks 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 must 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. For more information on JVisualVM and thread dumps, see Monitoring threads and taking a thread dump using JVisualVM.

You can generate Heap dumps from JVisualVM, and can analyze 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 such as Top, prstat, iostat, and vmstat. In addition, monitor various systems trace and log files.

Collecting Performance Data

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

Measure the memory, and CPU usage of the container during all tests. Identify the 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 improvements are identified.

Configure the operating system tool to measure memory, disk, and CPU usage during all tests. Identify the 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.

Allocate adequate resources for running BusinessWorks™ container and testing its performance. Deploy BusinessWorks™ application container and start measuring application performance using any monitoring tool like JConsole/JVisualVM. Verify that

your BW application running inside a container is providing the performance metrics, such as memory and CPU usage.

Deploy adequate hardware for running ActiveMatrix BusinessWorks™ software and testing its performance. Install ActiveMatrix BusinessWorks™, 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 a Performance Testing Plan

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 some 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 to 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 ActiveMatrix BusinessWorks jobs

- Worst performing activities, that is CPU time and Elapsed Time

The test must be completed end-to-end, and the application developer must fix any issues associated with the run. The overall percentage of error and warning must 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 must also define the payload for testing the deployed services. The application must be tested considering the peak load in the production environment.

Develop Peak Rate Tests

The overall business objectives can be different for each project. To measure the system against an extreme number of users, without failing, a peak load test is designed to determine whether the system can respond to the desired maximum number of users and requests without degradation in response time.

Performance depends on the hardware available in the environment. If the CPU and memory resources are not sufficient, then increasing the numbers further degrades 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 operating system 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 must extend results obtained in the section, "Develop Hardware and Resource Plan", 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 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 the 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.

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 the current state of the thread, running, sleeping, wait, and monitor. JVisualVM displays the thread view in real time.

Monitoring Threads and Taking a Thread Dump Using JVisualVM

Use JVisualVM to monitor threads and take thread dumps for an AppNode and bwagent.

Procedure

1. Enable JMX on the AppNode or bwagent by adding the following JMX properties in the AppNode or the agent TRA files for remote monitoring.

   - `java.property.com.sun.management.jmxremote=`*true*

   - `java.property.com.sun.management.jmxremote.port=`*8008*

   - `java.property.com.sun.management.jmxremote.authenticate=`*false*

   - `java.property.com.sun.management.jmxremote.ssl=`*false*

2. Start `jvisualvm.exe` from the `JDK_HOME/version/bin` directory.

3. Connect to the application container remotely or by using the `JMX_PORT`. Connect to the AppNode or bwagent remotely or by using the PID. To connect remotely, select **Remote** in the **Applications** tab and right-click **Add Remote Host**. Enter the remote **Host name** field.
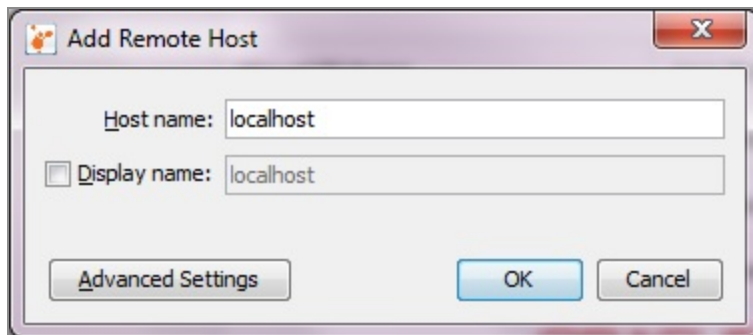
4. Add the JMX connection to the remote host as displayed in the images below.
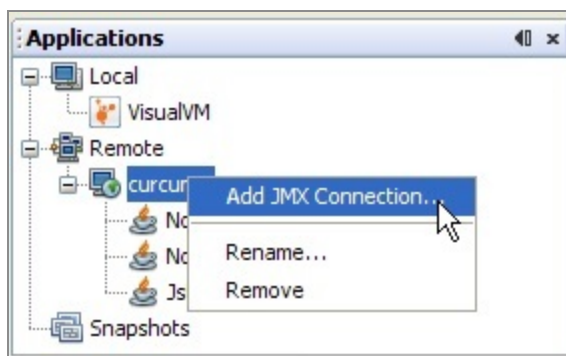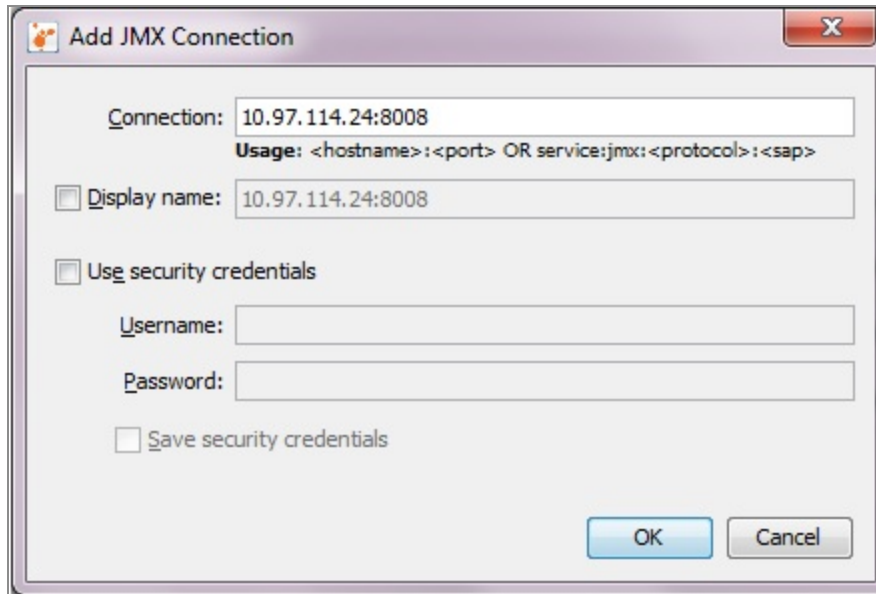
5. Right-click the remote JMX connection for the AppNode or bwagent and select **Open**.

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. You can perform a manual GC and obtain the heap dump too.

   The following figure demonstrates the typical heap usage pattern of the AppNode , which is a sawtooth pattern. This sawtooth pattern indicates the normal behavior of the JVM. For more information, see the Stack Overflow website. Here the memory usage steadily increases and then drops due to garbage collection.
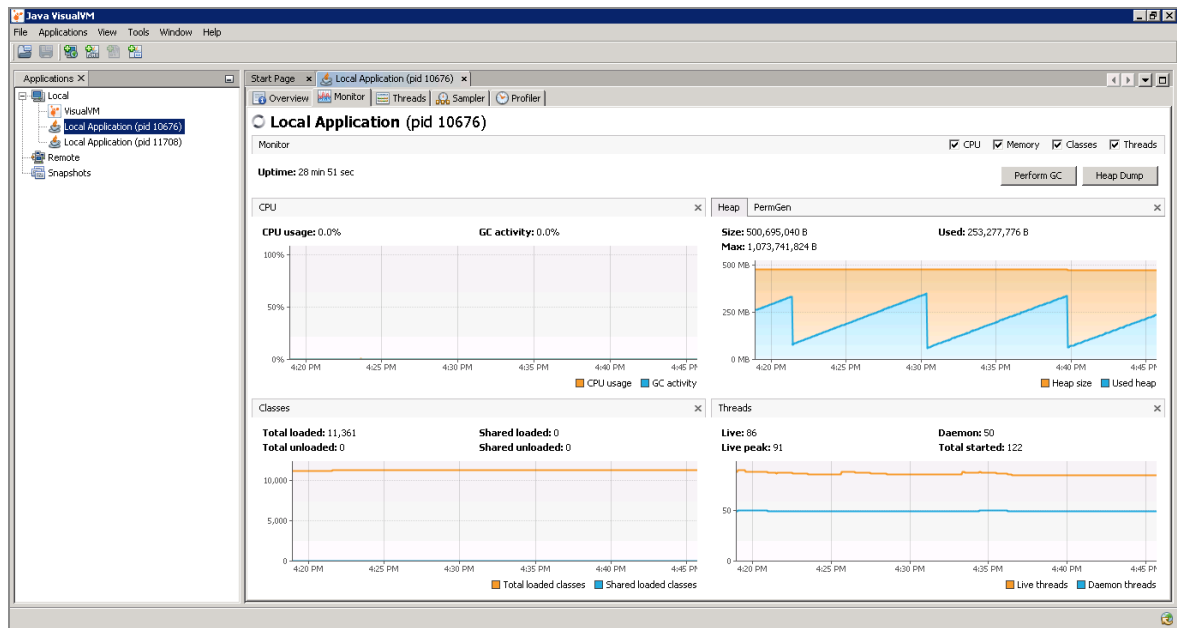
The figure below displays the heap dump summary once the dump is obtained.
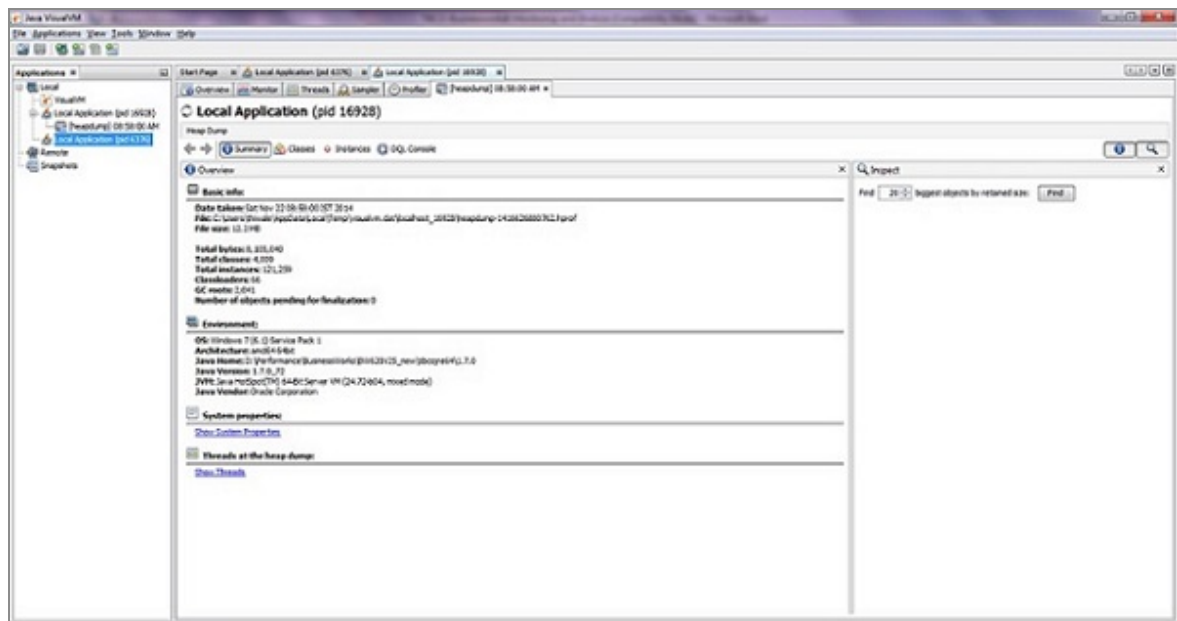
7. You can monitor the thread states and obtain the thread dump from the **Threads** tab.
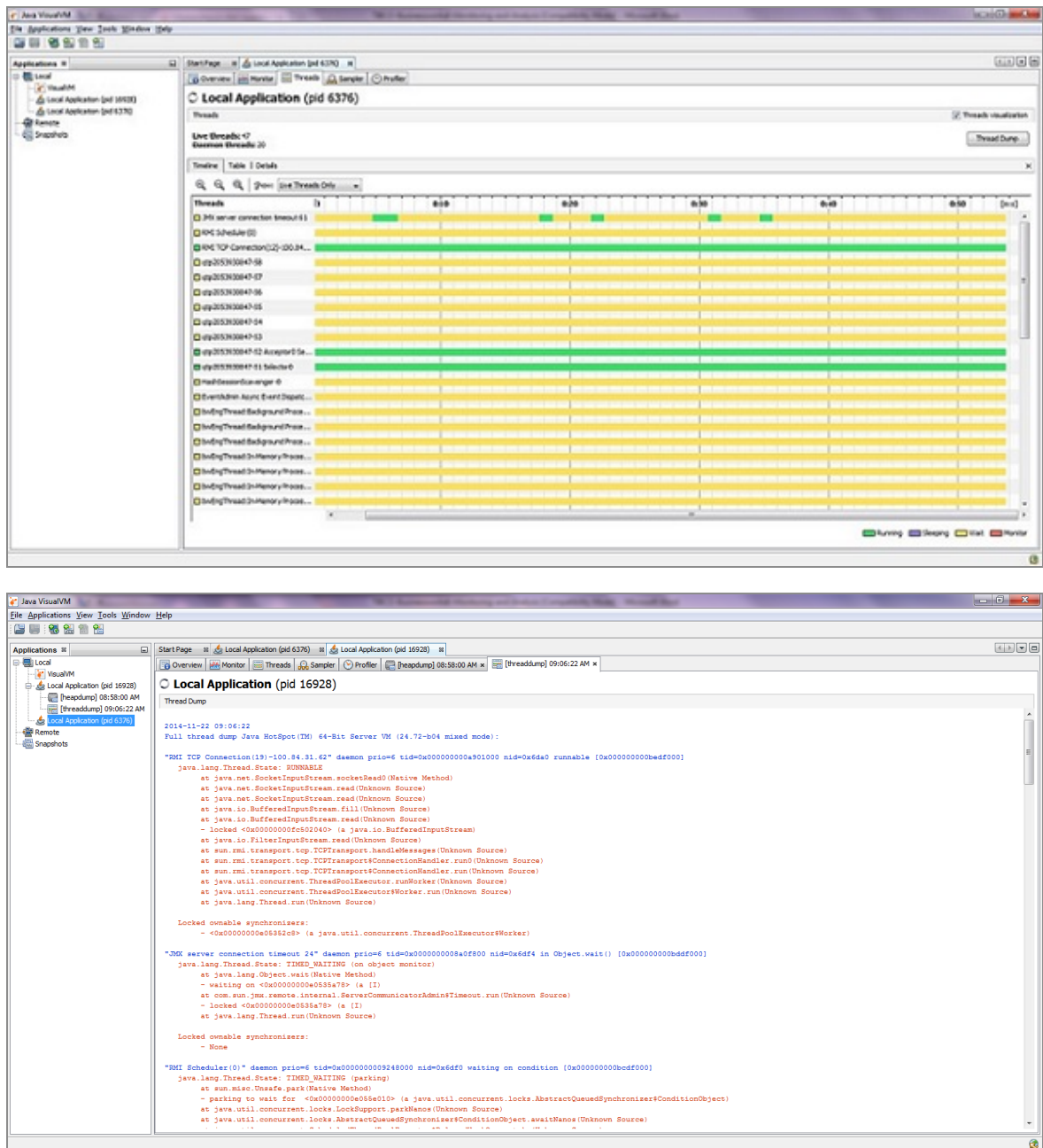
8. You can use JVisualmVM to configure an option to generate a heap dump if the AppNode or BWAgent runs out of memory. Right-click a JMX connection, and select **Enable Heap Dump on OOME** as shown in the image below.
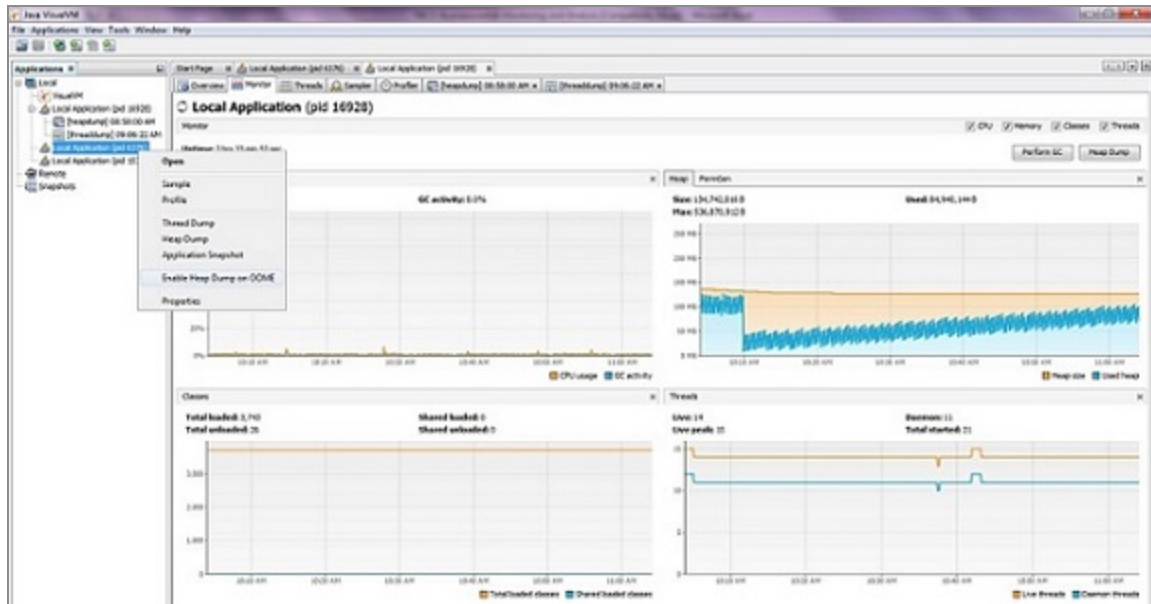
9. JVisualVM provides CPU and memory profiling capabilities. By default, the profiling tool is not in a running state until you are ready to profile the application. You can choose from the following profiling options:

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

    b. 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 the profiling data.

        When the profiling results are available, they are displayed in the **Profiler** tab.

        JVisualVM has the following plug-ins for the java implementation:

        i. A sampling profiler - Statistical and lightweight

        ii. 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 the address of the 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 information, see 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 you need to improve the performance, 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

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

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 the 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 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.

> **ℹ Note:** It is recommended to set minimum heap size to a default value and tune the maximum heap based on monitoring of memory usage and garbage collection frequency under load. It is also recommended to run baseline, scalability, and stability tests for identifying the optimum value of maximum heap size.

Perm Gen Memory

The parameters to set the Perm Gen memory of the AppNode are :

-XX:PermSize=64m

-XX:MaxPermSize=128m

> **ℹ Note:** 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. You have set this parameter in the *BW_JAVA_OPTS* environment variable.

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
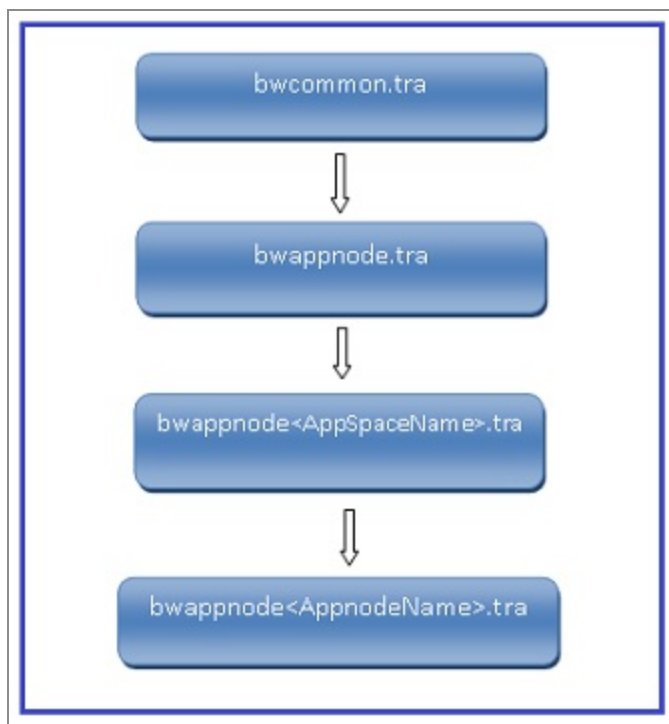
The JVM parameters can be changed in all the TRA files depending on various factors. The factors are described in the later sections.

The diagram above 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 its parent configuration.

As an example, 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 their children unless the children override the configuration.

Commonly Used TRA Files

The following table describes some of the commonly used TRA files:

TRA file	Description
`bwcommon.tra`	The `bwcommon.tra` file includes all the technical runtime parameters applicable to the ActiveMatrix BusinessWorks components (BWAgent, BWAdmin, AppNodes, and so on).
`bwappnode.tra`	The `bwappnode.tra` file includes the technical runtime parameters applicable to AppNodes.
`bwappnode<AppSpaceName>.tra`	The AppSpace TRA files are included in the TRA files of the related AppNodes. In other words, the properties included in the AppSpace TRA files are applicable to all the AppNodes that are part of it.
`bwappnode<AppNodeName>.tra`	The AppNode TRA files include all technical runtime parameters of the AppNode: the parameters inherited from the included files and the parameters defined into it.

Setting JVM Parameters for the AppNode Manually

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

- The `<BW_HOME>`/bin/bwcommon.tra file 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 are 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>`/.

   It sets the JVM properties for all the AppNodes in that particular AppSpace.

- You 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.x/domains/`<domainName>`/AppNodes/`<AppSpaceName>`/`<AppNodeName>`/bindirectory.

   It sets 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.

For more information, see :

- Usecase1

- Usecase2

- Usecase3

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.x/bin folder the following changes 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 out by default. These are the settings that an AppNode takes when created, and if the JVM parameters are already set at the AppSpace level, those settings take effect. For more information, see Usecase 2.

- **After changing the JVM parameters and without restarting the AppNodes** - No

changes in the AppNode settings.

- **After changing the JVM parameters and AppNode restart** - The changes are applied to all the AppNodes in all the AppSpaces in the environment. You must handle this change very carefully because, though it allows you 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 maximum heap of an existing AppNode is not explicitly specified, it is 1 GB by default. However, if the maximum heap in the files, `bwcommon.tra` or `bwappnode.tra`, is changed to a higher value at some point of time, a lot of memory is allocated to the existing AppNodes when they are restarted. This may not be a requirement for some AppNodes.

  On the other hand, if the memory allocated in the files, `bwcommon.tra` and `bwappnode.tra`, is changed to a lower value such as 512 MB, on restart the existing nodes have maximum heap of 512 MB. This may not be sufficient for an AppNode that has a higher memory requirement. This situation can lead to instability in the environment along with mismanagement of resources particularly in terms of memory.

New AppNodes

Changes get applied to any new AppNodes that are created in that particular AppSpace. AppNodes created in any other AppSpace have no impact. Any new AppNode created after setting the JVM parameters contains 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 maximum 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` must 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 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 take effect. For more information, see 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 are applied to all the existing AppNodes in that particular AppSpace. The AppNodes in other AppSpaces are not affected.

New AppNodes

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

In this particular use case, TIBCO recommends that since changing the settings at the AppSpace level affect all the AppNodes in the AppSpace, the requirements for all the AppNodes in that particular AppSpace must 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.

For more information about how to change the JVM parameters for a particular AppNode in an AppSpace, see Setting JVM Parameters for the AppNode Manually .

The following changes are made:

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 out 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, (see Usecase 1), or at the AppSpace level, (See Usecase 2), then those settings take effect.

- **After changing the JVM parameters and without restarting of AppNodes** - No changes in the AppNode settings.

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

New AppNodes

There is no impact on the AppNodes created.

Best Practices

This section describes some of the 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 ActiveMatrix BusinessWorks.

- ActiveMatrix BusinessWorks Transport and Resource Tuning Guidelines

- JVM Tuning Guidelines

- Tuning Guidelines

> **Note:** TIBCO recommends to first tune the BWEngine until all the resources are fully utilized and then go for container tuning.

Engine Tuning Guidelines

This section provides high-level steps for ActiveMatrix BusinessWorks performance tuning. Specific details are discussed in the subsequent sections.

When tuning the ActiveMatrix BusinessWorks engine allocate significant time to select 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 can allocate is set in the AppNode `config .ini` file

- Measuring the available CPU and memory resources on your system under a typical processing load helps 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 ways of using CPU resources to the maximum capacity:

  - 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 allocates .The default value of engine threads is eight.

The jobs in memory are run by the engine. The number of jobs that can be run 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 run a finite number of tasks or activities uninterrupted 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 must be measured under a typical processing load to determine if the default threadCount value is suitable for the environment. Please refer the below formats to set threadCount value on different container platforms: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 run 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

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 contains 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 is eight.

If you want to set the value for one process starter only, set the flow limit at the component level by using the environment variable, *BW_COMPONENT_JOB_FLOWLIMIT*.

> **Note:** For component level, use only the *BW_COMPONENT_JOB_FLOWLIMIT* environment variable.

If the number of jobs being created exceeds the flow limit, the engine suspends the creation of new jobs but continues running the jobs in memory. The engine resumes creating 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.

> **ⓘ Note:**
> - Only set the `bw.application.job.flowlimit` property for applications using non-HTTP-based transports, for example, JMS. If applications are using HTTP-based transports, ensure you set the **Max QTP Threads** value of the **HTTP Connector** shared resource to apply the Flow limit.
>
> - To push a new property value to runtime, stop the application, update this property in the AppNode or AppSpace `config.ini` file, and restart the application.
>
>   You can update the flow limit without restarting an application using the Admin UI and the REST API. For more information, see "Engine and Job Tuning" in *TIBCO BusinessWorks™ Container Edition Administration*.

- 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

> **ⓘ Note:** Get the logs for the component states such as Start, Stop, and Resume based on whether the `Flowlimit` is breached or complied by enabling the core runtime logger `com.tibco.bw.core` at INFO level.

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 service bindings job page threshold value. This value can be set in the AppNode config.ini file

> **ⓘ** **Note:** To dynamically push a new property value to runtime, stop the application, update this property in the AppNode or AppSpace config.ini file, and restart the application.

- 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 keeps open at one time.

The minimum value allowed is ten. Any value less than ten disables the bundle file limit, making the number of jar files the framework keeps open unlimited. By default the value is hundred. 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 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 file 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 an individual processes can be enabled or disabled from the TIBCO ActiveMatrix BusinessWorks™ Admin CLI by using the `enablestats` and `disablestats` commands respectively.

For more information, see about these commands, see the *TIBCO ActiveMatrix BusinessWorks™ Administration* guide

Process Execution Statistics

Using process execution statistics, you can collect information for individual processes and activities of an application such as 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 placed in `[BW_HOME]/bw/6.3/domains/perf/appnodes/<AppSpaceName>/<AppNodeName>/stats/activitystats.csv` file.

```
bwadmin[admin@perf/perf/perf]> enablestats activityinstance HTTPServer.application 1.0
TIBCO-BW-ADMIN-300413: Enabled statistics collection for Application [HTTPServer.application:1.0].
```

Interpreting Activity Instance Statistics

The following activity instance statistics are collected:

Statistic	Description
Application Name	Name of the application.
Application Version	Version of the application.
Module Name	Name of the TIBCO ActiveMatrix BusinessWorks™ module.

Statistic	Description
Module Version	Version of the ActiveMatrix BusinessWorks™ module.
Activity Name	Name of the activity.
Process Name	Name of the process.
Process Instance ID	Instance ID of the process.
Start Time	Process instance start time.
Eval Time	The Eval Time for an activity is the actual time (in milliseconds) used by the activity itself to complete while using the engine thread. Asynchronous activities may use other threads not included in this time.
Elapsed Time	Elapsed time of an activity is the time difference (in milliseconds) between start time and end time of the activity. Between the start and end time, control may get switched with other activities from the other jobs. This is the time taken to run an activity plus all the delays in acquiring resources like engine threads, JDBC connections, network. The elapsed time includes the execution time plus time taken for evaluating all the forward transitions from that particular activity and getting the next activity ready to run, which includes running its input mapping if all dependencies are met.
Status	Status of activity, for example: Completed, Faulted, or Canceled.

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 navigate to the AppNode level and run `enablestats processinstance <ApplicationName> <ApplicationVersion>`

4. By default the statistics are collected in the CSV file format at `[BW_ HOME]/bw/6.x/domains/perf/appnodes/`*`<AppSpaceName>`* `/<AppNodeName>/stats/processsstats.csv` file.

```
bwadmin[admin@perf/perf/perf]> enablestats processinstance HTTPServer.application 1.0
TIBCO-BW-ADMIN-300413: Enabled statistics collection for Application [HTTPServer.application:1.0].
```

Interpreting Process Instance Statistics

The following activity instance statistics are collected:

Statistic	Description
Application Name	Name of the application.
Application Version	Version of the application.
Module Name	Name of the TIBCO ActiveMatrix BusinessWorks™ module.
Module Version	Version of the ActiveMatrix BusinessWorks™ module.
Component Process Name	Name of process configured to a component. If the process is a non in-lined sub process, this could be empty.
Job ID	Job ID of the process.
Parent Process Name	If the process is an in-lined sub process, the name of the parent process.
Parent Process ID	If the process is an in-lined sub process, the instance ID of the parent process.

Statistic	Description
Process Name	Name fo the process.
Process Instance ID	Instance ID of the process.
Start Time	Process instance start time.
End Time	Process instance end time.
Elapsed Time	Elapsed time for a process is the total time taken by the process, including the elapsed time for all the activities run for the process.
Eval Time	The Eval Time for a process instance is the total evaluation time (in milliseconds) for all the activities run for the process instance.
Status	Status of process instance, for example: Completed or Faulted.

Since elapsed time is the time taken to run 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 evaluation time means incorrect engine tuning or a bad design in the process. This means the engine is starving for threads. If there are no real activities that are taking high evaluation times, it is a matter of tuning the ActiveMatrix BusinessWorks engine. Elapsed time is always equal to, or more than the CPU time.

SupressXMLSchemaonFault (bw.engine.suppress.xml.schema.on.fault)

When set to true, suppresses the XML schema from error logs when a fault is encountered while parsing an XML file that is based on the specified schema.

If the **bw.engine.suppress.xml.schema.on.fault** property is set to false, left undefined, or not set in the config.ini file, the XML schema is displayed in the error log.

The **bw.engine.suppress.xml.schema.on.fault** property can be added, and set, in the AppNode or the AppSpace config.ini file.

JVM Tuning Guidelines

Each ActiveMatrix BusinessWorks engine runs as a multi-threaded Java server application. Processes and other objects used internally by 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 an AppNode, you must determine the largest heap size that can reside in a physical memory. For best engine performance, paging to disk must 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 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.

    For more information, see Setting Appnode JVM Parameters .

> **Note:** TIBCO recommends to set minimum heap size to default value and tune the maximum heap based on monitoring of memory usage and garbage collection frequency under load. It is also recommended to run baseline, scalability, and stability tests for identifying the optimum value of maximum heap size.

The total amount of JVM memory required to operate a ActiveMatrix BusinessWorks engine must 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.

> **Note:** While uploading ears, deploying applications, and browsing, increase the heap size in bwagent and `AppSpace.tra` files. The property specified in the `AppSpace.tra` file applies to all AppNodes associated with the AppSpace. However, the property specified in the `AppNode.tra` file only applies to a specific AppNode and overwrites the property specified in the `AppSpace.tra` file. The property specified in `bwagent.tra` does not overwrite properties in the `AppNode.tra` or the `AppSpace.tra` files.

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.

> **Note:** The client must 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.

We recommend that the application must 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` command. These properties can be appended to the `java.extended.properties` in the AppNode TRA file.

Transport and Resource Tuning Guidelines

Performance must be one of the criteria for using the appropriate transport. Besides performance, you must also consider interoperability, reliability, and security requirements.

You have choices of using HTTP or JMS, SOAP over HTTP, SOAP over JMS and TIBCO Active Enterprise™ 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 must use them only as general observations, and for initial understanding. For 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 ActiveMatrix BusinessWorks.

In some situations, you can alter the configuration of the HTTP server that receives incoming HTTP requests for 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 that accept the incoming HTTP requests. While tuning these threads, the rule of thumb, as per the Jetty documentation, is **acceptors > =1<=#CPUs**.

- 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 10.

- **Maximum QTP Threads**: The maximum number of threads available for incoming HTTP requests. The HTTP server cannot create more than the number of threads specified by this parameter. The default value is 75. This limit is useful for determining the number of incoming requests that can be processed at a time. Setting a high number creates that many threads and drastically reduce the performance.

TIBCO recommends 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.

In addition to the QTP threads, REST services also include the bw-flowlimit-executor-provider thread pool, which is a part of the Jersey framework. The QTP threads offload the work to bw-flowlimit-executor-provider threads. These threads handle the processing of the RESTful services. The core pool size of these threads is equivalent to the min QTP thread pool that is the settings are picked from the values provided for the QTP pool on the HTTP Connector resource. The pool size values cannot be modified directly, which means that if users want to modify the thread settings, they need to change the QTP thread pool values.

All the incoming requests are processed by one of the available Jersey threads. If in case the Jersey thread is not available, the request is queued (based on blocking queue size), and this holds true until the Queue gets filled up. If the load persists continuously and the Queue is already full, then the Jersey threads are scaled up to maxQTP(in other words, Jersey maxPoolSize), that is the setting are picked from the values provided for the QTP pool on the HTTP Connector resource.

The bw-flowlimit-executor-provider threads are created for every service deployed in the application EAR. If there are 10 services deployed in a single EAR, on a single AppNode container, then the bw-flowlimit-executor-provider threads created would be equal to the number of services deployed that is (10) * CorePoolSize.

For more information about configuration details of the parameters, see Tuning 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. For more information, see Hardware Configuration.

For scenario details, see Appendix A.

> ℹ **Note:** The numbers represented in the graphs are based on the tests conducted in the performance lab for ActiveMatrix BusinessWorks 6.3.0. Results may vary according to the components installed, the work load, the type and complexity of different scenarios, hardware and software configuration, and so on. They can be reproduced only under the exact environment and workload conditions that existed when these tests were performed.

Scenario (1) HTTP Performance

Scenario (2) SOAP (HTTP) Performance

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 arises 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 run 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 run. 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 must be increased only if the client side has more backlogs compared to the receive side. TIBCO recommends 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 are maintained. Using the connection pooling feature you can keep the default values for the number of connections that the client establishes with the service, or tune them appropriately. The values must not be set to zero (0).

For more information, see Tuning Parameters.

JMS Resource and JMS Transport

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

Usage Recommendations

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.

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.

When using Client Ack, the JMS session cannot receive a new message until the current message is acknowledged.

Using ActiveMatrix BusinessWorks you can configure multiple sessions to receive messages faster, and set number of sessions higher than the number of engine threads.

Acknowledge and confirm messages as soon as possible, to improve throughput.

By holding Client ack to the end of the process, you block that session. This means you slow down the rate at which ActiveMatrix BusinessWorks pulls messages from the JMS server, which holds messages for a longer period of time.

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 is sufficient in this scenario.

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

For multiple JMS Receiver activities on an AppNode, the polling interval value has an impact on the CPU utilization of the AppNode. Lower the polling interval, higher is the CPU utilization, and higher the polling interval, lower the CPU utilization. In an environment where there are

Usage Recommendations

constraints on the available CPU resources, TIBCO recommends that the polling interval value must be increased to lower the CPU consumption.

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.

For scenario details, see Appendix A .

The graphs below illustrate the maximum throughput performance for scenarios with and without SSL for the given hardware. For more information, see Hardware Configuration .

For scenario and settings details, see HTTP Connector Resource .

> ℹ **Note:** The numbers represented in the graphs are based on tests conducted in the performance lab for TIBCO BusinessWorks™ Container Edition 6.3.0. Results may vary according to the components installed, the workload, the type and complexity of different scenarios, hardware, software configuration, and so on. They can be reproduced only under the exact environment and workload conditions that existed when these tests were performed.

Performance of HTTP vs HTTPs
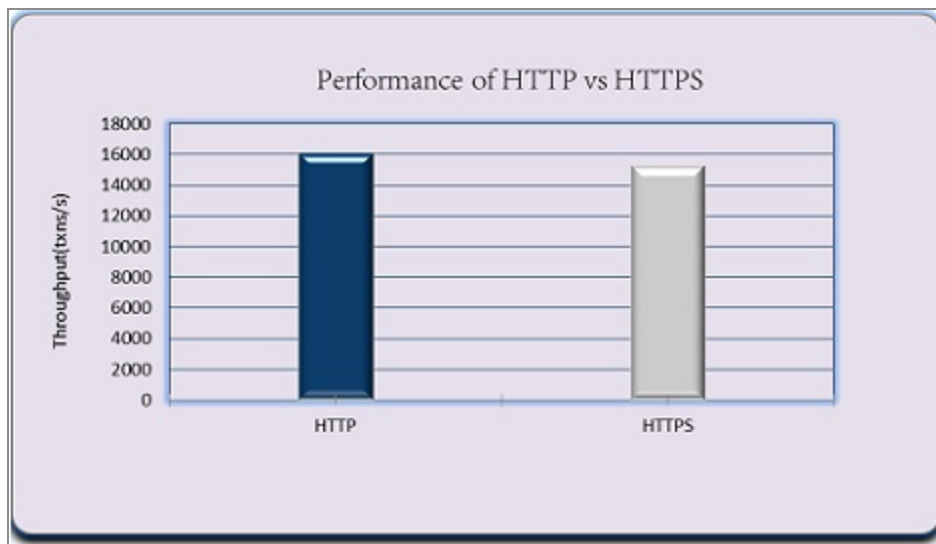
Performance of SOAP (HTTP) vs SOAP (HTTPs)
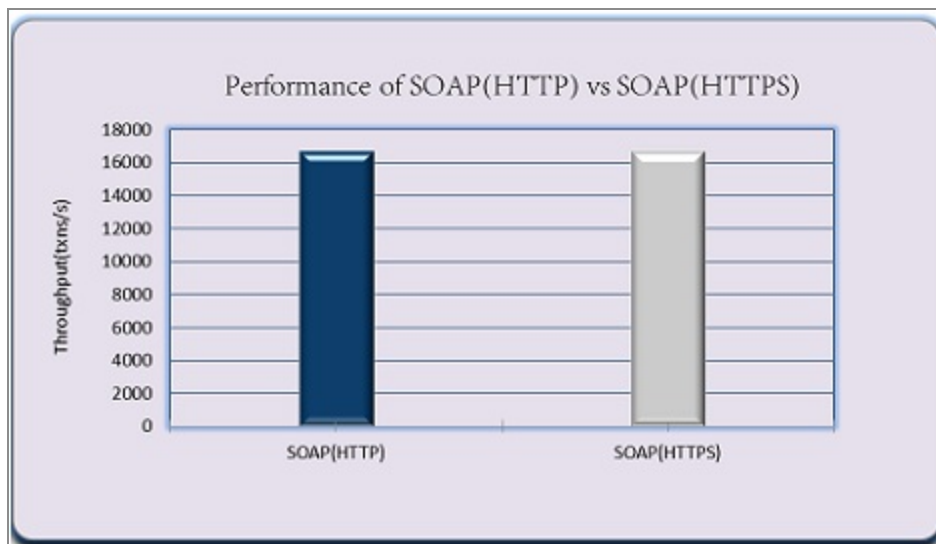
Tuning Parameters

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

- HTTP Connector Resource
- HTTP Client Resource Tuning Parameters
- JDBC Connection Resource
- TCP Connection Resource
- JMS Receiver and Wait for JMS Request
- JMS Receiver
- Apache Common Logger
- Blocking Queue Size

HTTP Connector Resource
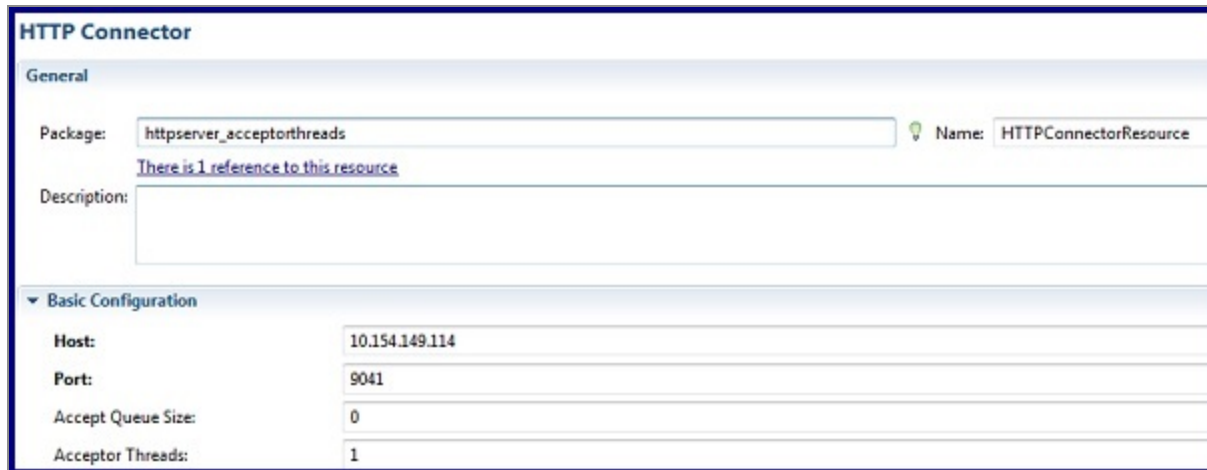
This section describes the tuning parameters for the **HTTP Connector** resource that you can configure in TIBCO Business Studio™ for BusinessWorks™.

Basic Configuration

Field	Description
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.
Accept Queue Size	This is the number of connection requests to be queued before the operating system starts sending rejections.

Field	Description
	The value can be set to either 0 or -1 .These values signify that the queue size is 50 or OS-specific.

HTTP Connector - Basic Configuration

Advanced Configuration

Field	Description
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 default values are: • Minimum QTP threads = 10 • Maximum QTP threads = 75

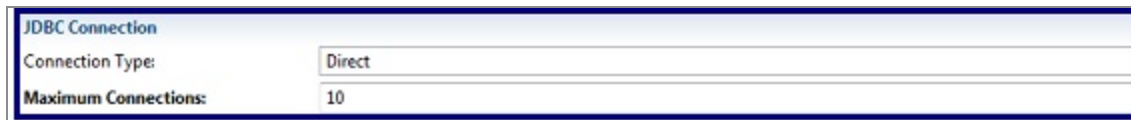HTTP Connector - Advanced Configuration

HTTP Client Resource

This section describes the tuning parameters for the **HTTP Client** resource that you can configure in TIBCO Business Studio for BusinessWorks.
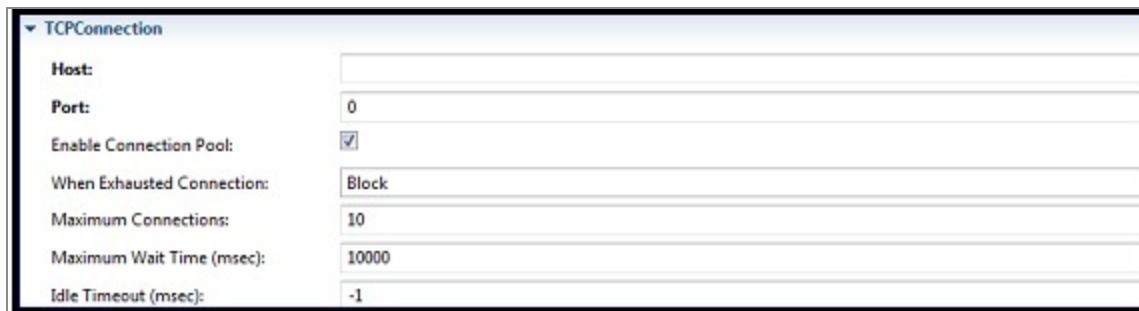
HTTP Client

You can configure the following fields.

Field	Description
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 not selected. For applications that create many long lived connections, increase the value of this parameter.
	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 not selected.
	This value cannot be more than the maximum total connections. The value for these threads can be modified at design time in TIBCO Business Studio

Field	Description
	for BusinessWorks as shown in the snapshot. Every connection created here also counts into Maximum Total Connections.
	Default value = 20

HTTP Client Resource

Thread Pool Resource

You can optionally create the client thread pool that 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. The values for these threads can be modified at design time in TIBCO Business Studio for BusinessWorks.

The default values are:

- Core Pool Size = 5

- Max pool Size = 10

HTTP Client Thread Pool
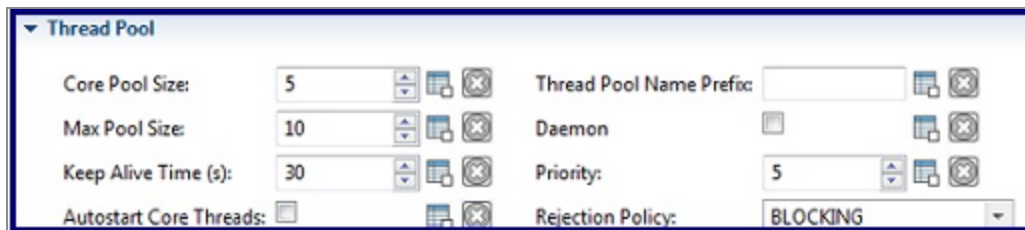
JDBC Connection Resource

This section describes the tuning parameters for the **JDBC Connection** resource that you can configure in TIBCO Business Studio for BusinessWorks.

JDBC Connection

You can configure the following field.

Field	Description
Max Connections	This parameter specifies the maximum number of database connections to allocate. Default value = 10

JDBC Connection Resource

TCP Connection Resource

This section describes the tuning parameters for the **TCP Connection** resource that you can configure in TIBCO Business Studio for BusinessWorks.

TCP Connection

You can configure the following fields.

Field	Description
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.

Field	Description
	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.

Default value = 10000 ms |

TCP Connection
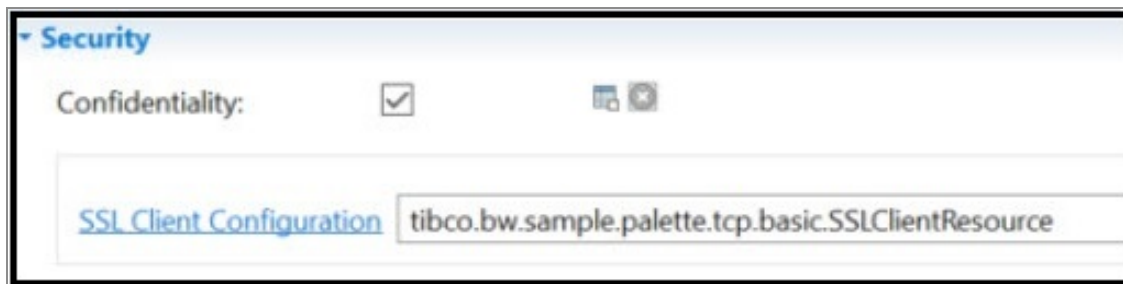
Security

Select the **Confidentiality** checkbox to encrypt or decrypt messages. When you select the checkbox, the SSL Client Configuration field is visible. For more information about SSL Client Configuration, see "SSL Client Configuration" in *TIBCO ActiveMatrix BusinessWorks™ Bindings and Palettes Reference*.

TCP Connection Security

JMS Receiver and Wait for JMS Request

This section describes the tuning parameters for the JMS messaging servers. You can configure the value in the **General** tab of the **JMS Receive Message** and **Wait for JMS Request** activities in TIBCO Business Studio for BusinessWorks

Field	Description
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. Default value = 1

JMS Palette
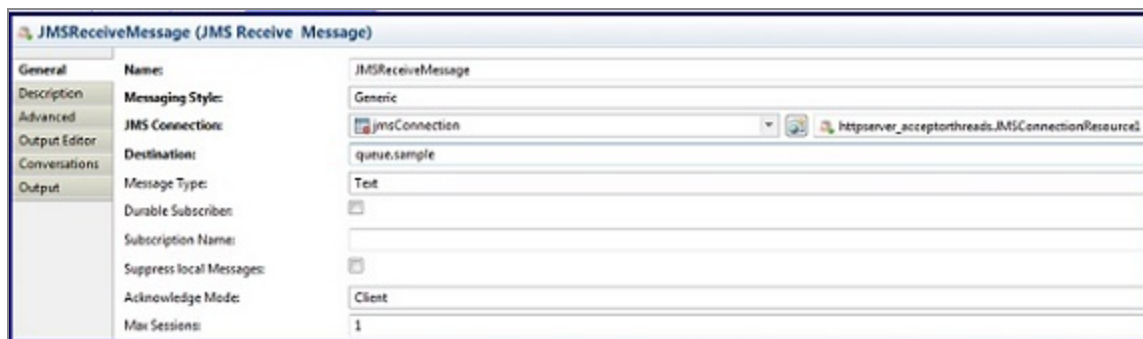
JMS Receiver

You can configure the polling interval variable in the **Advanced** tab of the **JMS Receive Message** activity.

Field	Description
Polling Interval (sec)	This property specifies the polling interval in seconds to check for new messages. If a value is not specified for the property, Setting a value in this field overrides the default polling interval. Default value = 2

JMS ReceiveMessage

Apache Common Logger

This section describes the tuning that can be done at the Apache logging level for performance improvement. The Apache Common logger is enabled by default and disabling the Apache logger improves CPU utilization and response time.

To diasble the logger, set the property `Dorg.apache.commons.logging.Log=org.apache.commons.logging.impl.NoOpLog` in the AppNode `config.ini` file, and append the property `Dorg.apache.commons.logging.Log=org.apache.commons.logging.impl.NoOpLog` to java.extended.properties in the `AppNode.tra` file.

Blocking Queue Size

This is the number of requests to be queued before the operating system starts sending rejections. In case of unavailability of Jersey threads, all the incoming requests get queued up until the queue gets full. The default blocking Queue size is Integer.MAX_VALUE which is unbounded. The Blocking Queue size can be modified at design time in TIBCO Business Studio for BusinessWorks as shown in the following image.

Parameter location: **Bindings > RestService > Advanced Configurations > Blocking Queue Size**
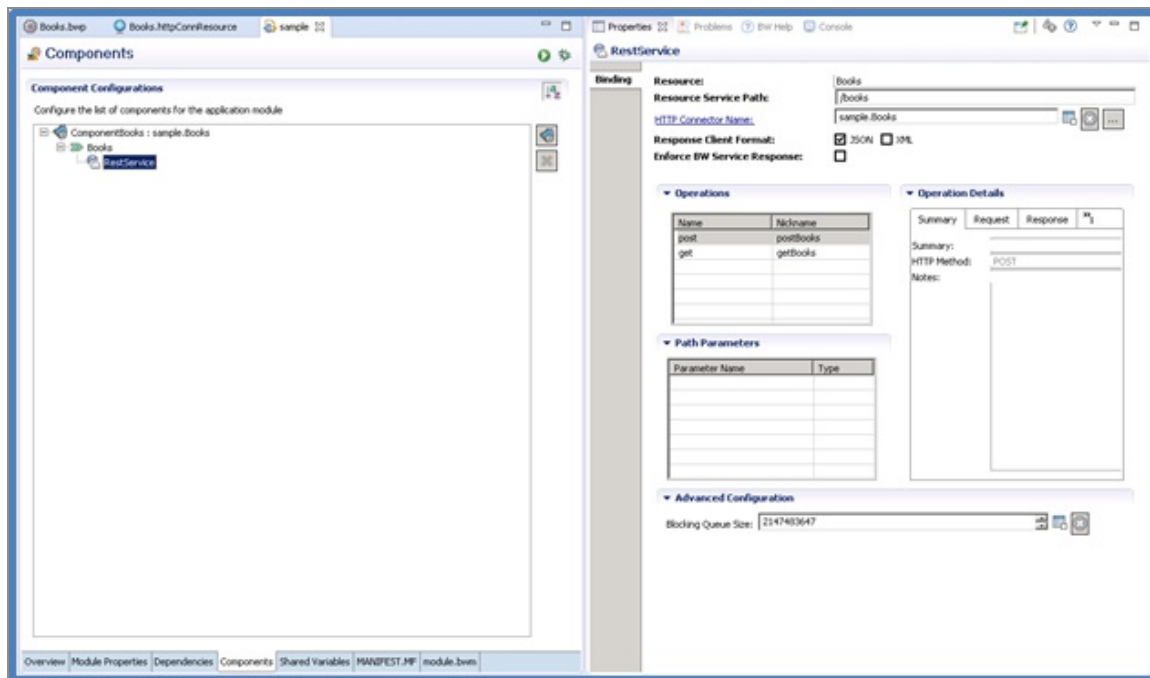
Debugging Performance Issues

This section describes how to debug for performance related issues.

- Debugging High CPU Utilization Issues

- Debugging High Memory Utilization Issues

- Debugging High Latency Issues

Debugging High CPU Utilization Issues

The CPU utilization of an AppNode for a particular service depends on the complexity of service implementation, payload, workload, number of services deployed on the AppNode, and the CPU or Memory resources made available to the AppNode.

Before you begin

Assuming all the components of the engine are tuned for debugging high CPU utilization issues on the appnode, collect the following data that helps in further debugging and understanding the issues.

Procedure

1. Capture thread dumps for analyzing the thread state and calls. Capture five thread dumps at an interval of 5 seconds each. Redirect all the collected threads dumps to separate files.

    - The thread dumps can be captured using the *jstack* utility shipped with JDK.

      ```
      ./jstack <PID of appnode> > ThreadDump_n.txt
      ```

    - The thread dumps can be captured through *JConsole* or *JVisualVM* utilities.

2. Capture top CPU consuming threads data for 5 minutes run by using a *jvmtop* utility.

    For more information about the *jvmtop* utility, see support article KB000034702.

```
./jvmtop <PID of appnode> > JVM_topthreads.txt
```

3. Capture method level CPU profiling data for 5 minutes run by using a *jvmtop* utility.

```
./jvmtop --profile <PID of appnode> > JVM_CPUProfile.txt
```

4. Capture AppNode logs for the test run duration.

5. Capture the CPU and memory utilization of the AppNode for 5-minutes run.
   - The data can be captured through a *top* utility on Unix.

   ```
   top -p <PID of appnode> > top_appnode.txt
   ```

   - The data can also be captured through *JConsole* or *JVisualVM* utilities.

6. Capture the AppNode `config.ini` and TRA file of the AppNode. This helps analyze the JVM parameters and other engine tuning parameters.

7. Capture system configurations of servers such as CPU details, RAM, and number of cores where, TIBCO BusinessWorks™ Container EditionAppNodes, external services, and load generator are running. Capture details of `/proc/meminfo` and `/proc/cpuinfo` files.

```
cat /proc/meminfo and cat /proc/cpuinfo
```

Debugging High Memory Utilization Issues

The memory utilization for a particular service on the AppNode depends on the complexity of service implementation, payload, workload, the number of services deployed on the AppNode, CPU, or memory resources made available to the AppNode.

Before you begin

Assuming all the components of the engine are tuned for debugging high memory utilization issues on the AppNode, collect the following data that helps in further debugging and understanding the issues.

Procedure

1. Capture the heap dump on the AppNode when memory issues are observed on the AppNode.

   - The heap dumps can be captured by using the *jmap* utility.

     ```
     jmap –dump:live,file=memorydump.hprof <PID of appnode>
     ```

   - The heap dumps can also be captured through *JConsole* or *JVisualVM* utilities.

   The heap dump can be analyzed using the memory analyzer tool for checking memory leaks and top components of memory.

2. Capture *jstat* data for checking the allocation and utilization of different memory pools for 5-minutes run.

   ```
   jstat –gc <PID of appnode> > jstat_gc.txt
   ```

3. Capture thread dumps for analyzing the thread state and calls. Capture five thread dumps at an interval of 5 seconds each. Redirect all the collected threads dumps to separate files.

   - The thread dumps can be captured by using the *jstack* utility shipped with JDK.

     ```
     ./jstack <PID of appnode> > ThreadDump_n.txt
     ```

   - The thread dumps can also be captured through *JConsole* or *JVisualVM*.

4. Capture AppNode logs for the test run duration.

5. Capture the CPU and memory utilization data of the AppNode for 5-minutes run.

   - The data can be captured through the *top* utility on Unix.

     ```
     top –p <PID of > > top_appnode.txt
     ```

   - The data can also be captured through *JConsole* or *JVisualVM* utilities.

6. Capture the AppNode `config.ini` and TRA file of the AppNode. This helps analyze the JVM parameters and other engine tuning parameters.

7. Capture system configurations of servers such as CPU details, RAM, and number of cores where ActiveMatrix BusinessWorksAppNode, external services, and load

generator are running. Capture details of `/proc/meminfo` and `/proc/cpuinfo` files.

```
cat /proc/meminfo and cat /proc/cpuinfo
```

Debugging High Latency Issues

The latency of a particular service depends on the complexity of service implementation, payload, workload, the number of services deployed on an AppNode, and CPU or memory resources made available to the AppNode.

Before you begin

Assuming all the components of the engine are tuned for debugging high latency issues on the an AppNode, collect the following data that helps in further debugging and understanding the issues.

Procedure

1. Capture the process execution statistics for the test run. This would help analyze the time spent in individual processes and activities. For more information about enabling statistics on deployed service, see the *ActiveMatrix BusinessWorks Administration* guide.

2. Capture thread dumps for analyzing the thread state and calls. Capture five thread dumps at an interval of 5 seconds each. Redirect all the collected threads dumps to separate files.

    -   The thread dumps can be captured using the *jstack* utility shipped with JDK.

        ```
        ./jstack <PID of appnode> > ThreadDump_n.txt
        ```

    -   The thread dumps can be captured through the *JConsole* or *JVisualVM* utilities.

3. Capture top CPU consuming threads data for 5 minutes run by using the *jvmtop* utility.

    For more information about the *jvmtop* utility, see support article KB000034702.

    ```
    ./jvmtop <PID of appnode> > JVM_topthreads.txt
    ```

4. Capture method level CPU profiling data for 5 minutes run by using the *jvmtop* utility.

```
./jvmtop --profile <PID of appnode> > JVM_CPUProfile.txt
```

5. Capture AppNode logs for the test run duration.

6. Capture the CPU and the memory utilization data for five runs on the AppNode.

   - The data can be captured through the *top* utility on Unix.

     ```
     top -p <PID of appnode> > top_appnode.txt
     ```

   - The data can also be captured through *JConsole* or *JVisualVM* utilities.

7. Capture the AppNode `config.ini` and the TRA file of the AppNode. It helps analyze the JVM parameters, and other engine tuning parameters.

8. Capture system configurations of servers such as CPU details, RAM, and number of cores where ActiveMatrix BusinessWorks AppNode, external services, and load generator are running. Capture details of `/proc/meminfo` and `/proc/cpuinfo` files.

   ```
   cat /proc/meminfo and cat /proc/cpuinfo
   ```

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.

Performance Improvement Use Cases - Design Time and Deployment

To accomplish a business objective or requirement, designing the applications appropriately is an important aspect, as this impacts the overall performance of the application and the system as a whole, to some extent. With complexities being introduced in the way the processes in ActiveMatrix BusinessWorks are designed it is imperative that users are able to adhere to best practices which eventually lead to better end-to-end performance at run time. The use cases discussed here are based on the experience for large production implementations and proof of concept scenarios where implementing certain design changes helped improve the performance.

Usecase 1: Using File as the Input Type for Parse Data Activity

Use Case 1: A customer in the banking domain observed that the time taken to parse records was increasing with every record.
The process is shown in the following image:
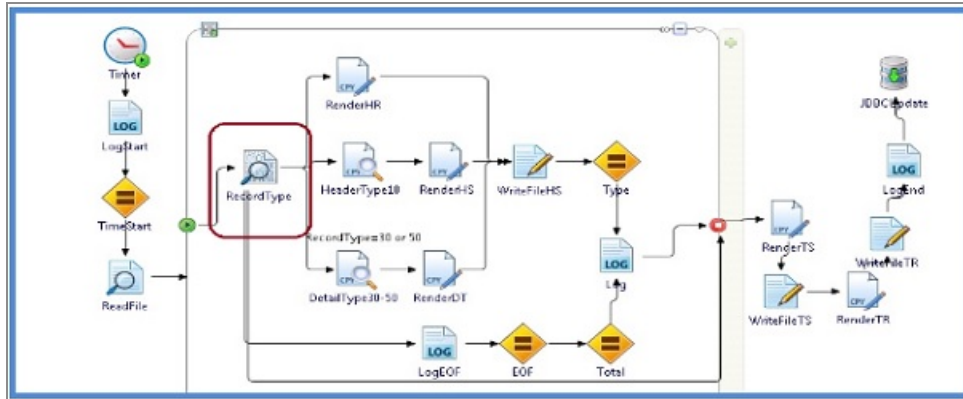
The project is designed to parse data in an iterative manner from a file in the text format. This data is converted into a schema which is then evaluated against certain conditions, by the transitions. Based on specific values, the data is rendered in a particular format using the data conversion ActiveMatrix BusinessWorks plug-in. This parsed data is written to files and then eventually updated to the database.

Initial analysis showed that the overall high latency was due to the **Parse Data** activity highlighted in the above image.

Further analysis revealed that the time taken to parse the records was high since the input type of the **Parse Data** activity was configured to string, as displayed in the image below. When the input type is set to string, the entire contents of the source are read. For accessing specific records the search operation is performed in such a way that the entire source file is scanned.
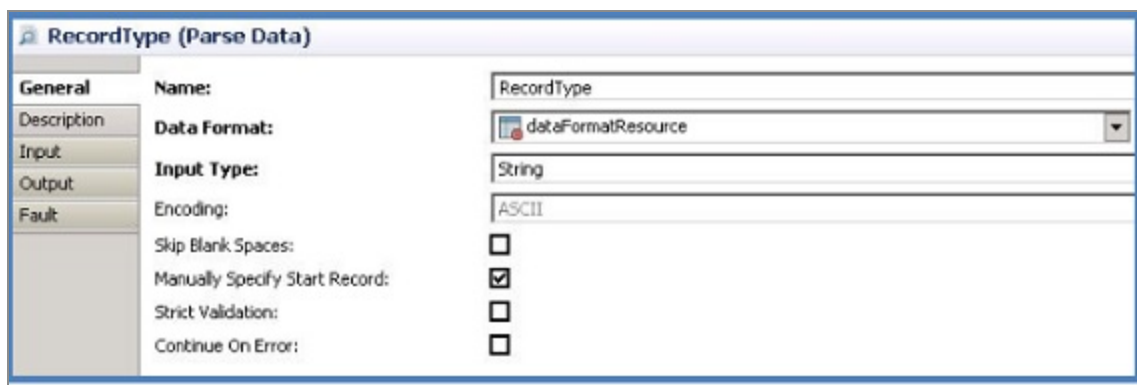
The other option to configure the input type is file. When the input type is file the data is read from the source by means of a pointer. While accessing a specific range of records the search is performed based on the position of the pointer which makes the operation faster.

Testing and Measurement

The testing was focused on the aspects listed below:

- Comparative tests were conducted with input type for the **Parse Data** activity configured to string and file. The tests were performed to parse records in multiple iterations.

- The latency, memory and CPU utilization was measured.

- The overall latency was reduced by almost 10 times when the input type for the **Parse Data** activity was set to file.

Solution for performance improvement

- TIBCO recommends that for faster processing, use the input type as file.

- In case of both the options, the input for **Parse Data** activity is placed in a process variable and this consumes memory. Hence large memory is required to read large number of records. To reduce memory usage, TIBCO recommends that a small set of records are read, parsed and processed before moving on to the next set of records.

Usecase 2: Schema changes for improved performance

In a customer project comprising of multiple schemas (more than 50), it was observed that the latency for a single request -response was high, that is around few seconds.

The project mainly included multiple REST services as shown in the following image.
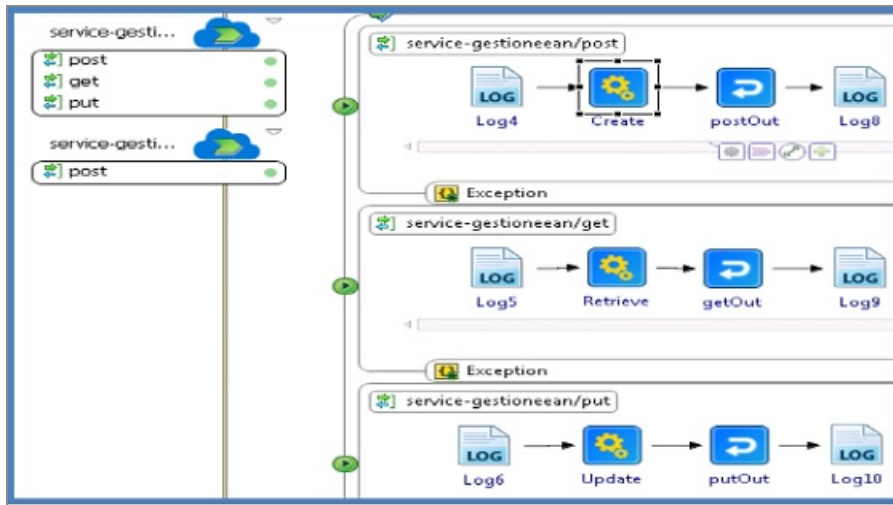
Some of the schemas in the project are shown in the following image.
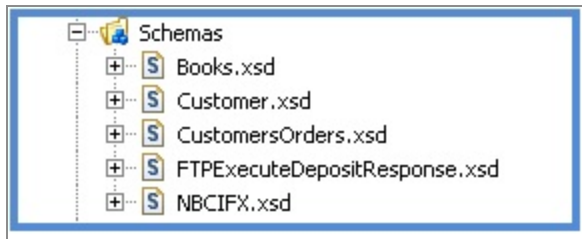
Analysis confirmed that the schema operations were heavy with the current design implementation which contributed to the high latency.

Testing and Measurement

The testing was focused on the aspects listed below:

- With the default settings, a test was run for a single request and the total latency was measured from the logs.

- Few changes were made in the schema definition where the `include` tags were replaced with `import`, and the test was repeated and time was measured. For more information, see

Solution for performance improvement

- If the schemas in the project consist of `include`, these can be replaced with

`imports`as shown in the example in the section, . This reduces the time considerably.

- This design implementation reduced the latency by almost 95 %.

Using XSD Schema Type for the Parse JSON activity

Use Case 3: In a customer usecase slow performance was observed in terms of latency, when the schema type for the **Parse JSON** activity was configured to **XSD** type as compared to **Generic** type. A comparison was done between the **XSD** and **Generic** type of schema for the **Parse JSON** activity.

The process is shown in the following image:
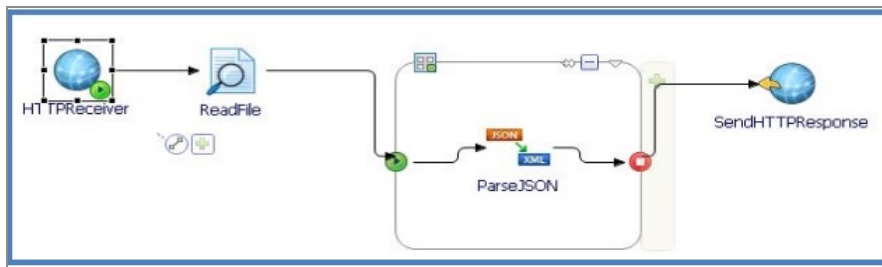
In this process, a JSON file consisting of multiple records or elements is read by the **Read File** activity, parsed by the **Parse JSON** activity in multiple iterations and then converted to XML.

The schema type for the **Parse JSON** activity can be configured to either **XSD** or **Generic** as shown in the following image:
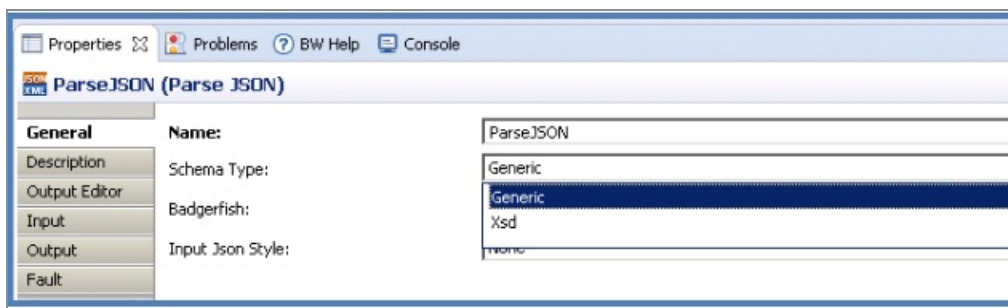
The **Generic** type converts a JSON string to an XML string without using any schema for conversion. The **XSD** type converts a JSON string to an XML document defined using a schema specified in the Output Editor. The user may want to use the **Generic** type a specific schema is not required for conversion or the **XSD** type can be used when conversion needs to be done based on a particular schema.

Testing and Measurement

The testing was focused on the aspects listed below:

- With schema type as **XSD**, tests were run to process the records from the file and the total time was measured for the end to end process to complete.

- With schema type as **Generic**, tests were run to process the records from the file and the total time was measured for the end to end process to complete.

Solution for performance improvement

- It was observed that with the schema type configured to **Generic** the time taken is 50% less than the time taken when the schema type is configured to **XSD**. TIBCO recommends to configure the schema type to **Generic** for better performance.

Disintegrating dependent services from the TIBCO ActiveMatrix BusinessWorks™ service

Use Case 4: A customer use case was designed such that the dependent services were integrated into the same project and multiple service calls were present in a single ActiveMatrix BusinessWorks™ application.
The following image shows one of the processes where there are multiple sub process calls.
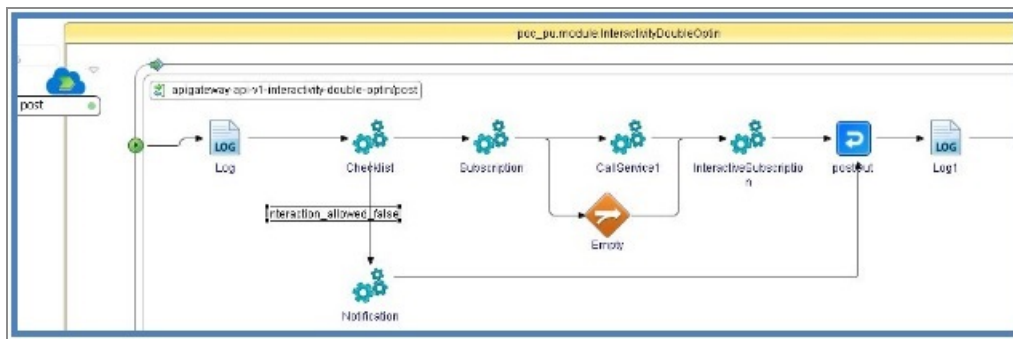
Each of these sub processes eventually calls the dependent services in the form of REST API or HTTP calls. For the given SLAs, the overall performance of the application was extremely low. Test results and analysis showed this was due to the way the application was designed. The dependent services were integrated with the actual ActiveMatrix BusinessWorks services in the same application and the application was deployed on a

single AppNode. Hence, the AppNode resources in terms of CPU, memory and threads were shared between the dependent services and actual ActiveMatrix BusinessWorks service.

Testing and Measurement

The testing was focused on the aspects listed below:

- Load tests were run with increasing concurrency and default thread settings.

- To understand the scalability of the project, the ActiveMatrix BusinessWorks engine threads and HTTP connector threads were tuned and the tests were repeated.

Solution for performance improvement

- The dependent services were created as separate applications. The actual ActiveMatrix BusinessWorks service and dependent services were deployed on separate AppNodes on separate machines. The test results showed improvement in performance with these design changes.

- In order that the actual ActiveMatrix BusinessWorks services have complete resource availability, TIBCO recommends that the dependent services are separated from the actual services and deployed on separate AppNodes. The design changes should be validated and implemented if necessary, whenever resource sharing seems to be an issue between the services.

Additionally, there are two cases that need to be considered when designing the application. One is when the dependent and actual services are part of the same application module and the second is when the dependent services are designed as a shared module.

1. **Same application module** : If both the services are part of the same application module they need to be created as separate processes and both the processes should be deployed on separate AppNodes. This can be achieved in the following way:

   Consider an application module that comprises of two processes P1 and P2. P1 is the main service and P2 is the dependent service. One way of deployment is by creating two AppSpaces with one AppNode in each of the AppSpaces. The application can be deployed in both the AppSpaces. The user can then keep component P1 running in

AppSpace 1 and stop P2. In AppSpace 2 the user can stop P1 and keep P2 running. P1 in AppSpace 1 and P2 in appspace 2 can communicate with each other.

In this way the separate resources are available for both the processes. It is also recommended to ensure that the thread pools are not shared by the activities across the processes.

TIBCO recommends that this approach is tested and validated before deployment.

2. **Shared module** : If the dependent services are part of a shared module they get bundled into the EAR of the main service . Hence, in this case separation of services cannot be achieved.

Changing XSLT Version to Improve Latency

In a customer use case, slower performance was observed in terms of latency, when the XSLT source version for Mapper activity was 1.0 as compared to 2.0. The project is configured to read from an XML file, which is parsed by the Parse XML activity and the output content is further processed by the Mapper activity, as shown in the following image:
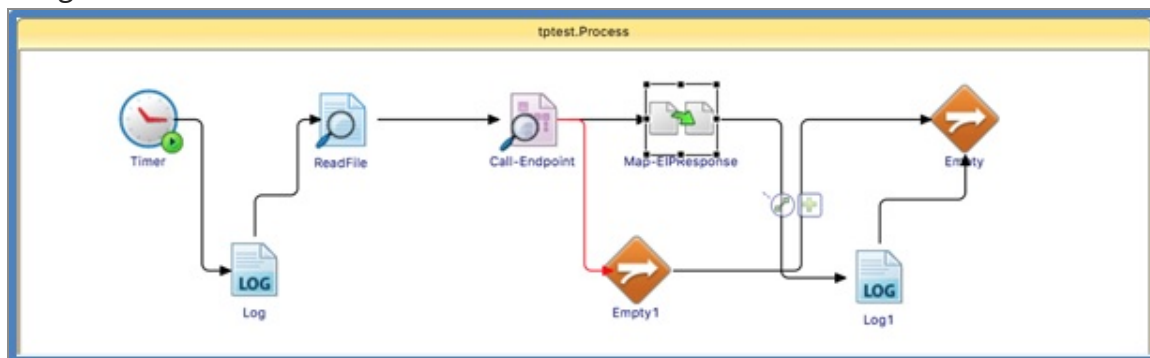
The XSLT source for 'Mapper activity was set to the default 1.0 version as shown in the following image. This version can be configured as follows: Select **Activity** > **Input Tab** > Right-click on the mapping element (RHS) > Select **Show Edit Tab** -> Select **XSLT Source**.
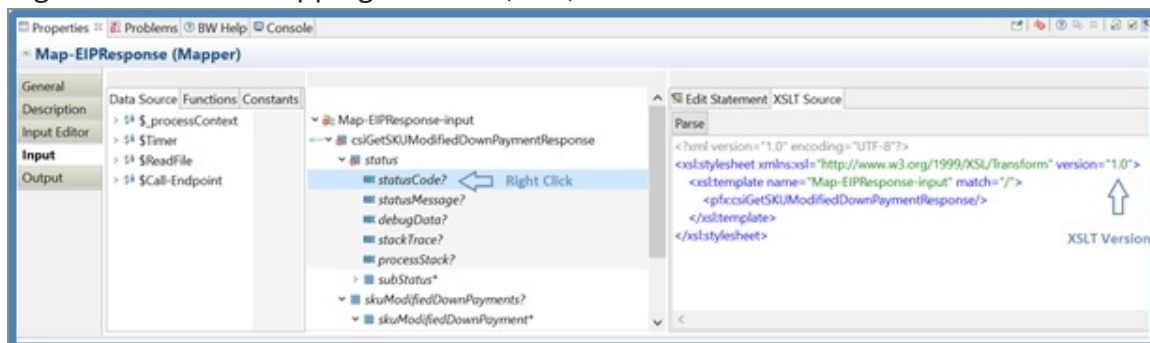
Testing and Measurement

The testing was focused on the aspect below:
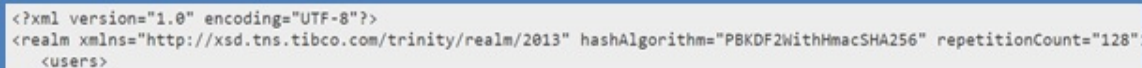
- The tests were run to measure end to latency with XSLT version set to 1.0 and then with 2.0.

Solution for performance improvement

It was observed that the latency improved considerably by 100% when the version was changed to 2.0 from 1.0.

Repetition Count Tuning for XML Authentication Policy

A use case was designed with XML authentication policy. For XML authentication policy, the username and password used during authentication are set in an XML file. This file consists of a parameter called the `repetitionCount`, which is the number of iterations used to compute the hash for the password. The higher the `repetitionCount`, the harder it becomes for an attacker to crack the password. However, using a higher repetition consumes more CPU time during the password verification. The default value is 1000. The following image shows an example of the XML file:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<realm xmlns="http://xsd.tns.tibco.com/trinity/realm/2013" hashAlgorithm="PBKDF2WithHmacSHA256" repetitionCount="128"
    <users>
```

In this particular use case, it was observed that the throughput was low and the service was not scalable although the resources were available.

Testing and Measurement

The testing was focused on the aspects below:

- Load tests were run with a fixed concurrency and the default `repetitionCount` (1000).

- The results provided very low throughput. This was analyzed and the analysis showed that the calls most of the time was spent in the calls related to computing

the hash for the password.

- Since the hashing is determined by the `repetitionCount`, this parameter value was reduced to 1 and the tests were run with the same concurrency.

Solution for Performance Improvement

It was observed that setting the `repetitionCount` to 1 improved the throughput by almost 10 times.

Performance Improvement Use Cases - Run Time

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

Throughput Improvement for SOAP and REST Services

Use Case 1 : The focus of the project was related to scalability and maximizing throughput for a given workload for SOAP and REST services deployed on a single AppNode.

For scenario details, see Performance Improvement Use Case 1 .

The was achieved by tuning certain TIBCO 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 testing was focused on the aspects listed below:

- 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 Performance Improvement : 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.

> ⓘ **Note:** The numbers represented in the graphs are based on tests conducted in the performance lab for ActiveMatrix BusinessWorks 6.3.0. Results may vary according to the components installed, the workload, the type and complexity of different scenarios, hardware, software configuration, and so on. They can be reproduced only under the exact environment and workload conditions that existed when these tests were performed.
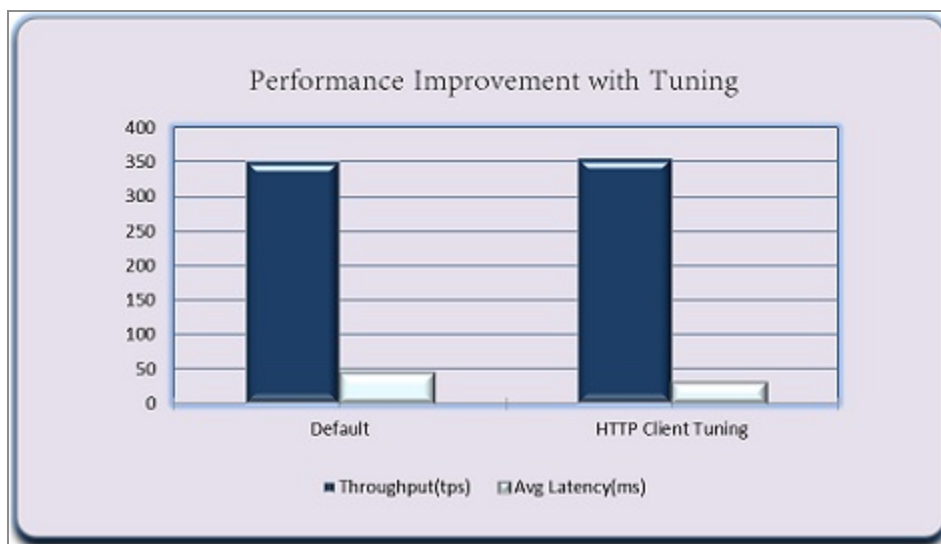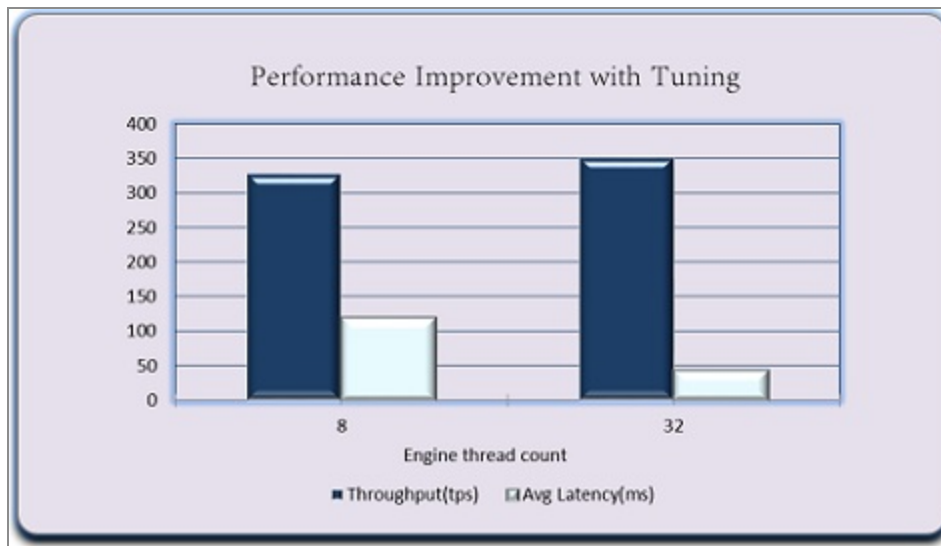
Throughput and Scalability Improvement for SOAP and SOAP with SSL Services

Use Case 2 : Given a TIBCO ActiveMatrix BusinessWorks™ project which comprises mainly of SOAP services, the aim was to benchmark and achieve improvement in terms of throughput, and scalability with maximum resource utilization.

For scenario details, see Performance Improvement Use Case 2.

For hardware details, see Hardware Configuration.

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

- The benchmarking process focuses on maximizing CPU and memory utilization. Benchmarking included running load tests for a given workload.

- Total end-to-end business process completed per second was calculated.

Solution for Performance Improvement

- The event statistics or application metrics feature is known to add some overhead in terms of performance, hence the event statistics were disabled which improved the performance by roughly 30% to 40%. For more information, see Application Statistics.

Configuration of the Swagger port to reduce CPU Utilization

The usecase comprised of multiple REST services. Both the services were deployed on separate AppNodes and each AppNode was created in a separate AppSpace. The AppNodes were configured to have the same Swagger framework port and The details of the Swagger port are provided in the section below.

For REST services, documentation endpoint configuration properties can be specified at the AppSpace or the AppNode level. Properties set at the AppNode level only apply to Applications running on that AppNode. One of the properties at the AppNode level is the Swagger framework port, the property for which is, `bw.rest.docApi.port=7777`

This property specifies the port on which Swagger framework serves the API's documentation endpoint,api-docs etc. For more information, see *TIBCO ActiveMatrix BusinessWorks™ Administration* guide.

The property can be configured in the AppNode `config.ini` file and in the TEA UI as shown below:

Parameter location = **AppNodes > Select AppNode > Configure > General**

Default Value = 7777

Testing and measurement

Testing was focused on the following aspects:

- The two REST services were deployed on separate AppNodes and started.

- No requests or load was sent to the services. The CPU usage of the AppNodes was continuously monitored.

- One of the AppNode showed continuous CPU activity without any load, even after stopping the application. This was because the same Swagger port was being used by the AppNodes.

Solution for performance improvement

- The Swagger framework port for one of the AppNodes was changed from 7777 to another value.

- This configuration change stops the CPU activity when the applications are in the stopped/running state.

- In the case of multiple REST services deployed to multiple AppNodes, TIBCO recommends to use unique Swagger ports for each AppNode.

Performance Improvement Use Case 1

One of the processes from Use Case 1 is shown in the image below. The complete scenario comprises of multiple processes, and each process has a combination of SOAP and REST services deployed on a single AppNode. The image below explains one of the process.

Performance Improvement Use Case 2

SOAP(HTTPS) services in Use Case 2 are configured as shown in the images below.

The modified elements are highlighted below. The `include` tags were replaced with `import namespace` in the schema definition.

Original Schema

```
<schema xmlns="http://www.w3.org/2001/XMLSchema"
xmlns:tns="http://carrefour.it/mdg/schemas/xsd"
elementFormDefault="qualified"
targetNamespace="http://carrefour.it/mdg/schemas/xsd">
<include schemaLocation="InfoLog.xsd"/>
    <include schemaLocation="Context.xsd"/>
    <include schemaLocation="Pagination.xsd"/>
    <include schemaLocation="Pos.xsd"/>
    <include schemaLocation="Response.xsd"/>
```

Modified Schema

```
<schema xmlns="http://www.w3.org/2001/XMLSchema"
xmlns:tns="http://carrefour.it3/mdg/schemas/xsd"
xmlns:tns1="http://carrefour.it/mdg/schemas/xsd"
elementFormDefault="qualified"
targetNamespace="http://carrefour.it3/mdg/schemas/xsd">
    <import namespace="http://carrefour.it/mdg/schemas/xsd"
schemaLocation="Infolog.xsd"/>
    <import namespace="http://carrefour.it/mdg/schemas/xsd"
schemaLocation="Context.xsd"/>
    <import namespace="http://carrefour.it/mdg/schemas/xsd"
schemaLocation="Pagination.xsd"/>
    <import namespace="http://carrefour.it/mdg/schemas/xsd"
schemaLocation="Pos.xsd"/>
    <import namespace="http://carrefour.it/mdg/schemas/xsd"
schemaLocation="Response.xsd"/>
```

Performance Improvement Use Case 2 - Schema changes for better performance

The modified elements are highlighted below. The `include` tags were replaced with `import namespace` in the schema definition.

Original Schema

```
<schema xmlns="http://www.w3.org/2001/XMLSchema"
xmlns:tns="http://carrefour.it/mdg/schemas/xsd"
elementFormDefault="qualified"
targetNamespace="http://carrefour.it/mdg/schemas/xsd">
<include schemaLocation="InfoLog.xsd"/>
```

```
<include schemaLocation="Context.xsd"/>
<include schemaLocation="Pagination.xsd"/>
<include schemaLocation="Pos.xsd"/>
<include schemaLocation="Response.xsd"/>
```

Modified Schema

```
<schema xmlns="http://www.w3.org/2001/XMLSchema"
xmlns:tns="http://carrefour.it3/mdg/schemas/xsd"
xmlns:tns1="http://carrefour.it/mdg/schemas/xsd"
elementFormDefault="qualified"
targetNamespace="http://carrefour.it3/mdg/schemas/xsd">
    <import namespace="http://carrefour.it/mdg/schemas/xsd"
schemaLocation="Infolog.xsd"/>
    <import namespace="http://carrefour.it/mdg/schemas/xsd"
schemaLocation="Context.xsd"/>
    <import namespace="http://carrefour.it/mdg/schemas/xsd"
schemaLocation="Pagination.xsd"/>
    <import namespace="http://carrefour.it/mdg/schemas/xsd"
schemaLocation="Pos.xsd"/>
    <import namespace="http://carrefour.it/mdg/schemas/xsd"
schemaLocation="Response.xsd"/>
```

Tools for Memory Monitoring, Tracking, and Analysis

For monitoring, tracking, and analyzing memory usage, the following utilities are available.

TOP Command for Memory Monitoring

The `top` command is used for memory monitoring. It works only on Linux platform.

The `top` command produces an ordered list of running processes selected by user-specified criteria, and updates it periodically. By default, ordering is by CPU usage, and it shows processes that consume maximum CPU. The `top` command also shows how much processing power and memory are being used, as well as the other information about the running processes.

> **ⓘ Note:** This utility works on Linux OS only.

The `top` command output monitors the RSS memory as well as the CPU utilization of the ActiveMatrix BusinessWorks AppNode.

```
top -p PID > top.txt
```

Sample output is as follows:

```
Cpu(s):  4.7%us,  1.1%sy,  0.0%ni, 94.1%id,  0.0%wa,  0.0%hi,  0.1%si,
0.0%st
Mem:  65914304k total, 59840516k used,  6073788k free,  3637208k buffers
Swap: 15359996k total,   119216k used, 15240780k free, 43597120k cached

  PID USER     PR  NI  VIRT  RES  SHR S %CPU %MEM     TIME+  COMMAND


  925 root     20   0 3650m 997m  27m S  4.0  1.6 238:05.72 bwappnode-Http
```

Press 1 on same top output window and it would give usage per core

```
top - 02:13:25 up 160 days, 15:16, 26 users,  load average: 0.00, 0.00,
0.00
Tasks:   1 total,   0 running,   1 sleeping,   0 stopped,   0 zombie
Cpu0  :  8.3%us,  6.7%sy,  0.0%ni, 85.0%id,  0.0%wa,  0.0%hi,  0.0%si,
0.0%st
Cpu1  :  4.7%us,  0.5%sy,  0.0%ni, 94.8%id,  0.0%wa,  0.0%hi,  0.0%si,
0.0%st
Cpu2  :  4.2%us,  0.4%sy,  0.0%ni, 95.1%id,  0.0%wa,  0.0%hi,  0.3%si,
0.0%st
Cpu3  :  3.8%us,  0.4%sy,  0.0%ni, 95.8%id,  0.0%wa,  0.0%hi,  0.0%si,
0.0%st

Mem:  65914304k total, 59839448k used,  6074856k free,  3637208k buffers
Swap: 15359996k total,   119216k used, 15240780k free, 43597124k cached

  PID USER      PR  NI  VIRT  RES  SHR S %CPU %MEM     TIME+  COMMAND



  925 root      20   0 3650m 997m  27m S  4.7  1.6 238:08.62 bwappnode-Http
```

Native Memory Tracking

The Native Memory Tracking (NMT) is a Java HotSpot VM feature that tracks internal memory usage for a Java HotSpot VM.

> **ℹ Note:** This utility works on Windows OS and Linux OS.

For more information about Native Memory Tracking, see docs.oracle.com.

Before you begin

Start the JVM with summary tracking using the command line option –
XX:NativeMemoryTracking=summary. Set the property in the AppNode.tra file. For example,
java.extended.properties=-Xmx1024m –Xms128m –XX:NativeMemoryTracking=summary

Procedure

1. To enable native memory tracking on the JVM, set the parameter –

```
XX:NativeMemoryTracking=summary.
```

2. Establish an early baseline. Use NMT baseline feature to get a baseline to compare by running command: `jcmd <pid> VM.native_memory baseline`.

3. Collect the memory data after starting the test runs by running `jcmd <pid> VM.native_memory summary`.

4. To monitor memory changes, use the following command: `jcmd <pid> VM.native_memory summary.diff`

5. If the application leaks a small amount of memory, it takes a while to show up. Comparing the memory pools from NMT output help identify the memory pool contributing to increase in memory.

6. The jcmd utility is shipped with JDK. It is under *$JDK_HOME*/bin directory.

   For more information about jcmd utility, see docs.oracle.com

Jemalloc and Jeprof

The jemalloc tool tracks down native memory allocations and identifies native memory leak suspects. Use jeprof utility to analyze heap files.

> **ⓘ** **Note:** The jemalloc and jeprof utilities work on Linux OS only.

Procedure

1. Download jemalloc from GitHub.com.

2. Install the tool by following the steps specified at GitHub.com.

3. Set the following properties and start the BW AppNode from the same console:

```
cd [BW_
HOME]/bw/6.5/domains/<DomainName>/appnodes/<AppSpaceName>/<AppNodeN
ame>/bin
```

```
export LD_PRELOAD=/root/jemalloc/jemalloc-stable-
4/lib/libjemalloc.so
```

```
export MALLOC_CONF=prof:true,lg_prof_interval:30,lg_prof_
sample:17,prof_final:true,prof_leak:true
```

```
./startBWAppNode.sh
```

4. Once jemalloc is built, include jemalloc in BWCE buildpack.

5. Edit the bwce-buildpack/resources/prestart.sh file and add the following commands before the `exec ./tibco.home/bw*/*/bin/startBWAppNode.sh` command:

```
export LD_PRELOAD=/root/jemalloc/jemalloc-stable-
4/lib/libjemalloc.so
```

```
export MALLOC_CONF=prof:true,lg_prof_interval:30,lg_prof_
sample:17,prof_final:true,prof_leak:true
```

6. When the server is started and the memory allocation is done, `jeprof*.heap` files are generated in same *{BW_ HOME}* `/bw/6.5/domains/<DomainName>/appnodes/<AppSpaceName>/<AppNodeName>/bin` folder.

As the memory utilization grows, more files would be generated.

7. Analyze the heap files with the `jeprof` command.

```
jeprof --show_bytes <PATH to java> jeprof.*.heap
```

The jeprof utility is included in the `jemalloc\bin` folder. After execution of jeprof command, jeprof console opens.

8. Type top on the jeprof console. For example:

```
jeprof --show_bytes /usr/lib/jvm/java-8-oracle/jre/bin/java
jeprof*.heap
Using local file /usr/bin/w.
Using local file jeprof.19678.0.f.heap.
Welcome to jeprof! For help, type 'help'.
```

```
(jeprof) top
```

It shows the following output:

```
 Total: 267184 B
258032 96.6% 96.6% 258032 96.6% _3_2_5
3616 1.4% 97.9% 3616 1.4% _nl_intern_locale_data
2048 0.8% 98.7% 2208 0.8% __tzfile_read
1024 0.4% 99.1% 1024 0.4% getpwnam
1024 0.4% 99.5% 1072 0.4% getpwuid
448 0.2% 99.6% 448 0.2% __gconv_lookup_cache
224 0.1% 99.9% 224 0.1% strdup
160 0.1% 99.9% 160 0.1% __tzstring
128 0.0% 100.0% 3760 1.4% _nl_load_locale_from_archive
48 0.0% 100.0% 48 0.0% get_mapping
```

9. To run the jeprof command on a single file, use the following command:

```
jeprof --show_bytes <PATH to java> <Heap file name>
```

After execution of jeprof command, jeprof console opens.

10. Type top on the jeprof console. For example:

```
jeprof --show_bytes /usr/lib/jvm/java-8-oracle/jre/bin/java
jeprof.19678.0.f.heap
 Using local file /usr/bin/w.
 Using local file jeprof.19678.0.f.heap.
Welcome to jeprof! For help, type 'help'.
 (jeprof) top
```

It shows the following output:

```
Total: 267184 B
258032 96.6% 96.6% 258032 96.6% _3_2_5
3616 1.4% 97.9% 3616 1.4% _nl_intern_locale_data
2048 0.8% 98.7% 2208 0.8% __tzfile_read
1024 0.4% 99.1% 1024 0.4% getpwnam
1024 0.4% 99.5% 1072 0.4% getpwuid
448 0.2% 99.6% 448 0.2% __gconv_lookup_cache
```

```
224 0.1% 99.9% 224 0.1% strdup
160 0.1% 99.9% 160 0.1% __tzstring
128 0.0% 100.0% 3760 1.4% _nl_load_locale_from_archive
48 0.0% 100.0% 48 0.0% get_mapping
```

11. To stop profiling once the analysis is done and leak suspects are identified run command:

```
unset MALLOC_CONF
```

If profiling is not stopped, the jeprof heap files are continuously generated.

Detecting Increase in Heap Allocations with UMDH

The user-mode dump heap (UMDH) utility works with the Windows operating system to analyze the heap allocations for a specific process. UMDH utility is used to locate which routine in a specific process is leaking memory.

> **ⓘ Note:** The UMDH utility works on Windows OS only.

Before you begin
- Download and install the UMDH utility for Windows OS. For more information, see Debugging Tools for Windows from Microsoft documentation.

- Enable "Create user mode stack trace database" with `gflags.exe -i bwappnode-umdh.exe +ust` command. Get the process name from task manager.

```
C:\Program Files (x86)\Windows Kits\10\Debuggers\x64>gflags.exe -i
bwappnode-umdh.exe +ust
Current Registry Settings for bwappnode-umdh.exe executable are:
00001000
        ust - Create user mode stack trace database
```

- Before using UMDH, you must have access to the proper symbols for your application. UMDH uses the symbol path specified by the environment variable _NT_

SYMBOL_PATH. Set the variable to a path containing symbols for your application.

If you also include a path to Windows symbols, the analysis is more complete. The syntax for this symbol path is the same as that used by the debugger.

For more information, see Symbol Path for Windows Debugger from Microsoft documentation.

For example, if the symbols for your application are located at `C:\MySymbols`, then to use the public Microsoft symbol store for your Windows symbols, using `C:\MyCache` as your downstream store, run the following command to set your symbol path:

```
C:\Program Files (x86)\Windows Kits\10\Debuggers\x64>set _NT_
SYMBOL_
PATH=c:\mysymbols;srv*c:\mycache*https://msdl.microsoft.com/downloa
d/symbols
```

Procedure

1. Determine the process ID (PID) for the process to investigate.

   For more information, see Finding the process ID from Microsoft documentation.

2. Analyze the heap memory allocations for the process before the memory leak is detected, and save it to a log file.

3. Collect the data at application start up before sending load.

   For example, if the PID is 5872, and name of the log file is `log_before.txt`, use the following command:

   ```
   umdh.exe -p:5872 -f:log_before.txt
   ```

4. Use the UMDH utility to analyze the heap memory allocations for this process after the memory starts increasing, and save it to a log file.

5. Collect this data at regular intervals when an application starts leaking memory.

   For example, if the PID is 5872, and name the log file is `log_after.txt`, use the following command:

   ```
   umdh.exe -p:5872 -f:log_after.txt
   ```

6. The UMDH utility can compare two different log files and display the change in their respective allocation sizes. To redirect the results into a third text file, use the greater-than symbol (`>`). To convert the byte and allocation counts from hexadecimal to decimal, use the -d option.

   For example, to compare `log_before.txt` and `log_after.txt` files, and save the

results to the file `log_compare.txt`, use the following command:

```
umdh.exe -d log_before.txt log_after.txt > log_compare.txt
```

7. For each call stack that is labeled as "BackTrace" in the UMDH log files, there is a comparison made between the two log files. The snippet of the output is as follows:

```
 // where:

 //     BYTES_DELTA - increase in bytes between before and after log

 //     NEW_BYTES - bytes in after log

 //     OLD_BYTES - bytes in before log

 //     COUNT_DELTA - increase in allocations between before and
after log
 //     NEW_COUNT - number of allocations in after log

 //     OLD_COUNT - number of allocations in before log

 //     TRACEID - decimal index of the stack trace in the trace
database
 //         (can be used to search for allocation instances in the
original
 //         UMDH logs).


 +    8856 (  18400 -   9544)     12 allocs     BackTrace5
 +       3 (     12 -      9)   BackTrace5     allocations

       ntdll!RtlpAllocateHeap+2298
       ntdll!RtlpAllocateHeapInternal+727
       MSVCR100!malloc+5B
       jvm!JVM_ResolveClass+387AE
       jvm!???+0 : 53C415D6
       jvm!JVM_GetManagementExt+6A5FF
       jvm!JVM_GetManagementExt+786B1
       jvm!JVM_GetManagementExt+7A162
       jvm!JVM_GetManagementExt+CA4E
       jvm!JVM_FindSignal+178329
       jvm!JVM_FindSignal+1792E4
       jvm!JVM_FindSignal+179491
       jvm!JVM_FindSignal+17969F
       jvm!JVM_GetManagementExt+82712
       jvm!JVM_GetManagementExt+8305F
```

```
jvm!JVM_ResolveClass+5F5FF
jvm!JVM_FindSignal+68FA
MSVCR100!endthreadex+43
MSVCR100!endthreadex+DF
KERNEL32!BaseThreadInitThunk+14
ntdll!RtlUserThreadStart+21
```

This UMDH output shows that there were 8856 total bytes allocated from the call stack.

VMMap

VMMap is an utility to perform analysis of process virtual and physical memory.

This utility shows a breakdown of a process's committed virtual memory types and the amount of physical memory assigned by the operating system to those types. Use VMMap utility to show graphical representations of memory usage, summary information, and a detailed process memory map. Powerful filtering and refresh capabilities allow you to identify the sources of process memory usage and the memory cost of application features. VMMap utility works on Windows platform.

You can download the VMMap utility from Microsoft.com.

Memory Saving Mode

All activity output variables in a running process instance are stored in a memory, and hence consume memory. Memory saving mode allows memory used by an activity output variable to be released when the value of the variable is no longer needed.

In a memory saving mode, as each activity runs, the list of activity output variables is continuously evaluated to determine if subsequent activities in the process refer to the specific activity output variable. If no activities refer to the activity output variable, the memory used by it is released.

Memory Saving Mode can reduce the memory used by actively running process instances, as well as potentially improve the performance of checkpoints. By default, memory saving mode is enabled. This property enables the usage of memory saving mode, which frees activity output variables once they are no longer needed. The default value is true.

Memory saving is enabled at design-time and run-time.

To disable the Memory Saving Mode:

- For design-time: To disable the memory saving mode, unselect the **Save information to support memory saving mode** checkbox available at **Window > Preferences > BusinessWorks > Process Diagram** in the Memory Saving Mode section. Then, to remove the memory saving variable, right-click on **ActiveMatrix BusinessWorks™ Projects** and select **Refactor > Repair BusinessWorks Projects**. In the dialog, select the **Remove memory saving variables** option. On clicking the **Preview** button, the variables that can be removed from different activities are displayed on the Preview page, then select **OK**.

- For run-time: Configure the following `bwengine` property in the `BW_JAVA_OPTS` environment variable while running the application to disable the Memory Saving Mode:

  `bw.engine.enable.memory.saving.mode=false.`

Performance Use Case - Memory Optimization

The service under test was a REST implementation with other activities like Mapper, Render JSON, Parse JSON, and Invoke REST API. The implementation had multiple mapper activities with iterator and accumulate output. The max heap of the AppNode was set to 4 GB for these tests. The testing was focused on analyzing the memory usage of the AppNode under load.

Memory usage with `bw.engine.enable.memory.saving.mode` **set to false**

The following snapshot shows the JVM snapshot of the AppNode under heavy load with memory saving set to false.

Memory usage with `bw.engine.enable.memory.saving.mode` **set to true**

The following snapshot shows the JVM snapshot of the AppNode under heavy load with memory saving set to true.
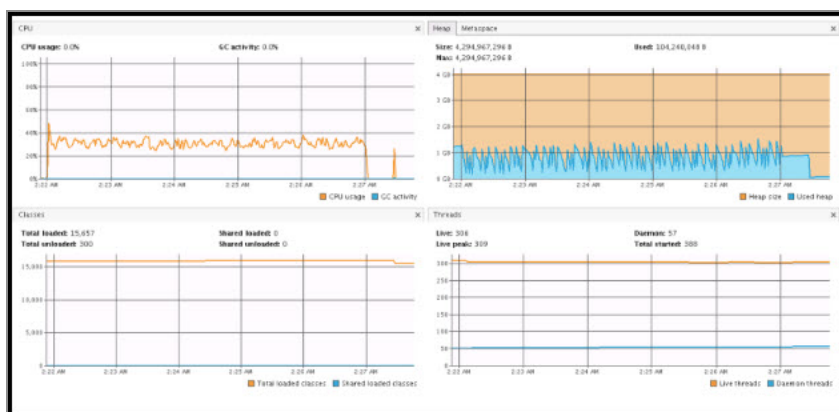
Performance findings

- Enabling memory saving reduced the heap usage of the application under heavy load.

- Enabling memory saving did not degrade the performance of deployed services in terms of latency and throughput.

> **Note:** The improvements showcased must be used as reference. The performance impact of enabling memory saving mode may vary based on service implementation, workload, and payload on the system.

Hardware Configuration

The following hardware configurations were used for tests conducted in the performance lab to establish benchmark results.

Components	Machine 1 (BW AppNode)	Machine 2 (Load Generator)
Server Class	ProLiant HP BL460c G1 (C-Class)	ProLiant HP BL460c G1 (C-Class)
Chip Vendor	Intel	Intel
CPU Model	Intel XEON	Intel XEON
Total Number Of Processors	2	2
Cores Per Processor	6	6
Threads Per Processor	2	2
Clock	1.6 GHz	1.6 GHz
Memory	32 GB	64 GB
OS	Linux 6.2 (Santiago)	Linux 6.2 (Santiago)
L2 Cache	1,333 MHz	1,333 MHz
Network	Gigabit	Gigabit

Appendix: A

This section describes the scenarios that were tested for performance and have been illustrated in the sections, ActiveMatrix BusinessWorks Transport and Resource Tuning Guidelines, and Performance Improvement Use Cases.

HTTP Receiver and HTTP Response

The project configuration for the HTTP Receiver and HTTP Response is shown in the image below.
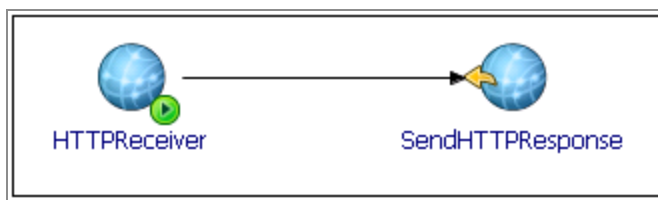
SOAP Service

The project configuration for SOAP(HTTP) service is shown in the image below.
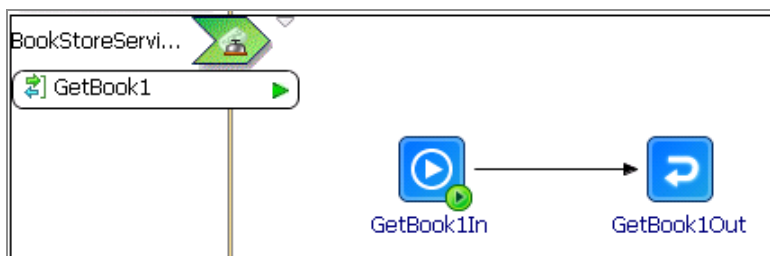
HTTP with SSL Configuration [HTTPS]

The project configuration for the HTTP Server is shown in the image below.
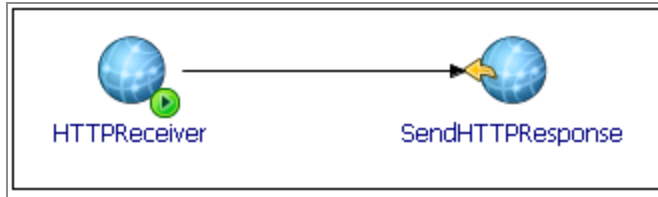
The HTTP server is configured with SSL as shown in the image below.

SOAP HTTP with SSL Configuration [HTTPS]

The project configuration for the SOAP(HTTPS) service is shown in the image below.
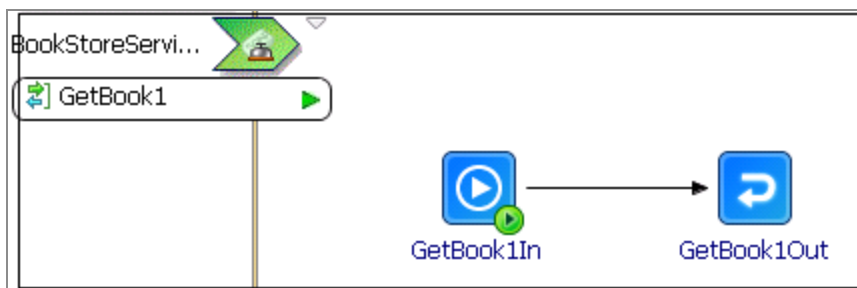
The SOAP(HTTP) service is configured with SSL as shown in the image below.

Performance Improvement Use Case 1

One of the processes from Use Case 1 is shown in the image below. The complete scenario comprises of multiple processes, and each process has a combination of SOAP and REST services deployed on a single AppNode. The image below explains one of the process.

Performance Improvement Use Case 2

SOAP(HTTPS) services in Use Case 2 are configured as shown in the images below.

The modified elements are highlighted below. The `include` tags were replaced with `import` `namespace` in the schema definition.

Original Schema

```
<schema xmlns="http://www.w3.org/2001/XMLSchema"
xmlns:tns="http://carrefour.it/mdg/schemas/xsd"
elementFormDefault="qualified"
targetNamespace="http://carrefour.it/mdg/schemas/xsd">
<include schemaLocation="InfoLog.xsd"/>
    <include schemaLocation="Context.xsd"/>
    <include schemaLocation="Pagination.xsd"/>
    <include schemaLocation="Pos.xsd"/>
    <include schemaLocation="Response.xsd"/>
```

Modified Schema

```
<schema xmlns="http://www.w3.org/2001/XMLSchema"
xmlns:tns="http://carrefour.it3/mdg/schemas/xsd"
xmlns:tns1="http://carrefour.it/mdg/schemas/xsd"
```

```
elementFormDefault="qualified"
targetNamespace="http://carrefour.it3/mdg/schemas/xsd">
    <import namespace="http://carrefour.it/mdg/schemas/xsd"
schemaLocation="Infolog.xsd"/>
    <import namespace="http://carrefour.it/mdg/schemas/xsd"
schemaLocation="Context.xsd"/>
    <import namespace="http://carrefour.it/mdg/schemas/xsd"
schemaLocation="Pagination.xsd"/>
    <import namespace="http://carrefour.it/mdg/schemas/xsd"
schemaLocation="Pos.xsd"/>
    <import namespace="http://carrefour.it/mdg/schemas/xsd"
schemaLocation="Response.xsd"/>
```

Performance Improvement Use Case 2 - Schema changes for better performance

The modified elements are highlighted below. The `include` tags were replaced with `import namespace` in the schema definition.

Original Schema

```
<schema xmlns="http://www.w3.org/2001/XMLSchema"
xmlns:tns="http://carrefour.it/mdg/schemas/xsd"
elementFormDefault="qualified"
targetNamespace="http://carrefour.it/mdg/schemas/xsd">
<include schemaLocation="InfoLog.xsd"/>
    <include schemaLocation="Context.xsd"/>
    <include schemaLocation="Pagination.xsd"/>
    <include schemaLocation="Pos.xsd"/>
    <include schemaLocation="Response.xsd"/>
```

Modified Schema

```
<schema xmlns="http://www.w3.org/2001/XMLSchema"
xmlns:tns="http://carrefour.it3/mdg/schemas/xsd"
xmlns:tns1="http://carrefour.it/mdg/schemas/xsd"
elementFormDefault="qualified"
targetNamespace="http://carrefour.it3/mdg/schemas/xsd">
    <import namespace="http://carrefour.it/mdg/schemas/xsd"
schemaLocation="Infolog.xsd"/>
    <import namespace="http://carrefour.it/mdg/schemas/xsd"
```

```
schemaLocation="Context.xsd"/>
    <import namespace="http://carrefour.it/mdg/schemas/xsd"
schemaLocation="Pagination.xsd"/>
    <import namespace="http://carrefour.it/mdg/schemas/xsd"
schemaLocation="Pos.xsd"/>
    <import namespace="http://carrefour.it/mdg/schemas/xsd"
schemaLocation="Response.xsd"/>
```

References

- High Load: https://wiki.eclipse.org/Jetty/Howto/High_Load

- Saw tooth Shaped Graph: http://stackoverflow.com/questions/7219532/why-a-sawtooth-shaped-graph

- jvmtop utility: https://github.com/patric-r/jvmtop

- Debugging Java Native Memory Leaks: https://www.evanjones.ca/java-native-leak-bug.html

- Tracking Down Native Memory Leaks in Elasticsearch: https://www.elastic.co/blog/tracking-down-native-memory-leaks-in-elasticsearch

- Discrete sequential memory leak analysis with jemalloc and jeperf: https://www.igorkromin.net/index.php/2018/06/21/discrete-sequential-memory-leak-analysis-with-jemalloc-jeperf/

TIBCO Documentation and Support Services

For information about this product, you can read the documentation, contact TIBCO Support, and join TIBCO Community.

How to Access TIBCO Documentation

Documentation for TIBCO products is available on the Product Documentation website, mainly in HTML and PDF formats.

The Product Documentation website is updated frequently and is more current than any other documentation included with the product.

Product-Specific Documentation

The following documentation for this product is available on the TIBCO ActiveMatrix BusinessWorks™ page:

- *TIBCO ActiveMatrix BusinessWorks™ Release Notes*

- *TIBCO ActiveMatrix BusinessWorks™ Installation*

- *TIBCO ActiveMatrix BusinessWorks™ Application Development*

- *TIBCO ActiveMatrix BusinessWorks™ Bindings and Palettes Reference*

- *TIBCO ActiveMatrix BusinessWorks™ Concepts*

- *TIBCO ActiveMatrix BusinessWorks™ Error Codes*

- *TIBCO ActiveMatrix BusinessWorks™ Getting Started*

- *TIBCO ActiveMatrix BusinessWorks™ Maven Plug-in*

- *TIBCO ActiveMatrix BusinessWorks™ Migration*

- *TIBCO ActiveMatrix BusinessWorks™ Performance Benchmarking and Tuning*

- *TIBCO ActiveMatrix BusinessWorks™ REST Implementation*

- *TIBCO ActiveMatrix BusinessWorks™ Refactoring Best Practices*

- *TIBCO ActiveMatrix BusinessWorks™ Samples*

How to Contact Support for TIBCO Products

You can contact the Support team in the following ways:

- To access the Support Knowledge Base and getting personalized content about products you are interested in, visit our product Support website.

- To create a Support case, you must have a valid maintenance or support contract with a Cloud Software Group entity. You also need a username and password to log in to the product Support website. If you do not have a username, you can request one by clicking **Register** on the website.

How to Join TIBCO Community

TIBCO Community is the official channel for TIBCO customers, partners, and employee subject matter experts to share and access their collective experience. TIBCO Community offers access to Q&A forums, product wikis, and best practices. It also offers access to extensions, adapters, solution accelerators, and tools that extend and enable customers to gain full value from TIBCO products. In addition, users can submit and vote on feature requests from within the TIBCO Ideas Portal. For a free registration, go to TIBCO Community.

Legal and Third-Party Notices

SOME CLOUD SOFTWARE GROUP, INC. ("CLOUD SG") SOFTWARE AND CLOUD SERVICES EMBED, BUNDLE, OR OTHERWISE INCLUDE OTHER SOFTWARE, INCLUDING OTHER CLOUD SG SOFTWARE (COLLECTIVELY, "INCLUDED SOFTWARE"). USE OF INCLUDED SOFTWARE IS SOLELY TO ENABLE THE FUNCTIONALITY (OR PROVIDE LIMITED ADD-ON FUNCTIONALITY) OF THE LICENSED CLOUD SG SOFTWARE AND/OR CLOUD SERVICES. THE INCLUDED SOFTWARE IS NOT LICENSED TO BE USED OR ACCESSED BY ANY OTHER CLOUD SG SOFTWARE AND/OR CLOUD SERVICES OR FOR ANY OTHER PURPOSE.

USE OF CLOUD SG SOFTWARE AND CLOUD SERVICES IS SUBJECT TO THE TERMS AND CONDITIONS OF AN AGREEMENT FOUND IN EITHER A SEPARATELY EXECUTED AGREEMENT, OR, IF THERE IS NO SUCH SEPARATE AGREEMENT, THE CLICKWRAP END USER AGREEMENT WHICH IS DISPLAYED WHEN ACCESSING, DOWNLOADING, OR INSTALLING THE SOFTWARE OR CLOUD SERVICES (AND WHICH IS DUPLICATED IN THE LICENSE FILE) OR IF THERE IS NO SUCH LICENSE AGREEMENT OR CLICKWRAP END USER AGREEMENT, THE LICENSE(S) LOCATED IN THE "LICENSE" FILE(S) OF THE SOFTWARE. USE OF THIS DOCUMENT IS SUBJECT TO THOSE SAME TERMS AND CONDITIONS, AND YOUR USE HEREOF SHALL CONSTITUTE ACCEPTANCE OF AND AN AGREEMENT TO BE BOUND BY THE SAME.

This document 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 Cloud Software Group, Inc.

TIBCO, the TIBCO logo, the TIBCO O logo, ActiveMatrix BusinessWorks, ActiveSpaces, Business Studio, TIBCO Business Studio, TIBCO Designer, TIBCO Enterprise Administrator, Enterprise Message Service, Rendezvous, and TIBCO Runtime Agent are either registered trademarks or trademarks of Cloud Software Group, Inc. in the United States and/or 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. You acknowledge that all rights to these third party marks are the exclusive property of their respective owners. Please refer to Cloud SG's Third Party Trademark Notices (https://www.cloud.com/legal) for more information.

This document includes fonts that are licensed under the SIL Open Font License, Version 1.1, which is available at: https://scripts.sil.org/OFL

Copyright (c) Paul D. Hunt, with Reserved Font Name Source Sans Pro and Source Code Pro.

Cloud SG 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 a specific version of Cloud SG software on a specific operating system platform.