Business Integration
Software

TIBCO BusinessWorks Performance Best Practices

This document describes architectural concepts related to performance tuning for TIBCO BusinessWorks 5.x. Included are different tuning parameters, steps required to configure parameters, and best design techniques for better performance. The reader should focus on a real customer use case to understand different issues and solutions associated with each. The last section in this paper describes a typical performance management process including critical performance criteria, debugging and monitoring a typical project.

This document is designed for TIBCO customers, TIBCO architects, and TIBCO sales consultants who are evaluating products for performance.

Prior knowledge of TIBCO BusinessWorks concepts is necessary for the best use of this document.

This document supplements the TIBCO BusinessWorks document set and can be distributed to TIBCO customers and partners.

TIBCO®
The Power of Now®

http://www.tibco.com

Global Headquarters
3303 Hillview Avenue
Palo Alto, CA 94304
Tel: +1 650-846-1000
Toll Free: 1 800-420-8450
Fax: +1 650-846-1005

For more information, please contact:

TIBCO Software Inc.
3301 Hillview Avenue
Palo Alto, CA 94304
USA

Table of Contents

1 Introduction

Most large TIBCO customers have a group that functions as an Integration Competency Center (ICC). Although the function of the group and its name differs from one company to another, the ICC is normally responsible for defining integration project guidelines, matrixes, and processes to measure and tune the performance of enterprise systems before going into production. Even medium and small deployments must have systematic analysis of performance. Careful performance analysis leads to a better understanding of throughput and response time. Analysis of predictable results helps define Service Level Agreements (SLAs) and assists in capacity planning.

The general goals for performance analysis are to find optimal response times (for e-commerce applications), lowest latency, and highest throughput for integration application with highest use of CPU time when multiple processors, machines, and engines are used. The goal of this exercise is to find this highest yield point; in other words, optimal performance.

TIBCO products deliver more than simple point solutions such as Integration, Business Process Management (BPM) and Business Activity Monitoring (BAM). The highest return comes when TIBCO delivers highly scalable solutions in large deployment environments.

2 TIBCO BusinessWorks 5.2 Performance Architecture

2.1 Main TIBCO BusinessWorks Performance Components

The most important component in TIBCO BusinessWorks is the TIBCO BusinessWorks Engine. Its purpose is to handle a continuous stream of thousands of processes, each with dozens of activities, in an operating environment with finite resources (memory, CPUs, threads, connections). The TIBCO BusinessWorks Engine also schedules jobs and gives each an opportunity to execute. It provides these additional functions:

- XML data transformation and validation
- XPath transitions and flow control
- Connection/session management with recovery/retries
- Engine crash and job recovery
- Exception management and logging
- Management and monitoring services

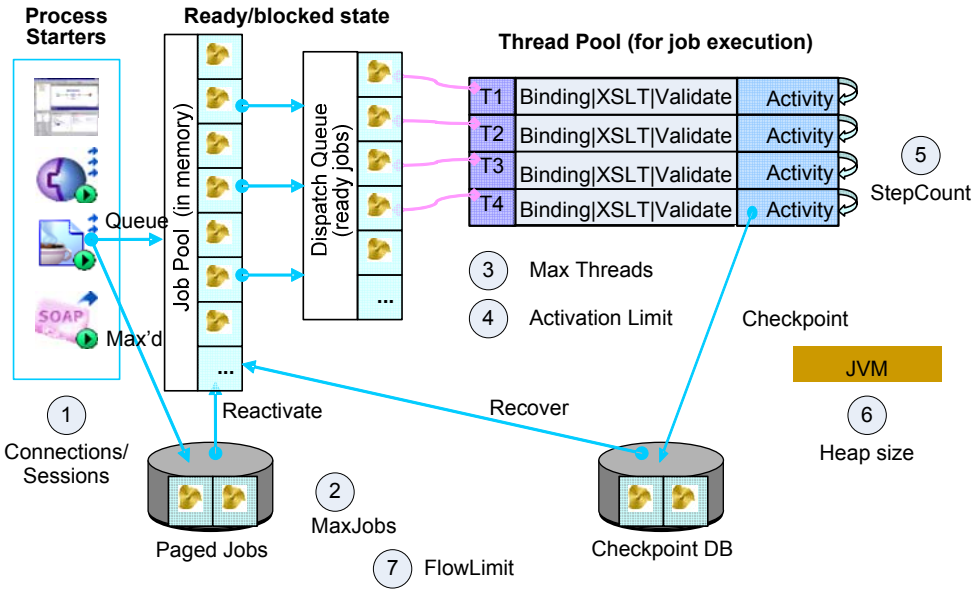

Figure 1. TIBCO BusinessWorks Message Flow Architecture

2.1.1 Engine Processing

Figure 1 illustrates the processing lifecycle of the process engine. Jobs are created by the process starters (1) as events arrive. The Job pool is an in-memory holding area for jobs. The job pool is limited by the amount of available memory on the machine and by the amount of memory allocated to the Java Virtual Machine (JVM) in which the process engine executes (6).

The job pool also can be limited by specifying a maximum number of jobs to hold in memory (2 and 7). Once the job pool reaches the specified maximum number of jobs, incoming jobs are paged out to disk or database as defined in the configuration panel.

The engine threads are the "CPUs" for job execution. A thread is either waiting for a job to be ready or is executing a job from the Dispatch Queue. Once a job is assigned to a thread, the number of activities or steps that will be executed uninterrupted is determined by the engine property `Engine.StepCount`. The default for `Engine.StepCount` is 20 steps. After that, the job goes to the back of the queue and yields the thread to another job in the front of the queue.

The exception to `StepCount` is when the job is blocked or in a transaction. If in a transaction, the thread will not be released even if `StepCount` has been exceeded. However, if job is blocked, the thread will always be released. `Activation Limit` means the blocked job is not eligible for paging since it must complete before another job can be paged in. Think of it as once the job starts executing, it is "pinned" in memory until it completes.

You can configure the number of engine threads used for executing processes in the deployment configuration of the Process Engine; the number of threads effectively limits the number of process instances that can execute concurrently. Set the thread number to a value that is appropriate for your operating system and your physical machine configuration. The thread count is set in the TIBCO Administrator via the Edit Service Instance dialog box, under the Server Settings tab.

Jobs in the Dispatch Queue are assigned to the next available thread for execution. The jobs are processed in a round-robin fashion. The job at the top of the queue has the highest priority. When a job in the Job Pool is ready to execute, it is added to the Dispatch Queue. When a job is blocked from execution — for example, when the job is waiting for a resource/event — the job is removed from the Dispatch Queue. A job goes to the end of the queue after it exceeds the specified `StepCount` (5). At that point, the thread is yielded to another job. A job is taken off the Dispatch Queue if it is blocked on a resource/event. In this case, the job may be paged out, depending on whether `MaxJobs` and/or Activation Limit is set.

A job may loose control of a thread if:

- The job ends normally
- An exception is raised and uncaught
- The job encounters a “blocking activity” that the TIBCO BusinessWorks Engine is aware of.

A blocking activity is one that may block a thread from doing useful work while waiting for a result. For example, all request-reply activities are blocking activities. It is important to note that blocking activity that occurs as a result of I/O calls in user-supplied Java code will keep the worker thread tied up while the I/O is waiting for the reply. A blocked job is taken off the Dispatch Queue until it is unblocked.

2.1.2 Determining the Available Memory

An important part of engine tuning is determining how many jobs of all types can fit into the available memory (size of JVM heap) at a given time. This is best done by running several sample workloads at a steady state and observing memory consumption behavior. From this, you can derive a safe number of Jobs. You can use TIBCO Administrator to control the execution of TIBCO BusinessWorks Services. This is useful if your system has limited memory or resources, or if you want to restrict process instances to run sequentially.

2.1.3 Flow Control

Restricting the flow of incoming events is necessary to avoid memory overflow when processing lags behind. The `FlowLimit` property limits the number of jobs created. This parameter is specified in the Job Starters.

High throughput scenarios are more likely to benefit from `FlowLimit` tuning, with certain exceptions. `FlowLimit` should be used with protocols that store the messages on the server (for example, TIBCO Enterprise Message Service™, email) so that the server sending the message has a backlog of events until the process engine can handle them. Implementation of the `FlowLimit` feature greatly depends on the specific implementation and Quality of Service (QoS) supported by each. For example, TIBCO Enterprise Message Service and TIBCO Rendezvous Certified Messaging support once and only or at least once delivery semantics, whereas TIBCO Rendezvous reliable and HTTP doesn’t. However, these transports do have a certain level of reliability built in using the TIBCO Rendezvous deamon or queue implementation similar to Tomcat under HTTP event source. For other job starters you may have to use `MaxJobs` to control flow.

Setting `MaxJobs` to more than one and Activation Limit enabled will limit n-jobs to be processed as a batch at one time. It is useful for situations where resources downstream for concurrent processing are limited.

The following section describes special cases and best practices.

Rules of Thumb for Job Sequencing

- If your TIBCO BusinessWorks Engine can handle peak message rates you should consider using `MaxJobs`=0 and `FlowLimit` = 0. Choose `MaxJobs`=n if TIBCO BusinessWorks Engine cannot run out memory during peak rate. However, if memory is limited (which is true in most cases), use of `FlowLimit` should be first evaluated. Please review limitations described earlier. If `FlowLimit` is not applicable, use of `MaxJobs` should be evaluated.
- Use `FlowLimit` in cases when you are using a protocol that doesn't have high reliability and you can still tolerate message loss.
- Use `MaxJobs` in cases when message loss is not tolerated but requires a slightly bigger buffer.
- `MaxJobs` can improve throughput when there are many blocking activities (such as `Wait for`).
- Setting `MaxJobs` to more than one and Activation Limit enabled will limit n-jobs to be processed as a batch at a time. This is useful when downstream resources for concurrent processing are limited.
- Sometimes flow control is needed to maintain a FIFO processing order. Previous recommendation was to use `MaxJobs`=1 and Activation Limit set to True. Setting `MaxJobs`=1 for sequencing is not recommended anymore. Instead, use job sequencing methods. The new TIBCO BusinessWorks 5.2 allows concurrency and still maintains sequence when multiple independent threads exist.
- Calculating `MaxJobs`=n is more of an art than technical rule. TIBCO Software suggests running a test with a peak message rate, until TIBCO BusinessWorks runs out message. Plot CPU and memory usage behavior to come up with the most optimal `MaxJobs` rate. Please note that there is no uniform behavior when TIBCO BusinessWorks runs of out memory, as overall behavior depends on JVM implementation on different platforms.
- See the section "Calculating MaxJobs or FlowLimit" for more detail.

2.1.4 Paging Jobs

- An existing job can be paged to disk in any of the following cases:
  - The job is blocked
  - The `MaxJobs` limit is exceeded
  - Activation Limit is disabled, which gives priority to new jobs.
- A **new** job can be paged to disk in any of the following the following cases:
  - The `MaxJobs` limit is exceeded and Activation Limit is enabled, or
  - All paged in jobs are unblocked.
- Paged jobs are returned to the Dispatch Queue:
  - When the job is unblocked and the `MaxJobs` limit is not exceeded
  - Some other paged in job is blocked and can be paged out to make room.

2.1.5 Paging Waiting Jobs

Some processes need to wait for an event for an extended period of time. If flow control is enabled, the process may be placed into the paged process pool during the wait, freeing space for another process to enter the process pool.

If the task that waits for an event is a Signal In or Sleep task, you can use the process paging feature to write the process to disk. For example, when the event occurs, a Rendezvous message arrives for the Signal In task, and the process automatically re-enters the process pool and becomes a ready process.

2.1.6 Enabling Paging

The `Activation Limit` setting is used in the deployment configuration to control process paging. By enabling paging, you allow the number of active and waiting jobs to be greater than the `MaxJobs`/`FlowControl` setting.

3 Best Practices

This section describes some of the most important observations and guidelines developed in field trials. These guidelines are based on numerous performance improvement projects and details of the new features introduced with TIBCO BusinessWorks 5.2.

- TIBCO BusinessWorks Engine Tuning Guidelines
- JVM Tuning Guidelines
- Transport and Resource Guidelines
- Message, Payload and Schema Guidelines
- Process and Activity Design Guidelines

3.1 TIBCO BusinessWorks Engine Tuning Guidelines

The following section provides very high-level steps for TIBCO BusinessWorks performance tuning. The specific details are discussed in the subsequent sections.

Rules of Thumb for TIBCO BusinessWorks Engine Performance Tuning

Allocate significant time for selecting the JVM vendor, version, and necessary tuning parameters.

The number of threads that an engine will allocate is set in the TIBCO Administrator Edit Service Instance dialog box, under the Server Settings tab, General section.

- Since engines process on a separate Java thread, the number of worker threads controls how many jobs can run simultaneously.
- Measuring the available CPU and memory resources on your system under a typical processing load will help you determine if the default value of 8 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.
- There are two ways to take benefit of CPU utilization:
  - Optimize engine thread
  - Increase number of TIBCO BusinessWorks Engines on a single node or distribute on different nodes. Your performance suite must cover different scenarios so that you can come up with the optimal configuration.
- Typical numbers of worker threads range between 8 and 32. Specifying too low a value can cause lower engine throughput even though spare CPU resources exist. Specifying too high a value can cause CPU thrashing behavior.
- If the pace of incoming processes still exceeds the number of threads available to run them, consider using flow control.

3.1.1 Calculating MaxJobs or FlowLimit

Before you delve deep into the JVM and TIBCO BusinessWorks Engine memory management, it is important to keep the following issues in mind. First, note that `MaxJobs` and `FlowLimit` are two concepts most important to control flow in to TIBCO BusinessWorks. However, use JVM parameters to control TIBCO BusinessWorks Engine memory. Before you start to configure anything you must design a test suite that follows these guidelines.

- Figure out how many jobs of all types can fit into the available memory (size of JVM heap) at a given time.
- Run several sample workloads at steady state and observe memory consumption behavior to derive a safe number.

- If you have a deployment that dynamically loads many sub-processes (for example, 100+). The TIBCO BusinessWorks designer should try to avoid such a heavy design. In case the design cannot be partitioned, here are a few rules that you must keep in mind:
  - Make sure all the sub-processes are selected when you build the archive. Since they are dynamic, they will not automatically be pulled in by the calling process.
  - Note that only those processes that are listed in the Enterprise Archive of the deployed project will be loaded at engine startup time. A dynamic process not already loaded will be loaded when first called. Once loaded, the process remains loaded until the engine exits.

Specifying a value for `MaxJobs` causes the process engine to incur some overhead for managing the paging of process instances to and from disk. If you have sufficient system resources and do not expect incoming events to exceed the limits of your system, consider specifying `MaxJobs` as 0 and `FlowLimit` as 0. This allows the process engine to create an unbounded number of services and eliminates the overhead of paging. Calculating `MaxJobs` and `FlowLimit` for a process engine depends on the following factors:

- Amount of memory reserved for the engine JVM
- Maximum size of a process instance object
- Number of process instances per engine

A best practice is to design the process deployment in such a way that the target peak loads can be accommodated by the TIBCO BusinessWorks Engine(s) without hitting the –Xmx memory limit, and then set `MaxJobs` and/or `FlowLimit` to reflect the number of concurrent jobs that are expected under this peak load. While leaving `MaxJobs` and/or `FlowLimit` at 0 will suffice under these conditions and would be adequate for this normal operation, it will lead to severe throughput degradation under abnormal overload (error recovery) conditions. By setting the `MaxJobs` and/or `FlowLimit` to the value expected under normal peak load conditions, the throughput of the engine will be maintained at a fairly high level during even abnormal overload conditions.

3.1.2 Memory Management Issues

While designing maximum JVM heap size, it is important to keep in mind that JVM itself is not the only component that is bound by memory. For example, you could have a situation where TIBCO BusinessWorks could have TIBCO Rendezvous Certified Messaging job starters, and TIBCO BusinessWorks is running at full capacity as defined by `FlowLimit`. In this case, the TIBCO Rendezvous Certified Messaging sender will consistently send messages faster than the TIBCO BusinessWorks/TIBCO Rendezvous Certified Messaging job starter can process them. Although TIBCO Rendezvous Certified Messaging doesn't store messages in memory, there is meta-information for control purposes stored in memory. This information can eventually fill up available memory if TIBCO BusinessWorks is not properly optimized to handle new messages. This can obviously create an unstable situation.

There two important steps that should be followed to avoid this condition:

- Try to properly optimize the TIBCO BusinessWorks Engine so that it doesn't run into the "slow consumer" syndrome.
- Use TIBCO Hawk® to monitor memory usage and take proper corrective action.

This issue can obviously be true for any queuing system where the producer is consistently faster than the consumer.

3.1.3 Tuning Engine

Although flow control parameters such as `FlowLimit` and `MaxJobs` are the most important, the TIBCO BusinessWorks Engine should also be tuned for the following important parameters.

3.1.3.1 Thread Count

The number of threads that an engine will allocate is assigned with the engine property. Tuning thread count to the optimum value is an art. Normally start with a default value of 8 threads and double it up until max CPU is reached.

3.1.3.2 Step Count

This TIBCO BusinessWorks property controls the max number of execution steps (unless inside a transaction) for a job before an engine thread switch occurs. The default value is 20, thereby preventing frequent thread switches (which can slow down the engine) in processes with a large number of steps.

`Parameter: Engine.StepCount`

3.1.3.3 MemorySavingMode

This parameter significantly improves memory footprint for processes that manipulate a large amount of data for a small part of their lifetime. This TIBCO BusinessWorks property allows the engine to release references to unused process data so that it can be garbage collected by the JVM, thus improving the performance of processes with large amounts of data. This feature is very useful in the scenario where a large amount of memory is occupied by a variable defined in a process; for example, a variable that reads in a very big XML file. In this scenario, it is best to release memory right after process instance completes.

`Parameter: EnableMemorySavingMode=True`

3.2 JVM Tuning Guidelines

Each TIBCO BusinessWorks engine runs as a multi-threaded Java server application. Processes and other objects used internally by TIBCO BusinessWorks are Java objects that consume memory while the engine is running. Java provides some useful parameters for tuning memory usage. TIBCO recommends that TIBCO BusinessWorks customers consider various factors when selecting a JVM. Besides JVM version and vendor, the most relevant tuning parameters are:

- JVM heap size
- Server VM vs. Client VM based setting
- Garbage collection settings

Rules of Thumb for JVM Selection

- Hotspot Server JVM will give better performance with TIBCO BusinessWorks. Sun also claims better performance for Hotspot Server JVM.
- Consider using the multi-threaded garbage collector, which is available in Jdk.4.x. Experimenting with processor sets with multiple engines is a good idea.
- More Information regarding these settings is at http://java.sun.com/docs/hotspot/gc1.4.2/index.html

This section provides basic guidelines for tuning Java memory usage to optimize engine performance. Platform-specific considerations may also apply. For details, please consult the documentation for your hardware or operating system.

3.2.1 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 needed for a TIBCO BusinessWorks Engine, you should determine the largest heap size that can reside in physical memory. For best engine performance, paging to disk should be avoided.

Rules of Thumb for Heap Size

- Recommended heap size for a small workload is 256MB, for medium 512MB, and for large workload 1GB or more. This recommendation should change if available memory is significantly higher.
- The Maximum heap size is set with the Service Instance Configuration panel in the TIBCO Administrator.

TIBCO Runtime Agent™ allows you to modify the Java heap size, JVM selection, and JVM settings that the TIBCO BusinessWorks Engine uses when it is started. This memory is committed for engine activities and job processing.

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

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

Setting these values to the same number fixes the memory size to that number. For details on these parameters, go to http://java.sun.com/docs/hotspot/ism.html.

The total amount of JVM memory needed to operate a TIBCO BusinessWorks Engine should be the memory for each process plus the maximum number of processes that can be in the process pool. If flow control is enabled, the process pool can contain up to the `MaxJobs` value.

The instructions for setting JVM Heap parameters are found in TIBCO Administrator™ documentation.

3.2.2 Setting JVM and Processor Affinity

If you have multiple processor machines, TIBCO Software recommends that you assign processor affinity with different instances of TIBCO BusinessWorks Engines:

- On Windows, use the `Task Manager` facility to assign affinity.
- On a Solaris platform, create processor set using `psrset/psradm/pbind`.
- On Linux, use the `taskset` command to set processor affinity.

3.2.3 JVM Garbage Collection

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 that the client should try to experiment with these options, but that results are not very deterministic. It can be argued that a multi-process system (4 to 12 CPUs) using `ParallelGC` can produce better results. 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.

Finally, the application should not make direct explicit garbage collection. If the application does make a call to explicit garbage collection and if you would like to disable it, use `verbose:gc -XX:+DisableExplicitGC`.

3.3 TIBCO BusinessWorks Transport and Resource Guidelines

Performance should be one of the criteria for using appropriate transport. Besides performance, the user needs to be concerned about interoperability, reliability, and security requirements. Users have choices of using XML over HTTP, XML over JMS, SOAP over HTTP, SOAP over JMS, TIBCO Rendezvous messages 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 should use them only as general observations for initial understanding. Since each customer has a different environment and different requirements, these results cannot be directly applied to their work. Instead, it is strongly recommended that you create a realistic scenario in the lab and derive appropriate results from that.

TIBCO Software's lab results indicate that, overall, HTTP is a better performing transport when compared to JMS:

- Figure 2 compares different QoS available in JMS vs. HTTP
- Figure 3 compares SOAP over JMS vs. SOAP over HTTP

It is important to note that both of these tests have been carried out under different test environments, and that these results can be very different based on individual use cases. The SOAP protocol adds significant overhead whether you use it on HTTP or JMS, since there is an overhead cost of creating and parsing the SOAP envelope.

In general, performance throughput decreases when payload is increased from 1KB to 5KB. Overall lab results for the particular test case also showed that SOAP over HTTP has a better throughput compared to SOAP over JMS.

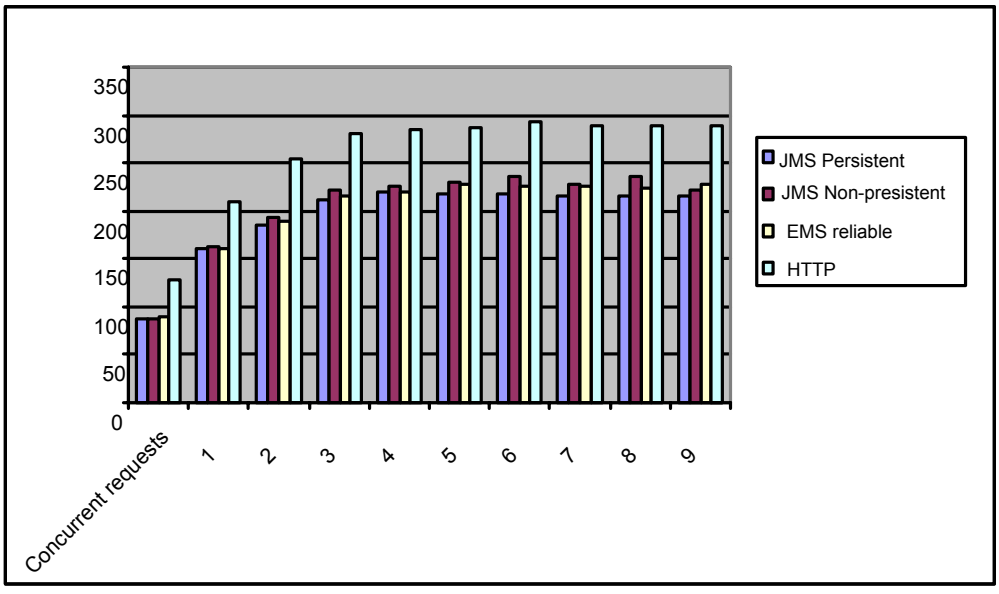

Figure 2. Comparing Throughput for SOAP With Different Transports

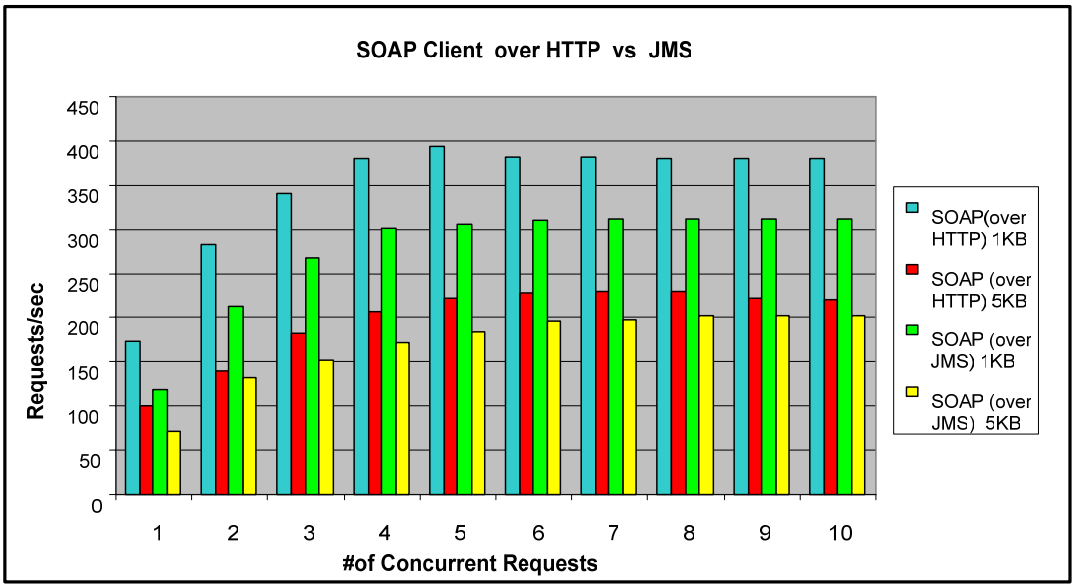

Figure 3. Comparing Throughput Between SOAP over JMS vs. HTTP

However, these results do not set the best choices for transport, and you should test transport under your major use case. You also need to consider QoS and scalability requirements.

The following observations can be made based on various comparisons of HTTP clients, HTTP servers, SOAP clients, and SOAP servers:

- As the payload increased from 1KB to 5KB, the throughput dropped.
- There was no significant increase in the latency when the payload increased from 1KB to 5KB.
- As the number of HTTP/SOAP client requests increased (concurrent requests), the throughput increased.
- As the number of HTTP/SOAP client requests increased, the CPU utilization increased.

Note

Results in Figure 2 and Figure 3 are taken under different environments and are not normalized.

3.3.1 HTTP/S Client

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

- HTTP/S Client Thread Pool
- Persistent Connection Manager (PCM)

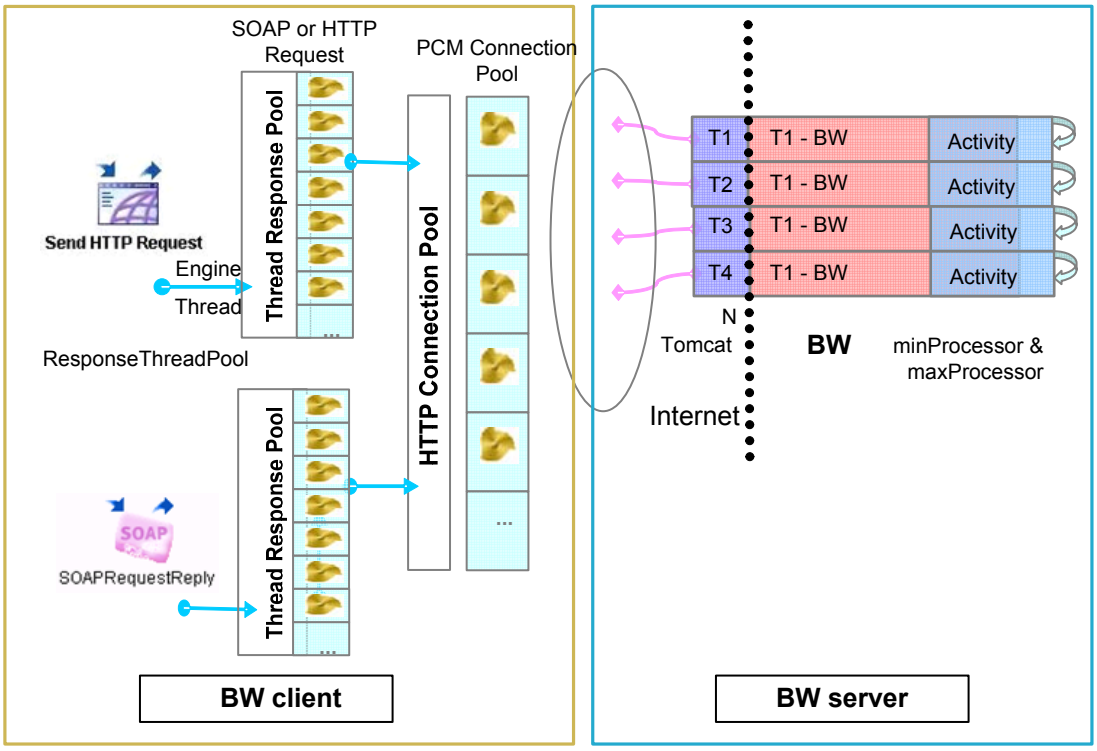

Figure 4. HTTP Performance Architecture

3.3.1.1 HTTP/S Client Thread Pool

If you are using HTTP/SOAP with HTTP/S Requestor (Request/Response) activity, it is very important to verify that the rate of request received on the HTTP server keeps up with the rate at which the client sends messages. This situation is likely to arise when there are very high numbers of concurrent requests being made.

Each Request/Response activity that uses the HTTP protocol (for example, Send HTTP Request or SOAP Request Reply) is associated with a unique thread pool. Each request is executed in a separate thread, belonging to the thread pool associated with the activity. The value of this property controls the size of the thread pool. The number of threads in the pool determines the maximum number of concurrent requests a request/response activity can execute. The default value of this property is 10.

The thread pool is created when the engine starts: therefore, be careful to 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 all the way to 50. However, this value should be increased only if the client side has a lot more backlogs compared to the SOAP receive side. It is recommended that you design a test framework that helps you monitor such behavior and determine optimal thread pool count.

`bw.plugin.http.client.ResponseThreadPool`

3.3.1.2 Persistent Connection Manager

Persistent Connections Pool (PCP) is a new important feature when the client makes a call to the same server very frequently. The cost of the HTTP connection is deemed important, and state management of the connection is very well understood. In TIBCO BusinessWorks 5.2, the Send HTTP Request-Reply and SOAP/HTTP Request-Reply activities can now share connections. See *TIBCO BusinessWorks Palette Reference*, Chapter “HTTP Palette”, section ”Persistent connections” for more information.

HTTP (SOAP/HTTP) request-reply activity configuration is enhanced to allow users to setup a pool of persistent HTTP connections with the server. However, this feature should be used only after fully understanding implications of using this feature. Hence, by default HTTP provides non-persistent connections. If persistent connections are chosen, it can improve performance because instead of creating and destroying a connection for each job, a pool of reusable shared persistent connections will be maintained.

Decision to use persistent vs. non-persistent connections

TIBCO Software’s internal performance tests indicate that persistent HTTP connections provide improved results for SOAP over HTTP and pure HTTP clients. These results were collected under a strict, confined environment with no other traffic on the network and with only a single hop. Hence, it is possible that these results can be even better where HTTP connection creation cost is higher, such as when the HTTP client and server are located on different network segments with multiple hops. These results vary further by other parameters, such as the number of clients or number of destinations. However, these results should not be taken as a reason to use persistent connection: there are many conditions where using persistent connection may create inconsistent behavior.

Persistent HTTP connections have a number of advantages:

- By opening and closing fewer TCP connections, CPU time is saved in routers and hosts (clients, servers, proxies, gateways, tunnels, or caches), and memory used for TCP protocol control blocks can be saved in hosts.
- Network congestion is reduced by reducing the number of packets caused by TCP opens, and by allowing TCP sufficient time to determine the congestion state of the network.
- Latency on subsequent requests is reduced since there is no time spent in TCP's connection opening handshake.
- HTTP can evolve more gracefully, since errors can be reported without the penalty of closing the TCP connection.

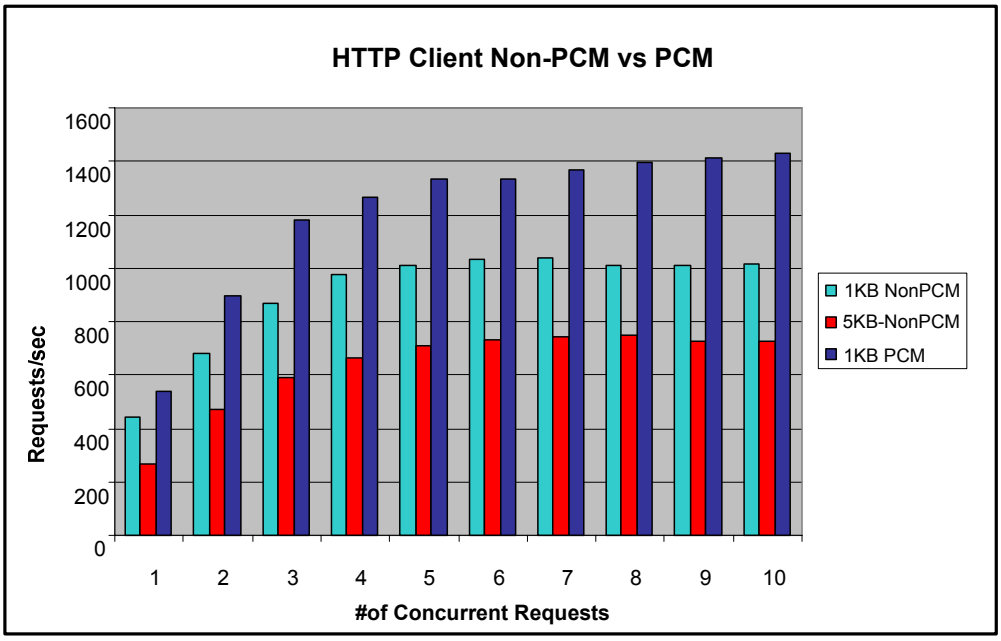

Figure 5. Comparing Throughput Between HTTP Non-PCM Client and HTTP PCM Client

This figure shows throughput of two clients: the HTTP client that is not Persistent Connection Manager-based and the HTTP Persistent Connection Manager client. The most important disadvantages of using persistent connections are:

- The HTTP (SOAP/HTTP) client may report exceptions that are not usually reported when using non-persistent connections.
- The HTTP server may end up receiving duplicate requests from the same client. Usually, these issues occur when the client is not configured to check for stale connections. For different reasons, the connections that are persisted may become stale. When attempting to use a stale connection, the underlying HTTP client application may fail, throwing an exception that may be propagated to the business process layer.

Best Practices for using persistent connections are as follows:

- The client should check the connections before using them. In TIBCO BusinessWorks 5.2, you can select an option that checks for stale connections (see property `bw.plugin.http.client.checkForStaleConnections`). When the underlying HTTP application detects a stale connection, it closes it and gets another connection from the pool. After three unsuccessful attempts to get a non-stale connection, the client fails, throwing an exception. However, this option should be used with caution, since it causes the overall performance to drop significantly, making the use of a non-persistent connection a better choice.
- Open and close fewer TCP connections to save CPU time.
- Use persistent connections when making repeated connection to the same endpoint, when a stale connection is not an issue, and when the server handles or supports duplicate messages.
- When using HTTP over SSL, note that this adds overhead and that overall request/response throughput may drop.

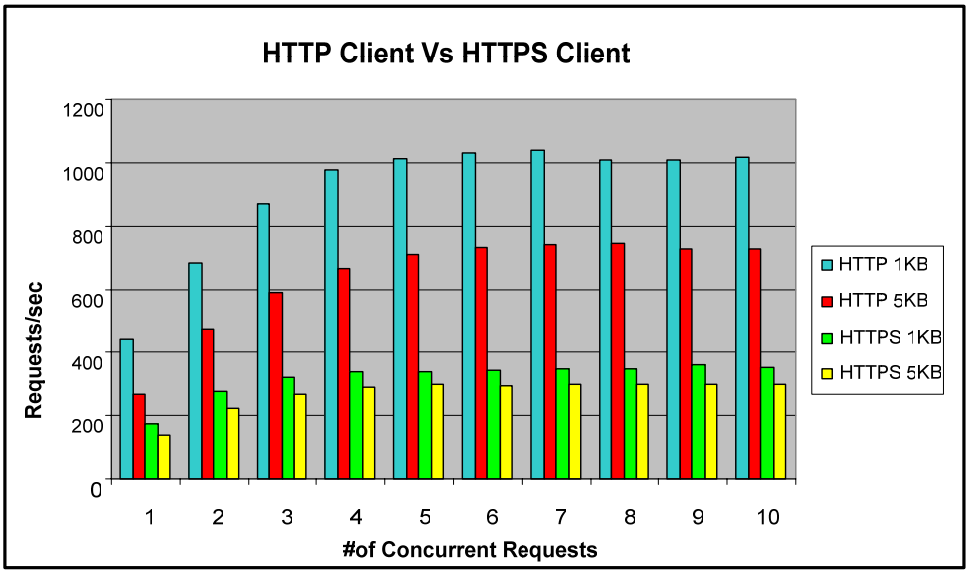

Figure 6. Comparison of Throughput Between HTTP Client and HTTPS Client

When the client cannot check for stale connections, it is very likely that the server would receive duplicate messages. If you cannot rely on the client to detect the stale connection, the next option is to make sure the server provides support for duplicate message detection.

3.3.1.3 Flow Control in HTTP

If you use the `FlowLimit` parameter for the HTTP Client, you should be aware of certain design limitations related to implementation of such a protocol caused by limitatonsl of transport level QoS. The HTTP event source acts similar to Tomcat implementation of `maxProcessor` and `minProcessor`, with one important difference:

- If `FlowLimit` parameter is set (i.e. >0), a general rule of thumb is that `bw.plugin.http.server.maxProcessors = FlowLimit - 1`, and `bw.plugin.http.server.minProcessors = maxProcessors / 2`.
- If `FlowLimit = 0` or Not Set Default values, set `bw.plugin.http.server.maxProcessors = 75` and `bw.plugin.http.server.minProcessors = 5`.

TIBCO BusinessWorks users are encouraged to tune this property carefully to achieve best results.

Details about configuring Persistent Connection Manager are described in the section "Configuring Persistent Connections."

3.3.2 HTTP/SOAP Server

Following important considerations apply to HTTP and SOAP over HTTP server:

- Configure `minProcessor` & `maxProcessor` as described in the configuration section.
- HTTP servers can be deployed in various load-balanced situations. However, deploying multiple HTTP servers that listen on the same machine is not possible, since the HTTP port is bound to a specific machine. Alternatives are available, such as creating reverse proxy solution, but they are beyond the scope of this document.
- HTTP traffic that arrives from external sources should be evaluated carefully. If you can document externally arriving content and patterns of traffic distribution, use content aware XML accelerators or routers such as Cisco® Application-Oriented Networking (AON), Sarvega (now in INTEL), or Reactivity.

Details about configuring an HTTP Server are found in the section “Configuring HTTP Server.”

3.3.3 SOAP

Simple Object Access Protocol (SOAP) is a lightweight protocol for the exchange of information between web services. Currently, the supported transports for SOAP are HTTP and JMS. The performance characteristics of SOAP are closely tied to the performance of the HTTP and JMS implementation in TIBCO BusinessWorks, and the only way to load balance SOAP over HTTP is to include a Local Director for directing requests to a set of configured engines.

Major influences of performance for SOAP are similar to the generic influences:

- **Message Complexity**. The more complex the process the greater impact on the parsing (packing and unpacking) of the message as it travels through various stages of the SOAP protocol.
- **Secured**. The decision to use basic authentication will affect performance. This is straightforward and the impact is easily measured.
- **Load Balanced**. The use of a local director to distribute incoming requests.

3.3.4 JMS

Recommendations for using Java Message Service (JMS) are as follows:

Tips: Transport Usage Recommendations

- Within TIBCO BusinessWorks 5.1.3 and prior versions, transport such as JMS can be throttled by limiting the number of sessions. JMS doesn't deliver more messages until some of the sessions have been acknowledged.
- Within TIBCO BusinessWorks 5.2, significant improvements have been made to this mechanism. New for this release is the combination of TIBCO Enterprise Message Service features Explicit Acknowledge and FlowLimit. 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 one is acknowledged.
- TIBCO BusinessWorks allows you to configure multiple sessions to receive messages faster; and to set number of sessions higher than the number of engine threads.
- Acknowledge (confirm) messages as soon as possible to improve throughput.
- By holding `Client ack` to the end of the process, you will block that session. This means you will slow down the rate at which TIBCO BusinessWorks pulls messages from the JMS server, which will have to hold 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 more 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 or TIBCO Rendezvous CMQ transport to distribute the work. External mechanisms exist to allow HTTP to be used for this purpose also.
- You must use a hardware/software load balancer to improve HTTP performance and load distribution.
- If possible, choose `NON_PERSISTENT` as the delivery mode in replying to a JMS message.
- Using `JMS Queue Sender` and the `Wait For JMS Queue` message instead of the combined `JMS Queue Requestor` activity may improve throughput.
- `Simple` and `Text` JMS message types have the lowest processing overhead.
- To design a long running process to fetch a message and process it, use `Get JMS Queue` message activity in a loop instead of `Wait For JMS Queue` message. In most cases, a JMS starter will be sufficient in this scenario.

3.3.5 FTP

FTP activities that share the same host, port, username, and password within the same process definition can now share the same session. The Quit (post-command) field specifies that the command should close the session. Keeping the session open for all activities can improve performance because there is significant overhead creating a session for each activity. See *TIBCO BusinessWorks Palette Reference*, Chapter "FTP Palette", description of Quit (post-command field) for more information.

3.3.6 JDBC

JDBC activity is the single most frequently used data access activity that has a significant performance implication. It is recommended to use certified, high performance type 2 JDBC drivers.

Internal benchmarking tests show that Oracle thin driver performed better than DataDirect driver for batching (insert operation). However, individual use cases should be evaluated within customer's lab for the final determination.

The JDBC Query activity can now fetch batches of records at a time instead of retrieving the entire result set. JDBC Update activity has been enhanced to allow multiple statements to be executed. See *TIBCO BusinessWorks Palette Reference,* sections "JDBC Query" and "JDBC Update" for more information.

Use Batching instead of Statement when possible (Better Latency)

A general rule of thumb is to initially set the maximum number of database connections to slightly less than *engine thread count.* If there are not enough database connections allocated, some jobs may be blocked waiting to free a connection. You should continue to monitor the number of database connections using database tools. There are a few tools that you should be familiar with: TIBCO trace tool, described in this paper, and specific DB tools to monitor the overall number of database sessions.

3.3.6.1 Avoid Development Misuses

- Inefficient queries – sending SQL data that asks the database to do more work than necessary; for example, tables without indexes.
- Excessive querying – efficient queries called too frequently.
- Large data sets – processing large sets of data in `ResultsSets`.

3.4 Process and Activity Design Guidelines

3.4.1 Data and Caching

Data types within TIBCO BusinessWorks fall into several categories and have different performance characteristics.

3.4.1.1 Static data

These data are typically kept as a local copy; TIBCO BusinessWorks hangs on to it in memory and user doesn't need to worry about it being stale. One example is global variables implemented internally as the XML structure known as `$_globalVariables`. TIBCO Software recommends removing unused global variables from the project. Unused variables can increase the latencies, especially for repeat-until-true where iterator conditions XPATH formula contains Global Variables instead of constants.

The global variables used for input-binding come from `$_globalVariables`, which is just another XML structure. Internally, these structures are trees, but are not accessed directly. The tree must be walked to find the correct element. It does take longer to access the last element than the first element.

Having unused elements in the middle only makes it worse. Since these are read-only structures, synchronized access is not required. Once unused global variables are removed, arrange them from MFU (Most Frequently Used) to LFU (Least Frequently Used).

3.4.1.2 Near static data

- Keep a local copy, hang on to it in memory, and lazily check for updates.

Example: XML Documents (Name Value Pair)

3.4.1.3 Dynamic data

- Work on local data, cache carefully, and use optimistic locking.
- TimesTen is an in-memory database that can improve the performance of checkpoints and wait/notify activities. Install and configure the TimesTen database, which is now available at http://www.oracle.com/timesten/index.html and then configure the JDBC Connection used for process engine storage to use the Times10 database driver.
- TIBCO® BusinessWorks Smart Mapper is another example which has been used extensively in cross-referencing (1-1 or 1-n) data. It provides a data caching option which simply removes any performance overhead of looking up data from the database.

3.4.1.4 Shared data

- The Java Global Instance shared configuration resource allows you to specify a Java object that can be shared across all process instances in JVM.
- Shared Java object should be instantiated at engine startup time and shared by all activities that access it.
- If an object can be shared across multiple process instances, instantiate shared data only once to improve performance and reduce overhead.

3.4.2 Checkpoints

The high volume applications which make a significant use of checkpoint have to be continuously aware of the performance cost of writing data to the checkpoints. These applications frequently need faster access to checkpoint data, replication of checkpoint data between different systems for failover reason, and database server-failure protection.

TIBCO BusinessWorks uses JDBC to access checkpoint information.

3.4.3 Signal-In and Group

- Do NOT use Signal-in within a loop group to receive/process one message at a time.
- It is ineffective because the signal-in's internal queue is unbounded for out-of-band events waiting to be matched up with a job key.

3.4.4 Local Only Field

The Notify Configuration has been enhanced with a Local Only field to allow an in-memory notification when the Wait and Notify activities are performed on the same machine. See *TIBCO BusinessWorks Palette Reference*, Chapter "General Activities Palette, the field Local Only in Notify Configuration for more information.

3.4.4.1 Call Process

Every time CP is called, input data including process instance data are bounded to the sub-process input data structure. In most cases reference is made to original entity; however, whenever data is mapped a copy of data is created. Keeping this obvious cost in mind, the designer should always be mindful of use of CP and continuously evaluate cost vs. CP modularization.

Rules of Thumb for Process and Activity Design

- Call Process (CP)
  - Consider using “Spawn” CP for heavy asynchronous process (such as logging). This will not decrease response time (not CPU usage)
  - Be aware of tradeoff between data binding cost vs. modularization
  - CP should be frequently reusable sub-processes with reasonable complexity
  - Use shared process variables to avoid data copying cost in CP
- Review use of large strings that requires manipulation
  - Evaluate need for passing large strings especially between CPs
  - Any changes within CPs will create a new copy
  - Parse structure upfront
  - Keep string reference out
- FTP
  - Use session caching
- Wait for Notify
  - Do NOT use Wait for Notify within a loop group to receive/process one message at a time.
  - It is ineffective because the Wait for Notify’s internal queue is unbounded for out-of-band events waiting to be matched up with a job key.
- Notify Configuration (Local Only)
  - Use Local Only field in Notify Configuration. Allow an in-memory notification when the Wait and Notify activities are performed on the same machine.

3.5 Message, Payload and Schema Guidelines

Designing message and payload size should always be very carefully thought out. In addition, TIBCO Software recommends that you design your performance suite with different message size in mind, such as small, medium, large and extra large. TIBCO Software doesn't recommend any specific guidelines for the maximum allowed size to categorize each; however, it does recommend that the customers develop their guidelines within their performance test harness.

The general guidelines are as follows:

- Be aware of performance cost of message; keep message size as small as possible
- If you are using binary or large document, consider using file reference
- If your scenario includes a large memory space occupied by a variable defined in the process, such as `activity#1` (a variable that reads in a very big XML file), plan to release memory right after the process instance completes:

  `Enable MemorySavingMode=True`

This parameter significantly improves memory footprint for processes that manipulate a large amount of data for a small part of their lifetime. This TIBCO BusinessWorks property allows the engine to release references to unused process data so that it can be garbage collected by the JVM, thus improving the performance of processes with large amounts of data.

- Keep XML Structure and schema complexity as simple as possible
- Review use of large string that requires manipulation
    - Any changes within CPs will create a new copy
    - Parse structure upfront
    - Keep string references out
- Avoid or re-architect XML schema with XML elements that contain nested elements of nested elements. Many Open Applications Group Business Object Documents (OAG BODs) have these issues.

3.5.1 Data Representation Guidelines

While using JMS as a transport for TIBCO BusinessWorks, the following matrix should be considered as very high-level performance guidelines. Use these as a general rules, but you should certainly experiment with different options especially in the high throughput environment:

Table 1. Comparing Performance of Different Data Types

Message Type	Performance Rating	Comments
Simple	Very High	Without a body portion there is no extra parsing required. This is by far the best performing message type.
Text	High	Treated as single text field. No parsing needed.
Bytes	High	Bytes require parsing downstream in the process. This influences the rate the process starter is able to receive the JMS Message.
Map	Medium	A set of name-value pairs that can be accessed sequentially or by name. It requires a level of parsing in the incoming activity, which has an adverse affect on the incoming rate.
Stream	Medium	A set of ordered values. No parsing needed.
XML Text*	Medium	A single text field containing XML, which will be parsed by the process starter. This is a major change in TIBCO Active Enterprise™ 5 wire format. When XML Text format has been chosen, TIBCO Rendezvous will use the internal `TIBRVMSG_XML` type and compress the message on the wire.
Object	Low	A serialized Java object. The object is de-serialized in the process starter. De-serializing requires significant CPU and memory resources.

3.5.2 TIBCO BusinessWorks Mapper Performance

TIBCO BusinessWorks 5.x provides a full-featured XSLT editor. Although TIBCO BusinessWorks mapper provides greater flexibility, granularity and support for XSLT 2.0 features (such as for-each-group), it is important to keep in mind that improper use of drag-and-drop may end up creating inefficient XSLT.

Many TIBCO BusinessWorks mapper performance issues arise from this inefficient XSLT. Proper understanding of mapper and code optimization may be necessary for the best performance.

In general, users should pay attention to following guidelines:

- Use `Mapper` (startup time parse) instead of `Transform XML` activity wherever possible for better performance. Be aware that the `Transform XML` activity invokes XSL parser at the runtime and hence it may add performance overhead every time a process runs. `Mapper` on other hand dynamically loads definitions and evaluates at input document and mapping rules at the runtime. However, there are some cases appropriate for using the `Transform XML` activity, such as input and style sheets being different every time `Transform XML` activity is invoked.
- Use `tib:render-xml` function instead of `Render XML` if you are not modifying XML structure.
- Think of the process or source data as an XML tree.
- Note that large data sets (1000s of records) take time to traverse.
- Note that XPath expressions are evaluated to take you to a specific element (node) or group of nodes.
- Don't let the same expression get evaluated many times (especially if the data is large).
- Use variables and evaluate them only once.
- Because of XML tree traversing, always use the "iteration element" in a loop if there are more than a small number of records.

This section will focus on providing a better design practices for an improved performance using TIBCO BusinessWorks.

3.6 Processing Large Sets of Data

Common uses of a TIBCO BusinessWorks process are to retrieve a large set of data (for example, a JDBC result set) and then process it through a series of activities. Care should be exercised to limit this data to a manageable "chunk" before processing.

It is important to note that the default behavior of the JDBC query activity, and other activities such as parse data or parse XML, is to parse/retrieve the entire result into memory for downstream processing. This has two effects on performance:

- **Memory** Large parsed objects can consume considerable memory very quickly, therefore allowing an unspecified number of records to be retrieved at once can cause memory usage problems.
- **Indexing of records** If your data has a large number of repeated records/elements and you plan to iterate through them, performance will degrade in a non-linear (quadratic) fashion as the number of records grows. This is due to the increased lookup time as the data "tree" is traversed from the start record/element to the current node being processed. This effect can be amplified by referencing that node many places in mapping/binding activities. The use of variables can minimize this impact.

The solution to both of these problems is to retrieve the data in smaller sets/chunks and use a group to iterate through the entire set. The way to do this varies based on the input data source, but a couple of common inputs are results of Parse Data and JDBC Query activities.

For the parse data activity, you can easily configure it to retrieve a set number of records and process them before retrieving more. The configuration is documented in the *TIBCO BusinessWorks Palette Reference* in the sections “Parse Palette”, “Parse Data”, and “Parsing a Large Number of Records.” One feature of the parse data activity is that it does not buffer the whole file from the disk. Therefore, you can use it to process extremely large (1G +) files with a controlled amount of memory usage.

For the JDBC query activity, it is possible to retrieve the results in chunks by using more selective SQL and surrounding the JDBC query activity with an iterate group. For example, if the table holds 100K rows and you retrieve 1K at a time and iterate 100 times, the total processing time will be significantly less than retrieving all 100K rows at once. Some experimentation can be used to optimize the size of the chunk depending on the type of data and processing required.

3.6.1 Iterating Through Large Sets of Data Using a Mapper Activity

Many users are not aware that the mapping technology in TIBCO BusinessWorks (XSLT) has mechanisms for iteration and looping, which makes it possible to iterate through data within the mapper using the statement `xsl:for-each` or `xsl:for-each-group` instead of using a group around the mapper. This obviously does not apply if multiple activities must execute within each iteration.

If the mapper is the sole activity, it is much more efficient to use the mapper and utilize `xsl:for-each`, instead of placing the mapper in a group and using the “accumulate output” feature of the group.

3.6.1.1 Use of Variables

One of the single most important optimizations in the mapping activity and any activity that transforms data is the use of variables. The mapper activity retrieves data at runtime by evaluating XPath expressions and traversing the input tree to access the matching data. If the tree is large and the mappings define many lookups of related or identical data, using variables to store that data can improve performance significantly.

An example is a large number of records in an input tree that are processed one at a time.

The XPath for one output value in the mapper (contained in an iteration group) might look like the following:

```
$JDBC-Query/resultSet/Record[$index]/field1
```

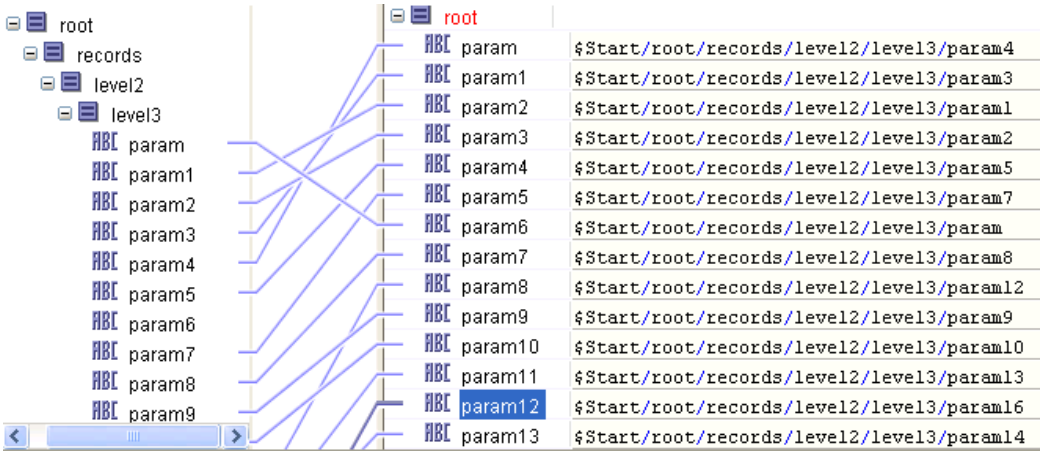

Figure 7. Inefficient If Data Is Large

If this expression is repeated many times for multiple fields, it will be more efficient to create a variable at the top of the mapper input pane (and generated XSLT) that evaluates the repeated portion of the XPath (`$JDBC-Query/resultSet/Record[$index]`) and then use the variable as part of all subsequent XPath expressions, such as:

```
$var1/field1

$var1/field2

$var1/field3
```

The benefits of using variables within the mapper will vary depending on how large the data is, how complex the expression is, and how many times it is repeated. The mapper GUI makes this easy to use. Using the mapper GUI, once you create the variable in the input pane, you will see it on the left side with the appropriate schema representation to allow drag-and-drop mapping.

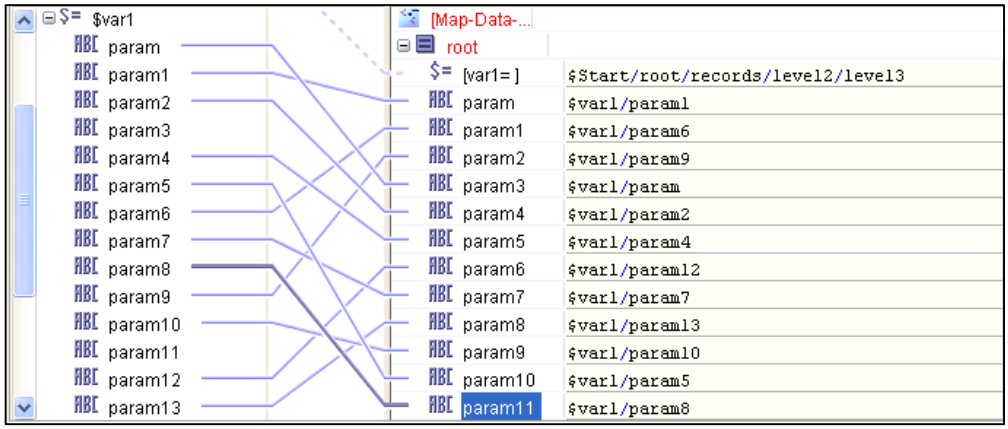

Figure 8. Efficient Design Using Local Variable

In TIBCO BusinessWorks 5.2, you can store the current iteration element in a process variable for faster access during iterations. See *TIBCO BusinessWorks Process Design Guide*, Chapter 6, section “Iterate Group” for more information.

3.6.2 Sorting and Grouping Data

A common mapping requirement that can result in very inefficient XSLT is to sort or group a list-style output by a value of some data in the input data. For example, you may need to create a list from an input list where every item that has a unique value gets created on the output, but duplicates are dropped. Or you may create a list in which every item that has a matching `itemType` gets grouped on the output.

One way to do this is use the preceding-sibling axis to check all of the previously processed items for a specific value before processing the current item. This obviously gets progressively slower as the number of items grows.

The solution is to use `for-each-group`, an XSLT 2.0 statement that is supported in TIBCO BusinessWorks 5.1.2 and later. There is an example documented in the *TIBCO BusinessWorks Process Design Guide* in the section “Converting a List into a Grouped List.” Using this feature can achieve dramatic performance improvements for these use cases.

3.6.3 Large Document/Record Use Cases

You may receive large documents or messages with the SOAP, HTTP, or Mail process starters. Large documents or messages can consume considerable memory and may degrade the performance of your system. You can specify a threshold size for incoming large documents and messages so that items that exceed the threshold are written to disk instead of stored in memory.

Once the large document or message is written to disk, you can use the Read File activity to obtain the contents of the file, if necessary.

See the description of the Advanced tab in the SOAP, HTTP, or Mail process starters in the *TIBCO BusinessWorks Palette Reference* for more information.

3.6.3.1 Process Large Incoming documents

- SOAP, HTTP, Email: Use option process starter streaming option
- Later read it using File Reader → Data Parse

3.6.4 SOA Best Practices

These are the best practices for working with Service Oriented Architecture (SOA):

- Eliminate frequently repeated XML tags and elements
- Consider `async` invocation style
- Use synchronous or fine-grained invocation for a frequent back-and-forth communication between nodes
- Keep in mind that exception handling and document passing requirements will increase, such as Universal Application Network (UAN)
- Carefully consider pros and cons of
  - Compressing technique (binary compression such as Gzip)
  - Reducing overly frequent validation
- Investigate parsing technology
  - Pull parser (XPP3), hardware based solutions (Cisco AON, Sarvega, Reactivity)
- Reduce overly chatty protocol uses, such as WS-Discovery

4 Performance Improvement Customer Use Cases

In this section we will walk you through several use cases that may help you understand a typical process to improve performance.

4.1 Throughput Improvement and Latency Reduction for Heavy Processes

4.1.1 Problem

It was required to reduce overall latency and improve throughput for the following processes:

- **Asynchronous processes and sub-processes**. For example, UAN processes have many transformations and various nesting levels of sub-processes. Processes are being invoked asynchronously and make calls to external calls asynchronously. Since initially the JMS infrastructure used "client acknowledgement", overall latency of the process increased dramatically. Using `Explicit Acknowledgement` from TIBCO Enterprise Message Service reduced the latency problem. The next improvement was achieved by moving from `MaxJobs-` to `FlowLimit`-based flow control, which together with `Explicit Acknowledgement` significantly improved throughput.
- **Nested sub-processes** with fair amount of transformation and XML payload.
- **External calls that use TIBCO Enterprise Message Service Topic Publisher and Subscriber**. These calls are frequently asynchronous and require that the client acknowledges a session. Unfortunately, receiving process that uses `client ack` limits overall number of processes by the maximum number of JMS sessions. This creates an artificial flow control limited by the JMS session.

4.1.2 Measurement

Throughput for this type of process is in the range of 10 to 20 per second. It is measured from when a request is made by the web client until a response is received. A typical request could go through the entire level described earlier and finally invoke an asynchronous call to the source systems (for example, database or CRM) and receive replies in asynchronous fashion.

4.1.3 Solution

TIBCO BusinessWorks 5.2 infrastructure significantly improves internal xdata processing and parsing code. In addition, many of the activities provide internal streaming and data caching techniques so that Call process with output, XML Parse, XML Render, and iteration activity are all significantly faster than with the prior versions. New infrastructure has significantly improved performance for nested call process execution. All of these product improvements significantly help improve overall latency, without making a single code change.

The new flow control feature along with use of the `Explicit Ack` feature significantly helps improve throughput. The `FlowLimit` feature allows for more graceful handling of memory when a sudden increase in the rate of in-coming messages hits by automatically turning the event source off and on, thereby throttling the number of jobs started in the engine.

If your JMS Server and the TIBCO BusinessWorks Engine (using the JMS Client) are using different JMS provider, you cannot use TIBCO Enterprise Message Service `ExplicitAck`.

Instead, you must use `Client Ack`, in which case the maximum number of open connections should be set higher than the number of threads. This is done because it is good to keep the pipeline of pending events filled. Too high a number, however, will result in more memory consumed.

TIBCO Enterprise Message Service `ExplicitAck` piggybacks on a newer feature in TIBCO Enterprise Message Service that has the `Explicit Ack` mode. This mode allows TIBCO BusinessWorks using TIBCO Enterprise Message Service's `Topic` to scale better when receiving many messages. `QueueReciever` already has a way to tune the number of sessions, but `Topic` was limited to a single session. `Explicit Ack` uses TIBCO Enterprise Message Service's transport level protocol for acknowledging.

4.2 Web Service (HTTP) Throughput Improvement

4.2.1 Problem

A document handling system uses WS calls to retrieve, search, and store documents. The documents can be of different size. The input document can be structured (XML) or unstructured (images). High throughput for a process making WS calls is required (for example, 50 to 400 per second).

4.2.2 Measurement

This overall web services project is focused on several macro and micro performance objectives, with the following two main issues:

- Total end-to-end business process completed per second
- Throughput degradation between SOAP client and SOAP server invocation for many concurrent user requests. The benchmarking process focuses on maximizing CPU and memory utilization.

4.2.3 Solution

The case should be resolved in the following way:

- Start your initial process test on Windows OS and optimize the numbers first. Later, move to the OS of your choice for the OS-specific tuning.
- Create two TIBCO BusinessWorks processes making several SOAP/HTTP concurrent requests, and one process starting SOAP event source. Measure overall throughput per minute on both sides. Notice any degradation on either side. Tune the HTTP client thread pool (as described in the section "HTTP/S Client Thread Pool") to avoid any degradation between the SOAP client and the SOAP server.
- If you have SOAP or HTTP, using Persistent Connection Manager (PCM) improves performance. Actual performance numbers depend on various reasons described in the section "Persistent Connection Manager".
- A business case involving high throughput with lower processing will benefit from proper `FlowLimit`. You must experiment with best `FlowLimit` numbers based on your scenario. Note that `MaxJobs` tuning may not be useful in this case.
- Note that TIBCO Enterprise Message Service tuning doesn't help significantly because this process spends a lot of time within the SOAP Request activity.

- Reduce the `engineThreadCount` to 8 (default). Only after reaching sufficient CPU usage should you increase `engineThreadCount` (general rule of thumb is two times the previous value) and keep increasing CPU usage to a very high number. Find the optimal numbers of `engineThreadCount`.
- Add more TIBCO BusinessWorks Engines only after tuning previous parameters and if you find that engine CPU is being utilized properly.
- Note that JVM and OS tuning plays the most important role in improving performance.
- Continuously monitor CPU activity both on the client side and on the server side. The goal is to scale processes using different techniques until CPU usage reaches 100%.

4.3 JMS Scenario

Frequently, customer deployment starts with limited scalability to achieve an initial target. However, businesses must build architecture that can scale horizontally to address significantly higher throughput requirements in the future.

The JMS server distributes messages to sessions in a round-robin fashion. Under Client Acknowledgement option, each TIBCO BusinessWorks process creates a number of sessions configured in the JMS event source activity. The response time is affected by the number of sessions connecting to the JMS server. The problem with this is that the messages are distributed to sessions in the order in which they connect. This means that a single process with a maximum number of sessions will receive only that number of messages maximum. In fact, the session will remain blocked until the process sends acknowledgement. Explicit Acknowledgement takes away requirements of setting maximum connection. It behaves like client acknowledgement, but the session is not blocked and one session handles all the incoming messages.

To explain this important subject we describe a real life client scenario where the response time is expected to increase as more and more hardware and engines are added to the overall architecture. This scenario involves a single TIBCO Enterprise Message Service server and multiple TIBCO BusinessWorks Engines on different machines. In order to improve response time, it is very important to understand the complex relationship between TIBCO Enterprise Message Service server, TIBCO BusinessWorks Engines, and how TIBCO Enterprise Message Service messaging and TIBCO BusinessWorks Job Starter protocol work.

The following scenario involves a single TIBCO Enterprise Message Service server publishing TIBCO Enterprise Message Service messages of the same size to three TIBCO BusinessWorks machines each running a single TIBCO BusinessWorks Engine. The following interaction will help you understand how the delivery of messages to queue consumers works in TIBCO Enterprise Message Service.

When a burst of messages arrives at the TIBCO Enterprise Message Service server, it attempts to deliver them to the consumers of the queue in a round-robin fashion. If there are 3 consumers, such as C1, C2, and C3, then the first message (Msg1) will be sent to C1, Msg2 to C2, Msg3 to C3, Msg4 to C1, and so on until either all the messages have been sent or all the consumers have reached their `prefetch` limit. If the latter is the case, then the subsequent messages are buffered in the TIBCO Enterprise Message Service server. To keep the sums simple, assume that the `prefetch` value for the queue has been set to 6. Assuming that the burst contained at least 18 messages, consumer C1 will have received messages Msg1, Msg4, Msg7, Msg10, Msg13 and Msg16. C2 will have received Msg2, Msg5, Msg8, Msg11, Msg14 and Msg17. C3 will have received Msg3, Msg6, Msg9, Msg12, Msg15 and Msg18.

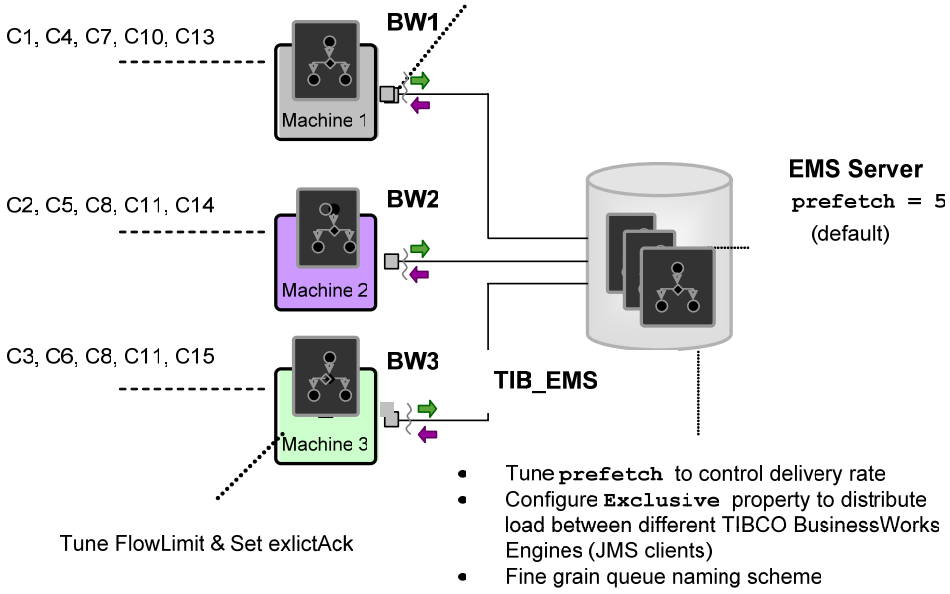

Figure 9. Delivery of Messages to Queue Consumers

4.3.1 Relationship With Prefetch

When you are using an asynchronous publish/subscribe call between the JMS server and TIBCO BusinessWorks Engine, tuning prefetch to higher value may be beneficial. This value may be configured in TIBCO Enterprise Message Service configuration within `queues.conf`.

The best practice for achieving horizontal tuning involves the following rules:

Rules of Thumb for TIBCO Enterprise Message Service Usage for Better Horizontal Scaling

- TIBCO Enterprise Message Service
  - Use `Explicit Ack` with `FlowLimit` with multiple TIBCO BusinessWorks Engines for the horizontal scaling. This is more efficient than client `ack`
- Non TIBCO Enterprise Message Service
  - Use `Client Ack`.
  - Maximum number of sessions should be set higher than the number of threads
- For Client Ack send confirmation as soon as possible to avoid blocking session
- To control memory growth or protect against fast producer situation Use `FlowLimit`
  - `Client Ack`: Hold confirmation to the end of process
- Once `FlowLimit` and `ExplicitAck` are set, configure `prefetch` (usually between 5 and `FlowLimit`) for optimized delivery control rate

5 Benchmarking and Testing Performance

5.1 Performance Benchmarking Process

Before you can gain a detailed understanding of TIBCO BusinessWorks Engine tuning, it is extremely critical to understand the most important concepts related to TIBCO BusinessWorks.

This section outlines the typical steps required to successfully evaluate and tune a TIBCO BusinessWorks environment. This process is meant to be used as a general guideline for each use case. Additional or fewer steps may be required, depending on individual factors. Some steps may also require multiple iterations. To use this information effectively, first eliminate external factors as a cause of any performance issues.

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

- Establish performance benchmarking criteria
- Review TIBCO BusinessWorks 5.x performance architecture
- Establish performance best practices guidelines
- Review tunable parameters

5.2 Performance Benchmarking Criteria

The first step in measuring performance is to identify the requirements of your business environment. Performance targets are typically determined by user response times and message processing requirements. Examples of typical performance requirements include:

- Engine throughput, or number of messages processed per second
- Processing speed, or average process instance duration and latency
- Web (e-commerce operation request response time) response time (request/response time)
- Resource utilization (CPU, memory, threads, sockets, process, file)
- Concurrent request, sleep time, users if applicable (registered and concurrent)

For each requirement, define the minimum, desired, and peak targets. This information allows you to identify the type of data to collect and to evaluate test results.

In addition to these normal load expectations, consideration should be given to abnormal error-recovery scenarios that may present unusually high loads. For example, the TIBCO BusinessWorks process might be pulling messages from a TIBCO Enterprise Message Service queue, and the above targets reflect the normal flow of messages through the queue. However, if communications to the TIBCO Enterprise Message Service server has been disrupted for an extended period of time, or if the TIBCO BusinessWorks Engine has been shut down, a much higher peak load may be experienced when communication is re-established or the engine restarted.

These scenarios must be addressed in considering TIBCO BusinessWorks Engine performance under load to ensure that the throughput does not deteriorate below the target in these overload situations. Business requirements also control the decision to use reliable or certified messaging. Certified messaging has an impact on performance.

5.3 Performance Testing Tools and Techniques

Once you have established appropriate goals for the 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. As mentioned earlier, key performance criteria vary significantly for data access and presentation views. Monitoring each component requires different techniques.

To monitor OS resources, you can use various OS-dependent tools such as PerfMon from Hewlett-Packard, TIBCO Hawk, SilkPerformer® from Segue, and various OS dependent utilities (for example, `Top`, `prstat`, `iostat`, `vmstat`, and so on). In addition, various systems trace & log files should also be monitored.

5.4 Collecting Performance Data

Using TIBCO Designer™, create a set of process definitions for testing purposes. These can be the actual process definitions that will be used in production, or more basic process definitions that mimic production scenarios. The granularity and scope of your performance requirements, should determine how closely process definitions are used for performance testing.

Configure the operating system tool to measure memory, disk, and CPU usage during all tests. Identify the TIBCO BusinessWorks metrics that will allow you to measure conformance with requirements. A general strategy is to begin with summary metrics, 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 needed information:

- To understand where process instance lifetime is spent, collect detailed process instance metrics while processing a few representative process instances.
- To 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. Where possible, restrict other local network, operating system, and engine activities during this time. If average metrics will be used, restrict the test window to ensure that data collected is relevant.

5.5 Deploying Performance Testing Framework

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

Once you have established the key performance matrix and determined tools and techniques to measure the components, the next step is to establish an appropriate framework for testing performance. Developing a framework depends on performance goals established earlier and business requirements. Overall requirements for data access layer and presentation layer performance testing are significantly different. For example, performance testing requires building an application that monitors overall throughput of the presentation layer at the user layer. This layer typically requires measuring above the established key matrix in the end-to-end manner.

Frequently, a customer would require building a script within third-party performance testing tools such as Mercury LoadRunner® or Segue's SilkPerformer. For the data access layer, these tools are frequently used to build extensible framework to invoke TIBCO Rendezvous, JMS, SOAP, or HTTP messages. Alternatively, simpler processes can also be coded with TIBCO BusinessWorks or `RvScript` from TIBCO Software.

5.6 Developing Performance Testing Plans

Developing a performance testing plan involves building an appropriate set of tests to meet business objectives. The section below provides series of tests that can be planned based on overall objectives.

5.6.1 Build a Baseline Test

For initial performance tests, consider using a simple scenario of a single engine and file-based data manager. After basic testing is complete, add more engines and a database data manager if your production environment will include these components. Then, repeat the tests for this more realistic scenario.

Begin collecting performance data using the TIBCO Administrator™, TIBCO Hawk and trace files. Details for these two techniques are described in the previous section. It is recommended that you Start and stop the performance data utility several times before running actual tests.

Perform tests for minimum, desired, and peak numbers identified in Step 1. Capture and store output using the third-party application. When the baseline tests are complete, stop the performance data collection utility, stop sending messages, and then stop the engine.

5.6.2 Compare Baseline to Targets

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

5.6.3 Build Stability Test

Frequently, many performance issues can be identified in the stability test where by application is deployed under lower load condition such as 5 to10 concurrent users with pre-established think time. This test will focus on end-to-end successful transactions compared to measuring overall response time. Since the test system involves various components, it is important to get familiar 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 (sessions) if applicable
- Monitor any incomplete TIBCO BusinessWorks jobs
- Worst performing activities (CPU time, Elapsed Time)

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

5.6.4 Develop Incremental Tests

The user should 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.

5.6.5 Develop Peak Rate Tests

Overall business objectives can be different for each project. If it is desired to measure the system against an extreme number of users and not fail, a peak load test can be designed to determine whether the system can respond to the desired maximum number of users and requests without degradation in response time.

5.6.6 Develop Steady State Tests

This test is often desired when the business objective requires providing well established QoS for the business users, such as the number of concurrent users with the least amount of response time. Steady state keeps steady capacity or steady number of requests/min even if concurrent users keep increasing. This test often focuses on maintaining partner QoS and capacity and not peak number of users.

5.6.7 Develop Hardware and Resource Plan

The choice of a proper OS with an 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 ideally consider different operating systems, number of CPUs, and other hardware resources.

5.6.8 Develop Component Deployment Plan

The test plan should ideally extend results obtained in the previous test and design for the optimal production deployment. In this phase, based on test results obtained in the previous test, ultimate production deployment is planned. Possible steps are: 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.

5.7 Identifying Potential Improvement Areas

If performance requirements are not met and improvement is needed, possible modifications may include:

- Adding hardware resources
- Modifying JVM parameters
- Increasing engine threads
- Running multiple engines
- Reducing message size, message burst size, or message frequency
- Modifying process definition design or activity settings

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

5.8 Implementing Specific Enhancements

Typically, scaling involves adding hardware resources or engines. Tuning involves changes to engine property files or scripts and changes to process definition design in TIBCO Designer. 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 a graduated way. In other words, a reasonable approach is to implement the first modification, 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 modifications are related to the same resource, memory usage. If the engine requires more memory, then increasing the number of threads in JVM would be ineffective. However, some scenarios might require measuring the effect of each modification separately, by backing out one change before implementing another.

5.9 Comparing Results

After implementing modifications and repeating test scenarios, compare the adjusted results to baseline test results. Exporting performance data to a third-party application can simplify this step.

If performance improved because of the modifications, compare the adjusted results to performance requirements. If requirements are met, begin testing the next use case. If requirements are not met, you should repeat step 5.6.4 “Develop Incremental Tests” and step 5.6.5 “Develop Peak Rate Tests” for additional enhancements.

5.10 Testing and Debugging Performance Issues

5.10.1 TIBCO Administrator GUI

TIBCO Administrator is a good tool for taking a quick preliminary performance snapshot. Using its GUI, you can quickly observe performance related issues. Since numbers are reported in milliseconds, it is critical that conclusions are based on a relatively high number of Execution Counts (see below). You must also be cautious when interpreting the Elapsed Time measurement: it is the time difference between entering the activity and ending the activity. However, the particular activity may have been blocked due to various reasons such as sleep, wait task, or perhaps thread blocking. While not a perfect measure, Elapsed Time gives an indication whether something is wrong within a business process or activity.

Table 2. Throughput View from TIBCO Administrator

Name ↓	Execution Count	Elapsed Time (ms)	CPU Time (ms)	Errors	Status	Function
start	600	11	11	0	OK	`ProcessGroup`
Open Connection	600	2282243	2282245	0	OK	`WfPluginManageConnectionActivity`
AcquireTask	600	929649	929649	0	OK	`WfPluginManageTaskActivity`
close Connection	600	212226	212228	0	OK	`WfPluginManageConnectionActivity`
End	600	295	297	0	DEAD	`ProcessGroup`

This table shows information from the TIBCO Administrator. For more information, see the *TIBCO Administrator User's Guide*.

Negatives

- Does not show detailed statistics
- Gives cumulative output, but not per Job

5.10.2 TIBCO Hawk

TIBCO Hawk provides the same statistics as described by TIBCO Administrator. You should export the information to an Excel file for further analysis. This test has the following characteristics:

- Provides all the information as TIBCO Administrator plus more parameters
- Subscribes to `GetActivities` and exports to Excel

5.10.3 Trace Files and Trace Analyzer

Upon identifying the specific bottleneck in the activity, it is best to turn on all the tracing using the TIBCO Administrator GUI.

Tracing will add additional lag due to I/O required for writing the trace, but it will still be a very useful tool for finding further bottlenecks. The trace output gives information based on each job. While it is verbose, you can follow the activities that have been identified as the bottleneck during the previous step. The trace output will give you a very comprehensive view of the flow of an individual job. It may be easy to determine if an activity is spending too much time or is blocked due to various reasons.

- Most detailed and meaningful
- Can analyze various scenarios
- Get a complete trace output of the test run
- Run the trace through trace analyzer (utility `bwTraceFormatter`)
- Sort by `CPU Time` to see all worst performing
- Sort by `job-id` and `CPU Time`
- Also look at `Elapsed Time`

Whenever there is a big differential between `Elapsed Time` and `CPU Time`, you must understand all the interactions within the activity.

Note

Be aware that turning on trace will distort the overall performance number; however, this process is still useful for debugging activity-level performance issues.

5.10.4 Statistics Tips

In TIBCO BusinessWorks 5.2, Engine Command activity has been added to allow you to do basic management and statistics reporting within a process definition. See *TIBCO BusinessWorks Palette Reference*, Chapter “General Activities Palette”, section” Engine Command activity” for more information.

Rules of Thumb

Basic management and statistic reporting within a process definition should be used in the following cases:

- When TIBCO Administrator is not used in the environment
- In a scripted deployment environment

6 Configuration

6.1 Configuring Persistent Connections

Persistent connections are created for each HTTP server that Send HTTP Request activities in process instances communicate with. Each HTTP Client holds a persistent connection until the HTTP server sends the response message. The connection is then released by the client and returned to the pool.

You can specify the maximum number of connections to create in the persistent connection pool, and you can also specify the maximum number of persistent connections for each HTTP server. Connections for each HTTP server are created in the pool until the maximum is reached.

When a Send HTTP Request activity requires a connection, the pool is searched for a connection that corresponds to the HTTP server. If a corresponding unused connection is found, it is used. If no connection is found to a corresponding HTTP server, a new connection is created if the maximum pool size has not been reached. If the maximum number of connections for that particular server has been reached, the request must wait for a connection to be released before using it.

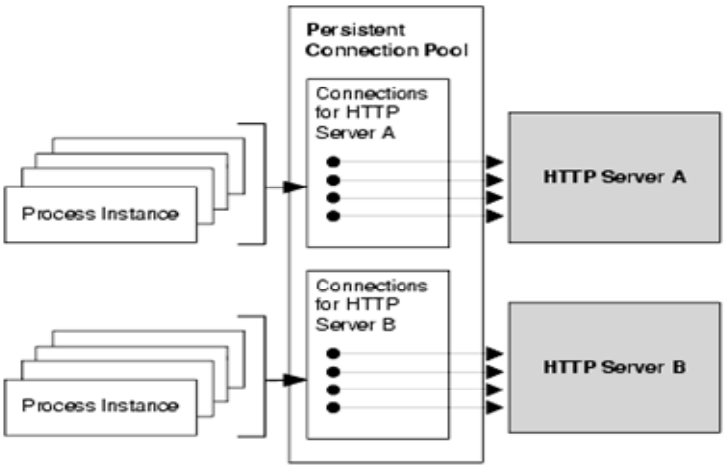

Figure 10. Persistent Connection Pool

Persistent connections are managed by custom engine properties. See *TIBCO BusinessWorks Process Design Guide,* section “Available Custom Engine Properties” for more information.

6.1.1 Engine Properties that Control Persistent Connections

6.1.1.1 bw.plugin.http.client.usePersistentConnectionManager

This property specifies that a pool of HTTP connections to each HTTP server should be created so that connections can be reused by Send HTTP Request activities. Not all HTTP servers support persistent connections. Refer to your HTTP server documentation for more information about support for persistent connections.

When this property is set to true, a pool of connections is created for each HTTP server that the HTTP (SOAP/HTTP) Request-Reply activities connect to. The total number of connections in the pool is limited by the `bw.plugin.http.client.maxTotalConnections` property. The number of connections for each host is limited by the `bw.plugin.http.client.maxConnectionsPerHost` property.

The default value of this property is false.

6.1.1.2 bw.plugin.http.client.maxConnectionsPerHost

The value of this property is ignored unless the `bw.plugin.http.client.usePersistentConnectionManager` property is set to true. This property specifies the maximum number of persistent connections to each remote HTTP server.

The default value for this property is 20.

6.1.1.3 bw.plugin.http.client.maxTotalConnections

The value of this property is ignored unless the bw.plugin.http.client.usePersistentConnectionManager property is set to true. This property specifies the maximum number of persistent connections to create for all HTTP servers.

The default value for this property is 200.

6.1.1.4 bw.plugin.http.client.checkForStaleConnections

The value of this property is ignored unless the bw.plugin.http.client.usePersistentConnectionManager property is set to true. When using persistent connections, a connection can become stale. When this property is set to true, a persistent connection is checked to determine if it is stale before it is used by an HTTP (SOAP/HTTP) Request-Reply activity. Checking for stale connections adds significant processing overhead, but it does improve reliability.

The default value for this property is false.

6.1.1.5 bw.plugin.http.client.ResponseThreadPool

The HTTP (SOAP/HTTP) client uses a thread pool for sending the HTTP messages. Each HTTP request is sent out in a separate thread in order to not keep the engine's thread blocked while waiting for the response message. These threads are taken from a thread pool. Each HTTP (SOAP/HTTP) Request-Reply activity has its own thread pool. The thread pool's size can be configured using this property.

The default value for this property is 10.

6.1.2 Configuration Considerations for Non-Persistent Connections

Many operating systems by default are tuned for the limited numbers of `UserPort` parameters. Generally, this parameter corresponds to the number of sockets open at a time. For example, on Windows, `MaxUserPort` specifies the ephemeral port numbers (by default, Windows allows 1024-5000).

Another important parameter is `TcpWaitedTimeDelay`, which is normally set to 30 seconds. This parameter determines the length of time that a connection stays in the `TIME_WAIT` state when being closed. While a connection is in the `TIME_WAIT` state, the socket pair cannot be reused. This is also known as the 2MSL state because the value should be twice the maximum segment lifetime on the network.

For scenarios where an HTTP server is using non-persistent connections (such as a Web server prior to HTTP 1.1 or HTTP 1.1 without the HTTP Persistent support), tuning should be very carefully understood on the Windows platform.

For example, on Windows 2000, the following two parameters should be carefully tuned:

6.1.2.1 MaxUserPort

Key: Tcpip\Parameters
Value Type: `REG_DWORD-maximum port number`
Valid Range: 5000-65534 (decimal)
Default: 0x1388 (5000 decimal)
This parameter controls the maximum port number used when an application requests any available user port from the system. Normally, short-lived ports are allocated in the range from 1024 through 5000. Setting this parameter to a value outside of the valid range causes the nearest valid value to be used (5000 or 65534).

6.1.2.2 TcpTimedWaitDelay

Key: Tcpip\Parameters
Value Type: `REG_DWORD-time in seconds`
Valid Range: 30-300 (decimal)
Default: 0xF0 (240 decimal)
This parameter determines the length of time that a connection stays in the `TIME_WAIT` state when being closed. While a connection is in the `TIME_WAIT` state, the socket pair cannot be reused. This is also known as the 2MSL state because the value should be twice the maximum segment lifetime on the network. See RFC 793 for further details.

Please note that other operating systems may have similar limiting parameters. Hence, you should consider tuning those parameters when similar limitations are encountered.

6.1.3 Detecting Persistent Connection

There are several public domain connection manager tools available to monitor progress of the connections. TIBCO Software recommends that during performance testing you monitor progress of connection using one of these tools.

6.2 Configuring HTTP Servers

In some situations, you may wish to alter the configuration of the HTTP server that receives incoming HTTP requests for TIBCO BusinessWorks.

There are two custom properties that you may configure:

- **`bw.plugin.http.server.minProcessors`**. This is the minimum number of threads available for incoming HTTP requests. The HTTP server creates the number of threads specified by this parameter when it starts up.
- **`bw.plugin.http.server.maxProcessors`**. This is the maximum number of threads available for incoming HTTP requests. The HTTP server will not create more than the number of threads specified by this parameter. This limit is useful for determining number of incoming requests that can be processed at a time. Setting a high number will simply create that many threads and eventually reduce the performance drastically.

Note

If you have a large number of incoming requests, you may wish to change these values to handle more incoming requests concurrently. By default, the minimum number of threads is 10 and the maximum is 75.

You can increase these numbers only if you have a higher peak concurrent request requirement, and you have large enough hardware resources to meet the requests.

To change these values, add the appropriate property to the `bwengine.tra` file. When you deploy your project using TIBCO Administrator, these properties will automatically become part of the deployment configuration.

See *TIBCO Administrator User's Guide* for more information about deploying your project.

Note

When a client sends a request that cannot be processed because no threads are available, TIBCO BusinessWorks returns a `ConnectionRefused` exception to the client.