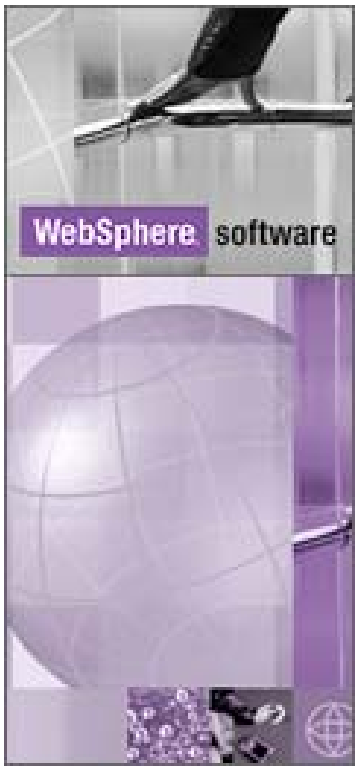
WebSphere Process Server V6.1 – Business Process Choreographer:

Performance Tuning Automatic Business Processes for Production Scenarios with DB2



Dec 2008

Thomas Muehlfriedel
Gerhard Pfau
Jonas Grundler
Dominik Meyer

Abstract

This document complements existing performance white papers, Red Books, and product information about WebSphere* Process Server. While the “WebSphere Business Process Management V6.1Performance Tuning” Red Paper [BPMPerf] provides broad coverage of performance tuning and best practices across WebSphere BPM products, this document provides a deep dive into performance tuning when using automatic business processes in Business Process Choreographer. Please see chapter 7 for related resources.

WebSphere Process Server supports two different types of business processes: Human workflows and automatic business processes. Human workflows are business processes that involve human beings on their main path of execution. Automatic business processes are run in a straight-through processing way, fully automatic, that is without human involvement on the main path. This paper concentrates on performance tuning automatic business processes, but it also contains best practices for Web client configuration, such as the BPC Explorer.

There are two categories of automatic business processes: long-running processes and microflows. The paper first discusses some general best practices for performance tuning, then describes how to tune microflows, and finally covers in detail the performance tuning mechanisms available for long running business processes. These mechanisms include tuning guidelines for the underlying databases, for the message queues involved, and for the components within WebSphere Process Server like the Business Flow Manager, the BPEL engine in WebSphere Process Server.

The target audiences of this paper are administrators, architects, and solution specialists that create and manage business process based applications on WebSphere Process Server.

Table of Contents

1 Overview

Business Process Choreographer (BPC) provides business process support within WebSphere* Process Server (WPS). Business processes consist of multiple steps, so called *activities*, and the connections between these activities. Business Process Choreographer supports two different types of business processes: Human workflows and automatic business processes. Human workflows are business processes that involve human beings on their main path of execution. Automatic business processes are run in a straight-through processing way, fully automatic. Sometimes they also involve humans, but only for exception handling. This paper concentrates on performance tuning automatic business processes. Performance tuning for human workflows includes tuning tips described in this document; specifics to it will be described in a later publication.

There are two categories of automatic business processes: *long-running processes* and *microflows*.

Long-running processes are interruptible. Each activity in the process can run in its own J2EE transaction. One transaction encloses one or many steps in the business process. The J2EE transactions are chained together using persistent messaging (JMS based navigation) or messages placed in the BPEDB database (WorkManager based navigation). Transaction boundaries can be optimized by the developer of the business process. After each transaction, the state of the business process is written into the BPC database. Long-running processes survive system failures and support parallel execution of activities that are on parallel paths within a business process.

Microflows run within a single J2EE transaction on the same thread from start to end without interruptions. They are light-weight compared to long-running processes because they do not need any persistent intermediate state to be written to a database nor do they send any persistent messages while navigating.

For a more detailed explanation of the differences between these types of business processes refer to the *WebSphere Process Server V6 – Business Process Choreographer: Concepts and Architecture* white paper [Concepts] referenced in the resources section of this document.

Due to the different nature of long-running processes and microflows you must perform different actions to optimize them. Tuning for best performance of microflows is described in the section *Best Practices for tuning microflows* whereas the tuning of long-running processes is discussed in the section *Best Practices for long-running processes*.

This paper presents best practices developed in a lab environment. These best practices may or may not apply to all production scenarios and topologies. It is suggested that each best practice is tested for applicability before it is applied to a production environment.

Business Process Choreographer can be used together with different database management systems. In this paper DB2* on distributed systems (Windows*, AIX*) is used. Recommendations may or may not apply to other DBMS systems.

2 General Tuning Information

Before covering the performance tuning specifics of the different types of business processes, this chapter summarizes some general tips for choosing hardware when setting up the systems.

2.1 Processor Cache

A best practice for database servers is to use processors (CPUs) with large processor caches and cache hierarchies with L2 and L3 caches of substantial size.
For application servers, the processor cache has a significant performance impact as well. For WebSphere Process Server we therefore also recommend using processors with large caches in the cache hierarchy (e.g., CPUs with L2 and L3 Cache of four MBytes and more).

2.2 Java* Virtual Machine Heap Size

Increasing the heap size of the Java Virtual Machine (JVM) of the application server can improve the throughput of business processes. Nevertheless, be cautious to keep it low enough to avoid heap swapping to disk. The BPM Performance Red Paper contains detailed information on how to tune JVM parameters. For details see [BPMPerf].

2.3 Garbage Collection Strategies

The JVM used by WebSphere Process Server V6.1 supports several garbage collection strategies. The three most famous garbage collectors are the *Throughput Garbage Collector*, the *Low Pause Garbage Collector*, and the *Generational Garbage Collector*.

The *Throughput Garbage Collector* provides the best out-of-box throughput for applications running on a JVM by minimizing costs of the garbage collector. However, this garbage collector has "stop-the-world" phases which cause the whole JVM to stop processing requests. Depending on the heap size and other factors the JVM stops processing requests for times between 100ms and multiple seconds during garbage collection.
If "mark stack overflows" are noticed in the verbose gc log consider increasing parallelism of the garbage collector threads as "mark stack overflows" can negatively impact performance. Use the "-Xgcthreads*n*" switch of the JVM in order to allow a parallelism of the GC threads greater than the number of processors. For a single CPU machine, set *n* to 4, e.g.

The *Low Pause Garbage Collector* provides garbage collection in parallel to the JVM's work. Due to increased synchronization costs, throughput decreases. Typically, if response time is more important than highest possible throughput, this garbage collector is the best choice.

The *Generational Garbage Collector* is new in the IBM 1.5 JVM. It is well suited for applications that produce a lot of short-lived small Java objects. As it reduces pause times it should be tried in such cases instead of the *Throughput* or *Low Pause* GC. When properly tuned, it provides the best Garbage Collection performance for some workloads.

See [DiagnoseJVM] on details of the IBM 1.5 JVM.

2.4 Database Connections and JDBC*

2.4.1 JDBC

In general, use JDBC type 4 drivers for all database connections. However, if you have installed your database on the same machine as your application servers consider using the type 2 “DB2 Universal JDBC Driver Provider” in order to get better performance.

2.4.2 Auditing

If you are using the audit log in Business Process Choreographer (e.g. to log the run of a business process) then please be aware when configuring the connection pools that a database connection to the BPEDB is needed to write to the audit log.

2.4.3 Common Event Infrastructure

If Business Process Choreographer is configured to emit events to the Common Event Infrastructure (CEI), a connection to the CEI resource is needed. For an asynchronous event factory profile, this is a JMS connection. If the CEI data store is enabled and synchronous event submission is used, a connection to the CEI data store is needed.

2.5 Thread Pools

Thread pools are used to control the parallel number of requests that are processed by a single JVM. There are 4 thread pools that must be considered, dependent on the load characteristics of your application:

- ORB Thread Pool: used to process remote EJB requests coming from a different JVM
- Web container Thread Pool: used to process http requests including SOAP over HTTP requests
- Default Thread Pool: used to process other requests such as those coming from persistent and non-persistent messages. This thread pool must be tuned if BPC uses JMS based navigation.
- WorkManager specific Thread Pools: If BPC is configured to use WorkManager based navigation, tune the BPENavigationWorkManager thread pool.

For further application server tuning information refer to the WebSphere Information Center at [TuneWAS] and [BPMPerf].

3 Best Practices for Tuning Microflows

Microflows are light-weight compared to long-running processes. They do not require any database access or persistent messaging support while navigating. The entire processing occurs in one single transaction. Because of these characteristics non-interruptible processes usually perform better than their interruptible counterparts in terms of throughput and response time. However there are restrictions with regard to supported types of activities and other limitations as explained in the *WebSphere Process Server V6 – Business Process Choreographer: Concepts and Architecture* white paper [Concepts] referenced in the resources section of this document.

In most cases, tuning microflows is very similar to tuning general enterprise applications running in a WebSphere application server. The most important tuning parameters that affect microflow performance are JVM configuration and Thread Pool size.

3.1 General considerations on system layout

3.1.1 Database connections

In microflows, database connections to the BPEDB are only needed if audit logging is enabled. If so, make sure that the database can handle the number of parallel database connections that are created by parallel running microflows and increase the number of connections on the BPEDB data source.

3.1.2 Thread Pool Sizes

While long-running business processes spend most of their lifetime in the default thread pool (JMS based navigation) or WorkManager thread pool (WorkManager based navigation), microflows don't have a specific thread pool assigned to it.
Dependent on from where the request to run a microflow comes from, a microflow runs within:

- the ORB thread pool (e.g. the microflow is started from a different JVM with remote EJB invocation)
- the Web container thread pool (e.g. the microflow is started using a http request)
- the default thread pool (e.g. the microflow is started using a JMS message)

If microflow parallelism is not sufficient, examine your application and increase the respective thread pool. They can be found at:

Application Servers →server →Thread Pools

The key is to maximize the concurrency of the end-to-end application flow until the available processor or other resources are maximized. BPM performance red paper [BPMPerf] contains more advanced tuning tips in this area.

4 Tuning Long-running Processes

The performance of long-running processes is to a large degree dependent on a properly tuned, enterprise class database management system, and well tuned application servers. This paper provides tuning guidelines using IBM* DB2* Universal Database (UDB) V8.2.2 as an example. Most of the rules should also be applicable to other production database management systems. According to the experience of the authors it is not advisable to use Cloudscape or Derby as a database management system for WebSphere* Process Server other than for the purpose of unit testing.

With WPS 6.1 [1], a new technique is available for navigating business processes which is called **WorkManager based navigation** (see chapter 4.4.2 for details). Consider to use this navigation technique in order to speed up your environment.

For good BPC API query performance, steps as described in chapter 4.2.6 are required.

The following describes how to tune a basic production setup of two machines. The steps are ordered according to best practices, that is creating of the process database and performing basic tuning, then creating and performing basic tuning of the messaging engine databases. Following these preliminary steps a chapter explains the basics of how the Business Process Choreographer engine works in order to motivate the following tuning guidelines.

4.1 General considerations on system layout

Before starting to tune a system, verify that the computers used are well balanced for the task, i.e., that the available resources CPU, memory and I/O have the right relationship. A computer with one (or many) very fast CPUs but low memory or low I/O performance will be hard to tune. For long-running processes high I/O performance on the database side in the form of multiple, fast disk drives (RAID arrays, SAN) are as important as enough processing power and a sufficient amount of memory.

For a small production scenario WebSphere Process Server application servers and the database can run on the same physical machine. Besides fast CPUs and enough memory, the speed of disk I/O is significant. As a rule of thumb we suggest to use between 1 and 2 GB of memory per JVM and as much memory as you can afford for the database. For the disk sub-system use a minimum of two disk drives (one for the transaction logs of WebSphere and the database, another one for the data in the database and persistent queues). More disk drives allow you to better tune the system. Alternatively, for a care-free setup use a RAID system with as many drives as you can afford.

For larger production systems it is advisable to use a WebSphere cluster of WebSphere Process Server machines running the business processes, and a separate machine for the database. In such a configuration the machine running WebSphere Process Server is sufficiently well equipped with two physical disks, one for the operating system and one for WebSphere Process Server, in particular for the WebSphere transaction log. In case the transaction log becomes a performance

[1] WorkManager based navigation can be configured since WPS 6.1.0.1. In cluster installations, apply i-Fix IZ21335

bottleneck, striped disks can be used to increase the write throughput as described in *Tuning Logical Volume Striping* [TuneStriping]).
The machine running the database management system (DBMS) requires fast and many CPUs, large CPU caches, a large amount of physical memory, and good I/O performance. Consider using a 64-bit system to host the database. 64-bit instances of the database can address much more memory than 32-bit instances. Larger physical memory improves the scalability of the database. Thus using a 64-bit instance is preferable.
To get fast I/O consider using a fast storage sub-system (NAS or SAN). If you are using a setup with individual disks then a large number of disks is advantageous. The example below shows a configuration that uses twelve disk drives:

- One disk for the operation system and the swap space (pagefile on Windows, paging space on AIX, swap space on Solaris, paging space on HP/UX)
- Two disks for the transaction log of the BPC database, either logically striped or preferably hardware-supported striped for lower latency and higher throughput.
- Four disks or for the BPC database table spaces in the database management system. If more disks are available the instance tables should be distributed on three or more disks. If staff activities are required, additional considerations have to be taken into account.
- Two disks for the transaction log of the Messaging database, again, these two should be striped as explained above.
- Three disks for the messaging database table spaces. These can be striped and all table spaces can be spread out across the array.

In scenarios where ultimate throughput is not required, a hand full of disks is sufficient to achieve acceptable throughput. When using RAID arrays instead of individual disks the optimum number of disks is higher, taking into consideration that some of the disks are used to provide fail-over capabilities.

4.2 Tuning the Business Process Choreographer Database

To get good performance you must carefully create the Business Process Choreographer database, and then tune it. The following sections describe how to create the database and how to tune it.

4.2.1 Creating the Database Configuration

In production scenarios it is desirable to control the way the data in the database is distributed over multiple disk drives. This section explains how to use table spaces to accomplish that.

Default table spaces

Scripts to create the database and table spaces are provided by Business Process Choreographer in the ProcessChoreographer subdirectory of your WebSphere Process Server installation. These scripts must be customized to accommodate your particular scenario. When creating the table spaces, try to distribute read/write operations over as many disk drives as possible.

By default the scripts create the following table spaces:

- The TEMPLATE table space contains the process templates. As template information is cached at runtime after the initial read, this table space is usually accessed less frequent. It is a good candidate for being put on the same drive as a table space that has a higher access rate.
- The STAFFQRY table space contains the tables that are used to store staff query results obtained from staff repositories like LDAP. When using many staff activities in business processes then tables in this table space are frequently accessed.

- The AUDITLOG table space contains the audit trail tables. If auditing using the BPC Audit Log facility is turned off, then tables in this table space are not accessed. Depending on the degree of auditing used access to tables in this table space may be significant though.
- The COMP table space is used for the compensation tables. If compensation is not used within business processes then tables in this table space are not accessed. Depending on the number of compensable processes and activities, the tables in this table space may have high access rates.
- The INSTANCE table space holds the process instance tables. It is heavily used whatever kind of interruptible process is run. If possible, spread this table space over multiple disk drives.
- The WORKITEM table space holds the tables required for processing work items. Work items are used by people interacting with business processes. The access load on this table space depends on the number of people activities in the business processes which can vary from very low to very high. The access rate will never be zero even when no people activities are used, because work items are also generated to allow the administration of interruptible processes.
- The SCHEDTS table space contains the table used by the WebSphere scheduling component that is used by BPC. Because of caching mechanisms in the scheduler, access to the tables in the scheduler table space is normally low.

The default way to create the database for Business Process Choreographer is to either use the *Business Process Container Install Wizard* or to run the script *createDatabaseDb2.ddl.* In both cases the database, the table spaces, tables, views, etc. are created in a default way. To maximize the options for performance tuning, manual control of the database creation and configuration is recommended.

Creating the database only

First create the database only. The example below shows a command that creates the BPC database in the directory /databases/BPE on a UNIX* platform.

```
CREATE DATABASE BPEDB ON /databases/BPE USING CODESET UTF-8 TERRITORY en-us;
```

On Windows, only the target drive letter can be given; the command creates the BPC database in the directory D:\DB2 if the default instance is used.

```
CREATE DATABASE BPEDB ON D: USING CODESET UTF-8 TERRITORY en-us;
```

Create table spaces

Next create the table spaces on the desired disks. Use the information given in the section on default table spaces before choosing how to spread the table spaces across your disks. The example below shows a script based on the file *createTablespaceDb2.ddl* located in the ProcessChoreographer subdirectory of your WebSphere Process Server installation. It creates table spaces using three different disk drives on a Windows system:

```
--
-- Licensed Materials - Property of IBM
-- 5655-FLW (C) Copyright IBM Corporation 2005. All Rights Reserved.
-- US Government Users Restricted Rights- Use, duplication or
-- disclosure
--
-- Scriptfile to create tablespaces for DB2 UDB
```

```
-- Replace occurence or @location@ in this file with the location
-- where you want the tablespace containers to be stored, then run:
--      db2 connect to BPEDB
--      db2 -tf createTablespaceDb2.ddl


CREATE TABLESPACE TEMPLATE MANAGED BY SYSTEM USING( 'D:/BPE/TEMPLATE' );

CREATE TABLESPACE STAFFQRY MANAGED BY SYSTEM USING( 'D:/BPE/STAFFQRY' );

CREATE TABLESPACE AUDITLOG MANAGED BY SYSTEM USING( 'E:/BPE/AUDITLOG' );

CREATE TABLESPACE COMP MANAGED BY SYSTEM USING( 'D:/BPE/COMP' );

CREATE TABLESPACE INSTANCE MANAGED BY SYSTEM USING( 'E:/BPE/INSTANCE');

CREATE TABLESPACE WORKITEM MANAGED BY SYSTEM USING( 'F:/BPE/WORKITEM' );

CREATE TABLESPACE SCHEDTS MANAGED BY SYSTEM USING( ' F:/BPE/SCHEDTS' );
```

Create tables

After completing all previous steps the Business Process Choreographer database schema can be created. This is accomplished by running the script provided for the respective database. For DB2 use the *createSchemaDb2.ddl* script in the ProcessChoreographer directory of your WebSphere Process Server installation.

4.2.2 Separating DB2 log and data

Putting the DB2 log on a disk drive that is separate from the data can improve performance, provided that a large enough number of disk drives are available. In small configurations, if only a small number of disks are available, then distributing the table spaces, as described in the previous section, is usually more beneficial than putting the DB2 log on a separate drive. For example, to change the location of the log files for a database named *BPEDB* to *F:\db2logs*, the following DB2 command can be used:

```
db2 UPDATE DB CFG FOR BPEDB USING NEWLOGPATH F:\db2logs
```

After completing the initial setup and configuration steps it is recommended to prepare the database for the expected load. On DB2 this is done by running the DB2 Configuration Advisor, as explained in the next chapter.

4.2.3 Tuning the Database Using the DB2 Configuration Advisor

DB2 comes with a built-in configuration advisor. After creating the database, the advisor can be used to configure the database for the usage scenario expected. To run the DB2 Configuration Advisor open the DB2 Control Center and launch the DB2 Configuration Advisor from the context menu of the BPC database. The input for the Configuration Advisor depends on the actual system environment, load assumptions, etc. The following sections describe the individual steps when running the Configuration Advisor and provide guidance on which input to provide.

Memory

When asked for the amount of memory to be made available for DB2, please calculate according to the following formula:

Physical memory

– Memory for Operating System
– Memory for Process Server (in scenarios with a single server)

= Memory for DB2

To obtain good performance never allocate more memory to the respective applications than there is physical memory. If the system is configured to exceed the physical memory size, the operating system will swap memory pages out to disk, which severely impacts system performance. *Note that the numbers suggested here are upper limits and should not be exceeded in order to avoid paging.* It might not be necessary or useful to allocate huge amounts of memory to the DBMS when the data volume is relatively small compared to the memory size.

Example 1 for a Windows DBMS system:

3 GB physical memory
– 512 MBytes Memory for Operating System
= 2.5 GBytes Memory for DB2

Note: On 32-bit Intel architectures and 32-bit Windows the process address space is limited to 2 GBytes, unless the hardware supports the Physical Addressing Extension mechanism (PAE) specified by Intel, and the appropriate version of Windows is used.

Example 2 for an AIX 64-bit DBMS system:

8 GBytes physical memory
– 512 MBytes Memory for Operating System
= ~ 7 GBytes Memory for DB2

Use the following suggestions to answer the questions from the DB2 Configuration Advisor.

Type of workload

When asked for the type of workload, select a mixed workload (queries and transactions).

Typical transaction

Specify how a typical transaction against the database looks like. For the average number of SQL statements per unit of work, select *more than 10*. This indicates that long transactions are used. To estimate the number of transactions per minute in your database estimate the activity throughput that the system is expected to deliver. Assuming that each activity is processed within its own transaction (which is over-conservative), the number of transactions per minute can be calculated according to the following formula:

number of transactions per minute = number of activities per second * 60

If for example an activity throughput of 30 activities per second is the target, then the number of transactions, that the database system has to perform is

number of transactions per minute = 30 activities per second * 60 = 1800

If the initial throughput estimation proves to be wrong, you might have to return to database tuning again later. If throughput measurements result in lower numbers while CPU utilization is high (above 80%), try with a lower number. If throughput measurements are above the original estimate, and the CPU utilization is low (below 60%) and the disk drives are not running at their maximum, you may try with a higher number.

Database administration priority

For optimum throughput, it is recommended to tune the database for faster transaction performance and slower recovery.

Data

The initial tuning of the database is performed without data in the database. Subsequent tuning steps should be performed on a populated database. For example, for DB2 updating access statistics using the *runstats* command is necessary to allow the optimizer to improve SQL performance.

Number of applications

The number of local applications connected to the database highly depends on the amount of parallelism in the entire system, not only the database. The amount of parallelism affordable is limited by memory and CPU resources. Details are discussed in the section on WebSphere tuning below. As a base line on a 2-way Intel based server with 1.7GHz CPU clock speed and 3GBytes of memory **N=50** parallel connections to the database produced good results. This value can also be used as a starting point on larger nodes. To be able to offer **N** number of connections to WebSphere, DB2 should be configured for **N + 10%** local applications.

If Business Process Choreographer runs on the same physical computer as the database, then it does not require remote database connections. Remote connections may be required for remote database management though. In this case it is advisable using a low value instead of zero. If Business Process Choreographer and database are installed on separate computers then the number of remote applications should be set to N + 10%.

Isolation level

The isolation level used by BPC is *read stability*. . Note that the DB2 Configuration Advisor does not make any changes to isolation level settings on the database; the isolation level you specify when running the Configuration Advisor is taken into account to determine the recommendations.

At the end the configuration advisor shows the changes that it suggests. These changes can either be applied immediately or be saved to a file and applied later.

4.2.4 Adjusting DB and DBM settings according to workload requirements

After the configuration advisor has configured the database, you may apply additional tuning by modifying the following settings.

MINCOMMIT

A value of ‘1’ is strongly recommended. The DB2 Configuration Advisor sometimes suggests other values.

NUM_IOSERVERS

The value of NUM_IOSERVERS should match the number of physical disks the database resides on. You should have at least as many IOSERVERs as you have disks. IOSERVERs do not use many system resources, so rather use a value that is too high than too low.

NUM_IOCLEANERS

Especially on multi-processor machines, enough IO cleaners should be available to make sure that dirty pages in the bufferpool are written to disk. Provide at least one IO cleaner per processor.

4.2.5 Updating DB2 Database Statistics

Optimal database performance requires the database optimizer to do its job well. The database optimizer acts based on statistical data about the number of rows in a table, the use of space by a table or index, and other information. Upon first run of a system the statistics will not represent the actual data distribution and load pattern. As a consequence the optimizer takes sub-optimal decisions, leading to poor performance. Therefore after initially putting load on your system, or whenever the data volume in the database changes significantly, you should update the DB2 system catalog tables containing the statistics by running the RUNSTATS utility. Make sure there is sufficient data (> 2000 process instances) in the database before you run RUNSTATS the first time.

Avoid running RUNSTATS on an empty database as this might severely impact database statistics in a negative way.

Create RUNSTATS Script

Running RUNSTATS is best done using a script customized for the respective database. The example below shows how to generate such a script.

Assuming that you are logged on as user *bpeuser* using password *password* and you are connected to the database *BPEDB*, the following DB2 commands generate a Windows command file that can be executed in order to update statistics for all tables in the relevant table spaces in the Business Process Choreographer database *(BPEDB)*. The TEMPLATE table space tables are omitted because the information there is not accessed and updated frequently.

```
db2 -x "select 'db2 runstats on table '
        concat rtrim(tabschema)
        concat '.'
        concat tabname
        concat ' with distribution and detailed indexes all '
from syscat.tables
where
        type='T'  AND
        TBSPACEID IN (select TBSPACEID from sysibm.systablespaces
              where TBSPACE IN ('INSTANCE', 'WORKITEM', 'STAFFQUERY',
               'AUDITLOG', 'SCHEDTS')) AND tabname not in ('SAVED_ENGINE_MESSAGE_B_T')" >
runStatsScript.cmd

echo db2 connect reset >> runStatsScript.cmd
```

The above statements work in the following way. In the first line we select from a system catalog table all table names for tables which reside in one of the relevant table spaces and generate the RUNSTATS command for each of them. The output is redirected to the command file *runStatsScript.cmd*. In the next line we generate the command to explicitly disconnect from the database. The resulting command file will look something like this:

```
db2 runstats on table BPEUSER.ACTIVITY_INSTANCE_B_T with distribution and detailed
indexes all
db2 runstats on table BPEUSER.AUDIT_LOG_T with distribution and detailed indexes all
...
db2 connect reset
```

Running the generated command file will update the statistics for the listed tables. Note, that the command file does not reorganize the database tables. You may want to extend the command file to perform the REORG command before calling RUNSTATS. Please refer to the DB2 documentation to learn on how to reorganize your database tables using REORG.

4.2.6 Enable Re-Optimization of JDBC Queries

If BPC API queries (as used by the BPC Explorer e.g.) are used regularly on your system, it is recommended to allow the database to re-optimize SQL queries once, as described at [QueryReopt]. This tuning step greatly improves the response times of BPC API queries.

4.3 Tuning Messaging

4.3.1 Messaging Basics

WebSphere comes with a built-in message queuing system called WebSphere Default Messaging. The largest organizational unit in WebSphere Default Messaging is the service integration bus (or just “bus”) which is defined at cell level in a WebSphere Application Server environment. A bus supports applications using message-based and service-oriented architectures. A bus is a group of interconnected servers and clusters that have been added as members of the bus. Applications connect to a bus using one of the messaging engines associated with its bus members.

The following figure illustrates the relationships between the key concepts. Applications exchange messages by connecting to destinations on the bus and not individual messaging engines/queue points as this would defeat the idea of a bus which is to make the location of a service transparent. The localized counterparts of destinations are the queue points in each messaging engine of each bus member.

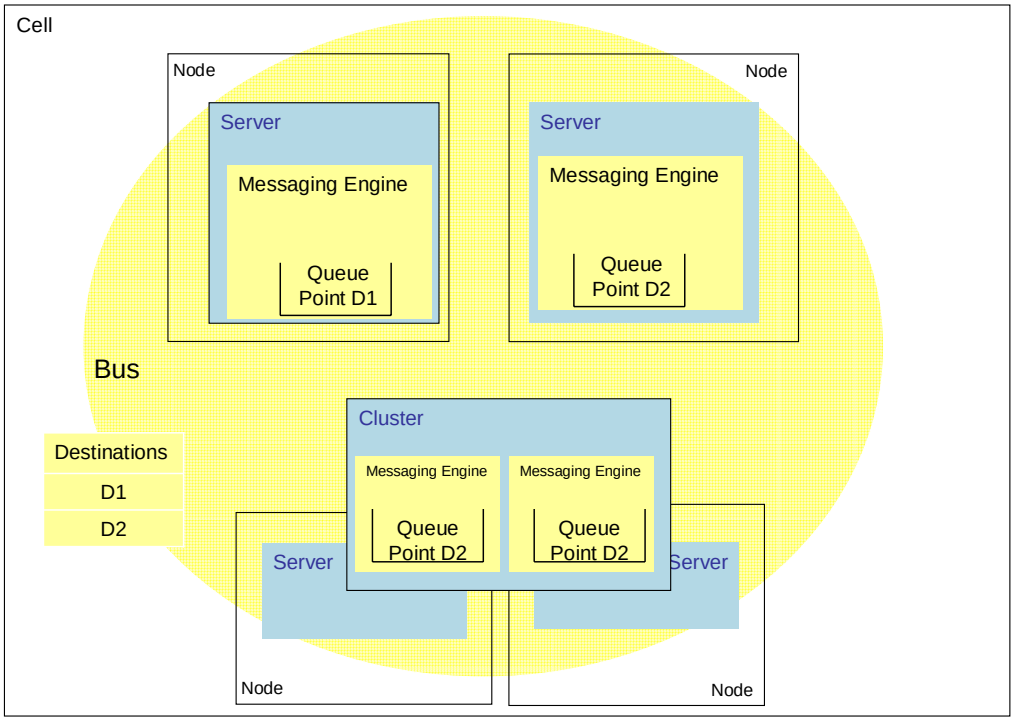


Figure 1: Messaging Topologies

For more details, about these concepts refer to “Chapter 10. Asynchronous messaging” in [WASRedbook].

4.3.2 Tuning messaging databases

The data store for a messaging engine in WebSphere Default Messaging is a database relevant for the overall performance of the system and thus must be included when tuning Business Process Choreographer. The default data stores for all messaging engines are configured during profile creation time to use a Derby database. Lab analysis with long-running business processes suggests that by changing the configuration to use DB2 instead of Derby persistent messaging performance can be improved by a factor of 2 to 3. Therefore the first step necessary is to create data stores for all messaging engines on a high-performance DBMS like DB2. Please refer to the WebSphere Process Server Information Center at [ME] for a description on messaging engines and data stores.

Setting up the data stores for the messaging engines

See [SetupME] for how to setup a data store for a messaging engine.

There are four relevant Buses in the system:

- CommonEventInfrastructure_Bus
- SCA.APPLICATION.<cell>.Bus
- SCA.SYSTEM.<cell>.Bus
- BPC.<cell>.Bus

Some buses contain the name of the cell (<cell> in the list above). Each of theses buses has a Messaging Engine connected, as can be seen in the table below. The <node> symbol represents the respective node name.

Create the database SIB and load the schemas.

Instead of having a DB2 database per messaging engine we put all messaging engines into the same database using different schemas to separate them.

Schema	Messaging Engine
CEI01	<node>.server1-CommonEventInfrastructure_Bus
SCAAPP	<node>.server1-SCA.APPLICATION.<cell>.Bus
SCASYS	<node>.server1-SCA.SYSTEM.<cell>.Bus
BPC	<node>.server1-BPC.<cell>.Bus

Figure 2: Message engines

To accomplish that we have to create database scripts. We create one DDL script for each messaging engine using customized versions of the following example command. The command has to be run from the *bin* directory of your WebSphere installation.

```
sibDDLGenerator.bat -system db2 -version 8.1 -platform windows -statementend ; -schema
<schema> -user <user> > createSIBSchema_<schema>.ddl
```

To customize the example command replace <user> with the user ID that will create the schema *(e.g..dbadmin)*. Create one variant of the command for each of the databases, replacing <schema> with the schema name of the corresponding database, as specified in the table above (CEI01, SCAAPP, SCASYS, BPC).

To be able to distribute the database across several disks, edit the generated DDL scripts. Put each table in a table space named after the schema used i.e. SCAAPP becomes SCAAPP_TB, CEI01 becomes CEI01_TB and so on.

Create the table spaces using the same techniques used for the Business Process Choreographer database, and distribute them across available disks as fitting. If possible place the database log on a separate drive.

The example below shows how to create the table spaces and how to move the transaction log to a separate drive.

```
DB2 CREATE DATABASE SIB1 USING CODESET UTF-8 TERRITORY en-us
DB2 CONNECT TO SIB1

DB2 CREATE TABLESPACE CEI01_TB MANAGED BY SYSTEM USING('G:\tspaces\SIB1\CEI01_TB' )
DB2 CREATE TABLESPACE SCASYS_TB MANAGED BY SYSTEM USING('G:\tspaces\SIB1\SCASYS_TB' )
DB2 CREATE TABLESPACE SCAAPP_TB MANAGED BY SYSTEM USING('G:\tspaces\SIB1\SCAAPP_TB' )
DB2 CREATE TABLESPACE BPC_TB MANAGED BY SYSTEM USING('G:\tspaces\SIB1\BPC_TB' )

DB2 UPDATE DATABASE CONFIGURATION USING NEWLOGPATH 'F:\DB2Log\SIB1' DEFERRED
```

Now the actual database schemas can be created using the DDL scripts generated by *sibDDLGenerator.bat* before.

Create the data sources for the messaging engines

Next WebSphere has to be configured to get access to the messaging database. To accomplish this the WebSphere Administrative Console is used to create a data source for each messaging engine and to configure each engine to use the new datastore.

The following table shows the default state:

data source	default jdbc provider
<node>.server1-CommonEventInfrastructure_Bus	Cloudscape JDBC Provider
<node>.server1-SCA.<node>.Bus	Cloudscape JDBC Provider
<node>.server1-SCA.SYSTEM.Bus	Cloudscape JDBC Provider
<node>.server1-ProcessChoreographer.Bus	Cloudscape JDBC Provider

Figure 3: Default data sources using Cloudscape

To allow using the DB2 databases create a new JDBC provider “DB2 Universal JDBC Driver Provider” for non-XA data sources. Next create four new data sources as single-phase commit data sources, one for each message engine. See also [ConfigJDBC].

Provide new names:

data source	JNDI name	type
CEI01_sib	jdbc/sib/CEI_Messaging	DB2
SCAAPP_sib	jdbc/sib/SCAAPP	DB2
SCASYSTEM_sib	jdbc/sib/SCASYSTEM	DB2
BPC_sib	jdbc/sib/BPC	DB2

Figure 4: New data sources using DB2

Now use the Administrative Console to change the data stores of the messaging engines.

In the Navigation Panel go to

Service Integration ->Buses

and exchange the data stores by entering the JNDI name of the appropriate DB2 data source and set the appropriate schema for each Bus/Messaging Engine displayed.

4.4 Tuning WebSphere Process Server

The business process engine in WebSphere Process Server (WPS) processes a long-running business process using a number of chained transactions. There are 2 types of process navigation techniques in WPS 6.1: **JMS based navigation** and **WorkManager based navigation**. Both types of navigation provide the same quality of service. The default is JMS based navigation.

WorkManager based navigation is available since WPS 6.1[2], and has been introduced to improve the navigation performance of the BPEL engine. Standard processing of chained transactions is implemented using (J2EE-) user-managed transactions and persisting both, business process state and navigation messages in the BPC database. In addition, an object cache is used to avoid re-reading the same object from the database, which reduces the communication with the database and thus improves performance. In exceptional cases such as transaction rollbacks, or in overflow situations, when more messages arrive than the process engine can process at a given point in time, JMS based navigation is used. WorkManager based navigation has to be explicitly turned on by setting the appropriate custom properties on the BPC container. As tests in our performance labs indicate, the performance of business processes which make heavy use of the internal BPC navigation can be improved by up to a factor of 2 when using WorkManager based navigation.

JMS based navigation has been used since support for long-running business processes has been introduced in BPC. JMS based navigation uses a combination of persistent message queues to store the business process navigation messages, as well as state in the BPC database to reliably store business process and human task instance data.

In either case, using a transactional model for both the navigation through a business process and the execution of the individual parts of a process makes the execution of a process reliable. Business process integrity is ensured in case of failure. The transactional nature of the underlying mechanisms makes sure that state is always preserved or at least a last-known-as-good-state is available and recovery is possible. The same is true for the Service Component Architecture (SCA), where all asynchronous communication is carried out using a message queuing system.

The figure below depicts a JMS invocation of a service. It can be seen that a number of queues are involved and a number of messages flow through the various queues before the result of the service invocation is returned to the calling process.

[2] WorkManager based navigation can be configured since WPS 6.1.0.1. In cluster installations, apply i-Fix IZ21335

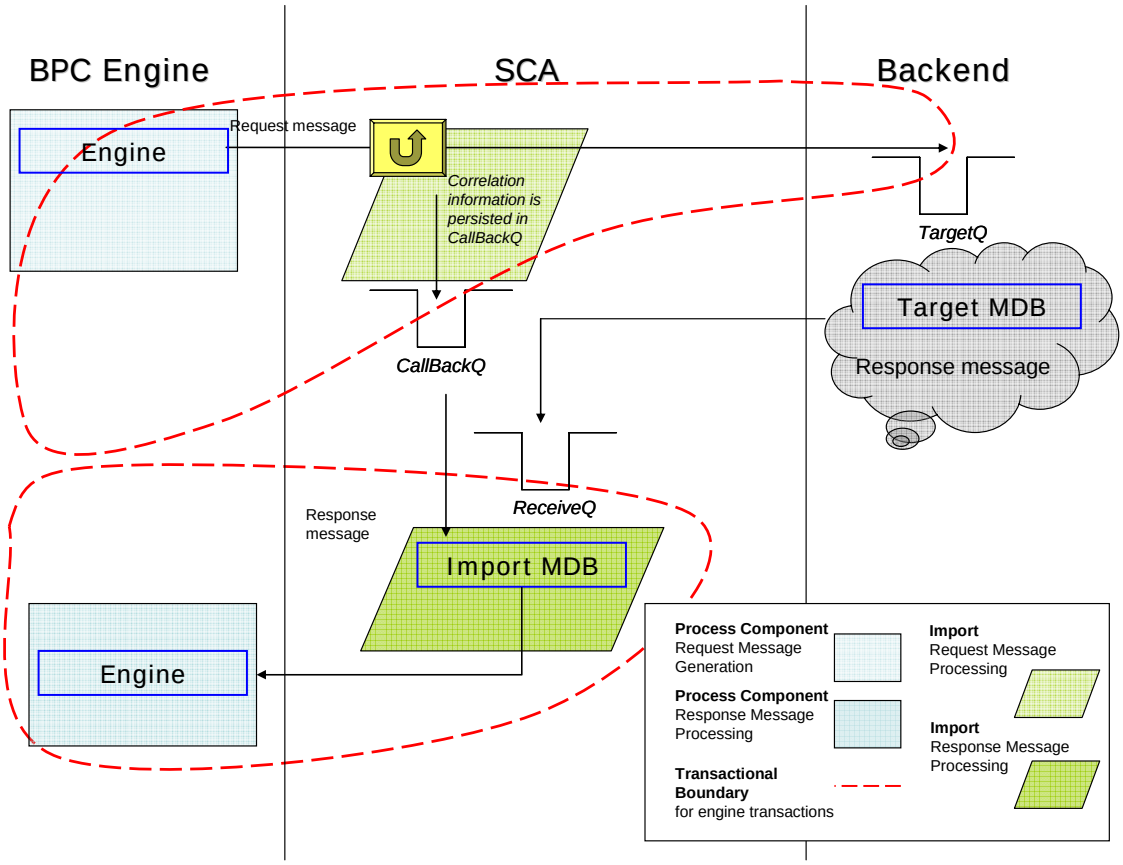


Figure 5: JMS service invocation with BPEL and SCA

It is important to understand the ramifications of a queue based system where messages flow from queue to queue and possibly pile up in one queue while another queue is empty. Generally it is undesirable to have one queue acting as a bottleneck where many messages pile up as this leads to increasingly long response times for the invocation or navigational steps involved. From the outside it seems as if process instances wait for a long time before they advance one more step. For the responsiveness of a process instance it is therefore advisable to make sure that the processing capacity in terms of throughput of the system is large enough to process incoming requests even at peak times in a reasonable time. The business process engine behaves like any queuing system in this respect. Overload causes response times to increase.

If JMS based navigation is used, there is no scheme (for example age-based or priority-based) to process older or more highly prioritized business process instances first. This makes it very hard to predict the actual duration of an instance, especially on an overloaded system. It is the responsibility of the person who is tuning the system to make sure to assign system resources appropriately to make sure that process instances are created at a rate that keeps the system load at acceptable levels.

If WorkManager based navigation is used, process instances which are currently processed are being further processed as long as there is outstanding work for these process instances. While this is very efficient, it prefers running process instances, and forces others to wait. In order to balance the assignment of execution resources (worker threads) between running process instances, a parameter exists that determines the maximum time a business process can use a worker thread exclusively.

Adjusting parameters starts best with:

- the activation specs of the affected MDBs in case of JMS based navigation
- the WorkManager threads in case of WorkManager based navigation

From there you need to work up the chain of dependant parameters. The following picture displays the dependency graph of the relevant tuning parameters:

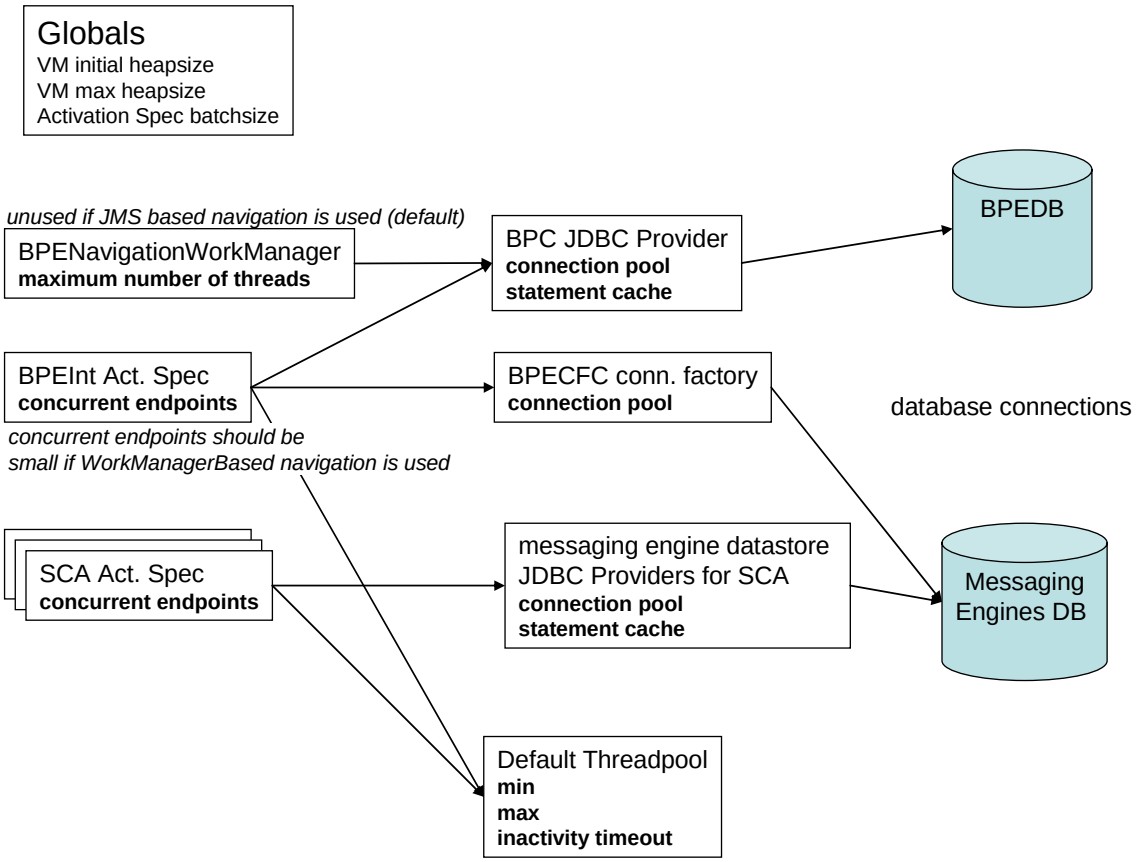


Figure 6: Resource dependencies for BPEL applications

The database symbols on the far right represent JDBC provider properties on the WebSphere Process Server (WPS) side, and connections allowed for databases on the DBMS side. The relationship is an asymmetric one: If the DBMS provides less connections than expected on the WPS side, then the WPS server's performance suffers badly, whereas an imbalance in the other direction does not usually cause large negative effects.

4.4.1 Understanding and determining BPEL engine parallelism

Knowing the maximum number of BPEL transactions running in parallel is key in order to determine the size of database connection pools and other resources' pools.

BPEL transactions are executed in the following situations:

- the BPEL engine (MDB) picks up a navigation message from the BPE internal queue (BPEIntQueue) – the number of these is limited by the maximum concurrency in the BPEInternalActivationSpec
- the BPENavigationWorkManager processes a navigation message

- an API request which e.g. creates a process instance or sends an event to an existing process instance. Those requests can enter WPS over HTTP (servlets or web services) or RMI/IIOP (remote EJB invocation) or JMS.
- an SCA MDB receives an asynchronous request or response for a long-running process instance

In practice it can be hard to precisely determine the maximum number of concurrent BPEL transactions. However:

- the minimum number of parallel BPEL transactions is:
  - the concurrent endpoints on the BPEInternalActivationSpec if JMS based navigation is used
  - the maximum number of threads on the BPENavigationWorkManager plus the concurrent endpoints on the BPEInternalActivationSpec if WorkManager based navigation is used
- the maximum number of parallel BPEL transactions is the sum of all thread pools available in WebSphere Process Server. This is basically:
  - Web container thread pool size +
  - ORB thread pool size +
  - Default thread pool size +
  - All WorkManager's thread pool sizes

A good knowledge of the applications that interact with Process Choreographer is necessary in order to determine the maximum number of concurrent BPEL transactions.

Determining the parallelism of BPEL transactions needed for process instance navigation: The optimum number of concurrent BPEL transactions working on process instance navigation (so that the system reaches maximum performance) is dependent on the services that are invoked by the process templates, and on the process templates themselves. As a rule of thumb: The longer the transactions of the engine instance, the more concurrent transactions are needed to completely utilize the system.

The duration of BPEL transactions is dependent on:

- The response time of the services invoked synchronously (e.g. by 'invoke' activities)
- The transactional boundaries that result from the process model, e.g. the "participate" flag setting for the transactional behavior of an invoke activity reduces the number of transactions and therefore lengthens the engine transactions.

To tune your system, start with two concurrent BPEL transactions per processor. Increase the concurrency if the system cannot be fully utilized

4.4.2 Tuning WorkManager based navigation related resources and parameters

Note: It should not be necessary to tune JMS based navigation related resources as described in the next chapter if WorkManager based navigation is configured, unless the JMS API is heavily used.

How it works

The picture below shows the navigation of a single process instance, using WorkManager based navigation (see [WorkManager] for more information about this technology):

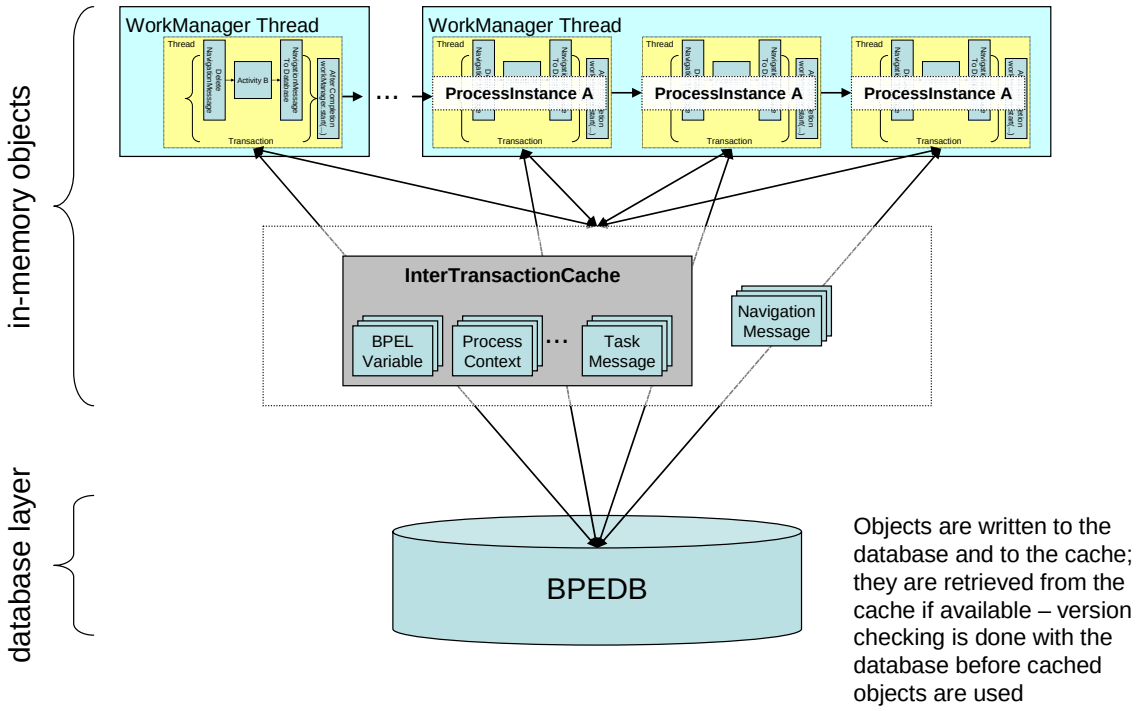


Figure 7: WorkManager based navigation: Cache and thread pool

Each BPEL transaction is executed within a J2EE transaction. Sub-sequent transactions on the same process instance are executed within the same WorkManager thread, which avoids switching the application context and saves processing time. In case of a fork within the same process instance (e.g. a BPEL flow activity), the work is distributed across threads of the WorkManager . As a consequence, there is no workload distribution across servers within a cluster during normal navigation of a single process instance. However, if asynchronous services are invoked, or if human tasks are involved, a business process may continue in another server. Also, if the system is overloaded for a long time, JMS based navigation will complement WorkManager based navigation – see parameter *workManagerNavigation. maxAgeForStalledMessages* and *workManagerNavigation.messagePoolSize* for details.
An object cache called InterTransactionCache is used to cache BPEL objects between transactions. The InterTransactionCache holds a configurable maximum number of objects. Additionally, it is cleared by the JVM's garbage collector if the JVM is running out of heap space.

How to configure

The following custom properties on the Business Process Choreographer Container are used to configure WorkManager based navigation. They are configured on the Business Process Choreographer Container with default settings.

Parameter / Object	**Recommended initial values**
allowPerformanceOptimizations	yes
interTransactionCache.size	1000

workManagerNavigation.maxAgeForStalledMessages	240
workManagerNavigation.maxProcessTimeOnThread	240
workManagerNavigation.messagePoolSize	1000
workManagerNavigation.recoveryIntervalForStalledMessages	2minutes
BPENavigationWorkManager (Work request queue size)	100
BPENavigationWorkManager (Minimum number of threads)	10
BPENavigationWorkManager (Maximum number of threads)	20

Set **allowPerformanceOptimizations** to *yes* in order to activate WorkManager based navigation after server or cluster restart. The default value is *no*. The following settings are only effective if allowPerformanceOptimizations has been set to *yes*.

interTransactionCache.size defines how many objects are cached between transactions. Don't set this value too high, as it might negatively impact the usage of the Java heap.

workManagerNavigation.maxAgeForStalledMessages defines the duration in seconds until a navigation message which is stored in the BPEDB database is sent to the BPE internal queue (BPEIntQueue) and processed with JMS based navigation. Unless a server crash occurs, or a server is overloaded for at least 240 seconds (as of the setting above), this will not happen.

workManagerNavigation.maxProcessTimeOnThread defines the duration in seconds until the navigation of a particular process instance must return a WorkManager thread back to the pool. As explained before, process instance navigation is optimized by using the same thread to process multiple transactions in a business process instance. However, this behavior could prevent other process instances from navigation in some circumstances, e.g. in case of big while-loops. Decrease this value if process instance navigation should be more evenly distributed across multiple process instances.

workManagerNavigation.messagePoolSize defines the size of the cache for navigation messages. Navigation messages are stored in the database in order to provide the same quality of service as JMS based navigation. If the number of messages in the cache exceeds the cache size, messages are written directly to the BPEIntQueue for processing by the MDB of Process Choreographer. This may happen under high load, where intra-process instance parallelism is very high, such as when the BPEL parallel foreach activity is used.

workManagerNavigation.recoveryIntervalForStalledMessages is the time frame between two checks on the system whether stalled messages (see workManagerNavigation. maxAgeForStalledMessages) exist. Each stalled message is sent to the BPEIntQueue and then processed using the JMS based navigation.. Note that this may happen if the system is highly overloaded for a long period of time, or if a JVM crashed.

The **BPENavigationWorkManager** provides the threads that are used when WorkManager based navigation is activated. It can be found at Resources→Work managers→BPENavigationWorkManager at server scope (standalone installation) or at cluster scope (cluster installation).

The BPENavigationWorkManager (Work request queue size) is the number of BPEL transaction execution requests (navigation messages) that are cached by the BPENavigationWorkManager. Set this parameter to twice the maximum number of threads of the BPENavigationManager. If this parameter is too low, navigation messages are sent to the BPEIntQueue under circumstances, although enough resources for processing the navigation message locally are available.

The **minimum number of threads of the BPENavigationWorkManager** should be set to the average number of BPEL transactions that are processed in parallel on a single server. If set too low, then incoming parallel requests may result in situations where additional threads have to be created, which may negatively impact performance.

The **maximum number of threads of the BPENavigationWorkManager** determines the parallelism that is used to process BPEL navigation messages. If the server's CPU is not fully utilized although enough work is available, increase this value. Note that shortcomings in I/O sub-systems (serving databases or the WPS transaction log, for instance) can be compensated by a higher parallelism to a certain degree.

4.4.3 Tuning JMS based navigation related resources

If JMS based navigation is used (default), navigation between transactions is implemented using JMS messages that carry the necessary information:

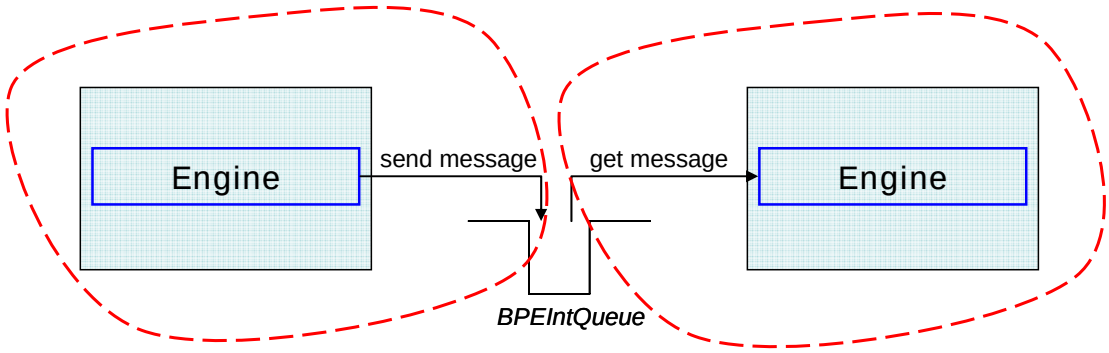


Figure 8: JMS based BPEL navigation

Tuning WebSphere Process Server for optimal operation of the process engine means adjusting the parameters of queues, activation specs for the Message Driven Beans (MDBs) listening on the queues and providing the necessary auxiliary resources (connections on the JDBC providers, thread pools) needed by the MDBs to operate.

As mentioned above the business process engine triggers the execution of long-running processes using internal navigation messages once the process is started. Therefore the number of concurrent engine threads ("engine instances") must be configured using an Activation Spec and a Queue Connection Factory. The Activation Spec *BPEInternalActivationSpec* is used for retrieving navigational messages from Business Process Choreographer's navigation queue. The connection factory *BPECFC* is used to put navigational messages into the same queue. Both resources must be configured to provide the desired degree of concurrency.

For *BPEInternalActivationSpec* the performance relevant parameters are:

- maximum concurrent endpoints
- maximum batch size

The number of concurrent endpoints determines the number of engine instances concurrently processing navigational messages on the BPE internal queue (BPEIntQueue), which is approximately the number of concurrently executed activities.
The batch size causes the messaging system to pre-allocate a 'batch' of messages, which reduces the work involved. Measurements have shown that for interruptible processes at high server utilization a value of more than '8' does not provide better performance. A value of '8' is therefore recommended.

For the *BPECFC* connection factory the performance relevant parameters is the connection pool size. It determines the number of connections used by BPEL transactions. In JMS based navigation, each BPEL transaction requires a connection to the *BPECFC* (so that they do not have to wait for a free BPECFC connection).
Be generous in specifying the size of the BPECFC connection pool; parallelism of BPEL transactions should never be restricted by connections to the underlying messaging layer. As described in chapter 4.4.1, calculate the maximum parallelism of BPEL transactions. This should be the value for the maximum size of the BPECFC connection pool. You can also use the Tivoli Performance Viewer which is part of the WebSphere Application Server administrative console to monitor the number of connections used during operation.

To ensure that enough connections to the messaging systems available, refer to "Tuning the JDBC providers" later in this document.

4.4.4 Tuning the SCA queues

Persistent, transactional messages sent over SCA queues are stored in a database. WebSphere Default Messaging optimizes performance for small messages (for DB2 as storage, optimization takes place up to a message size of 3 KBytes). If messages are of medium size (3 KBytes to 20 KBytes), e.g. an asynchronous service invocation that passes a big business object, and DB2 is used as the data store underneath the SCA queues then refer to [TuneSIBus] in the Resources section of this document for further performance optimization potential. Note that all internal messages of the business process engine are less than 3 KBytes.

SCA uses several queues to perform asynchronous communication. There is one queue per SCA module. In the WebSphere Administrative Console the activation specs for module queues can be found under

Resources>Resource adapters>Platform Messaging Component SPI Resource Adapter.

As with the BPEInternalActivationSpec, concurrency of message processing is managed by the maximum number of endpoints. The *batchsize* parameter should also be set to a value of eight ('8').

If JMS invocations are used, there are three more queues involved (destination, reply, callback) which need two activation specs. The queues have to be created manually as they are not generated. The activation spec for the destination queue needs to be created in the Administrative Console under

Resources>JMS Providers>Default Messaging.

The activation spec for the reply queue is created during deployment in the same location; the callback queue does not need an activation spec.

4.4.5 Tuning the JDBC providers

Business Process Choreographer database

As described earlier in this document, the business process engine stores different kinds of data in a database that is by default named *BPEDB*. Each engine instance requires a connection to the BPEDB database.
The maximum number of connections to BPEDB should not be less than the maximum number of parallel running BPEL transactions, as described in chapter 4.4.1. This is configured using the connection pool size of the BPEDB data source.

Also consider that each call to the BPE API such as retrieving task lists requires a connection to the BPEDB. Increase the size of the BPEDB connection pool according to the expected number of concurrent API requests.

Another performance relevant parameter is the *Prepared Statement Cache* size. Set this parameter to a value higher than 100 (default: 128). Note, that each database connection and each cached statement requires about 1 KByte of memory. Large database connection pools and large prepared statement cache sizes may require that you enlarge the heap size of your JVM.

Messaging databases

The data source *ProcessChoreographer_sib* acts as data store for the business process engine's internal messaging.

If JMS based navigation is used, messages into the BPE internal queue (BPEIntQueue) are either put by the process engine during process navigation, or by the Business Process Choreographer API. Therefore, the size of the connection pool for *ProcessChoreographer_sib* must be the maximum concurrent engine instances..

If WorkManager based navigation is used, messages are written to the internal message queue in exceptional cases only. It should not be necessary to increase the size of the connection pool for *ProcessChoreographer_sib* then.

If the JMS API is used, set the size of of the connection pool for *ProcessChoreographer_sib* to the maximum number of concurrent API calls sending JMS requests.

SCA activation specifications *(SCA activation specs)* are used for asynchronous communication, as described earlier. They have the same dependency on having enough JDBC connections available, as the *BPEInternalActivationSpec*.

4.4.6 Thread Pools

Threads for *SCA activation specs* and the *BPEInternalActivationSpec* are taken from the *Default Thread Pool*. ORB client requests also use threads from the Default Thread Pool. Therefore it is important to ensure that the size of the Default Thread Pool is big enough to satisfy all components. The size of the Default Thread Pool can be modified using the WebSphere Administrative Console. It can be found under

Application Servers->servername->Thread Pools.

4.4.7 Messaging Caches

There are two types of buffers which are used by a messaging engine to cache messages and message-related data: discardable data buffers and cached data buffers. By defining these caches, performance may especially improve in scenarios with large messaging activity.

To define the size of these caches you have set custom properties on messaging engine level (Service Integration > Buses > *bus_name* > Messaging engines > *messaging_engine_name* > Custom properties) for all four messaging engines with the key sib.msgstore.discardableDataBufferSize or sib.msgstore.cachedDataBufferSize respectively. The values of these properties represent the size of the cache in bytes.

Please refer to the WAS InformationCenter for more information on how these caches work [TuneSIBus]

5 Best Practices for Web clients

This section provides information on how to tune WebSphere Process Server for the usage of Web Clients. The first section focuses on the usage of BPC Explorer whereas the second section gives you client tuning tips which apply to all other web clients as well.

5.1 BPC Explorer Specific Tuning Settings

5.1.1 Make use of customized views

BPC Explorer provides the ability to define customized views which filter desired items by a given criteria. If you frequently work with a specific set of items, you should make use of this and define your own views. This will lead to a reduced result set and hence improve memory consumption and response times.
For example, if you frequently have to work with a list of all claimed human tasks, you should rather define a view which filters these items than scrolling through the default “Tasks administered by me” view which shows tasks in all states.
To define a customized view, log on to BPC Explorer with a user in the role *BPESystemAdministrator* and click “Define Views” in the top panel.

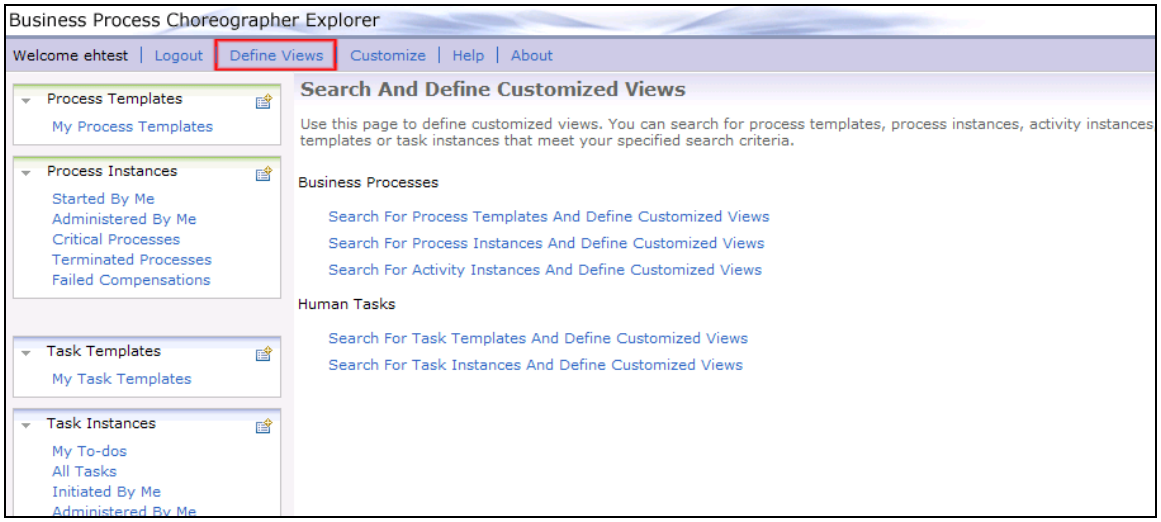


Figure 9: customized views in BPC Explorer

It is highly recommended to define a threshold for these customized views to reduce the number of items in the result set and prevent unnecessary storage of objects in the heap. You can do this by switching to the “View Properties” tab when defining a customized view. The threshold is entered in the field “Maximum Results”.

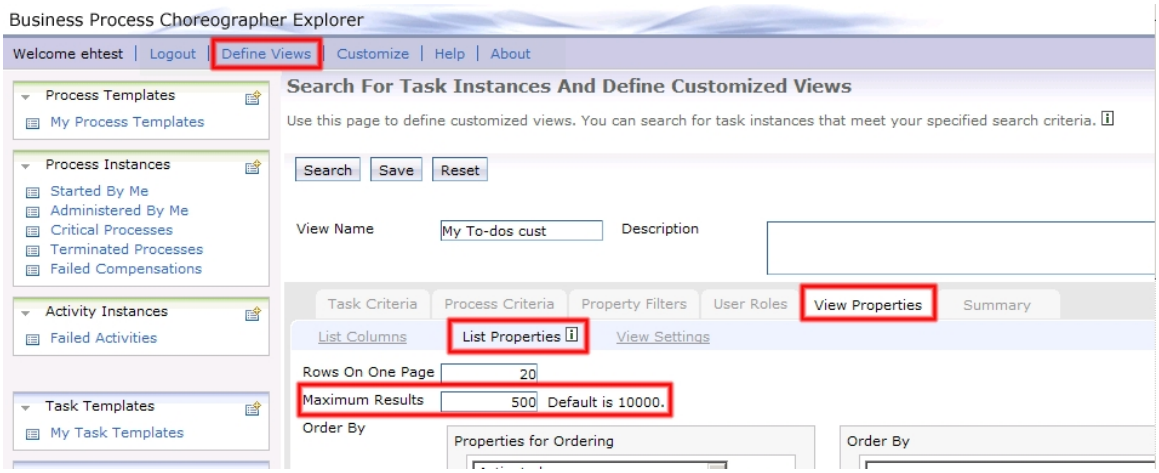


Figure 10: Setting a threshold for customized views

5.1.2 Set an appropriate login view

By default, BPC Explorer shows the “My To-dos” view after you logged in. If you don’t work frequently with this list of task instances, you may consider choosing a different log-in view that better reflects your major use case . If the login view contains only a small number of items then this is beneficial for performance when starting BPC Explorer. You can specify the login view by logging on to BPC Explorer with a user in the role BPESystemAdministrator. Click “Customize” in the top panel and select the desired login view from the drop-down box in the section “Select the login view”.

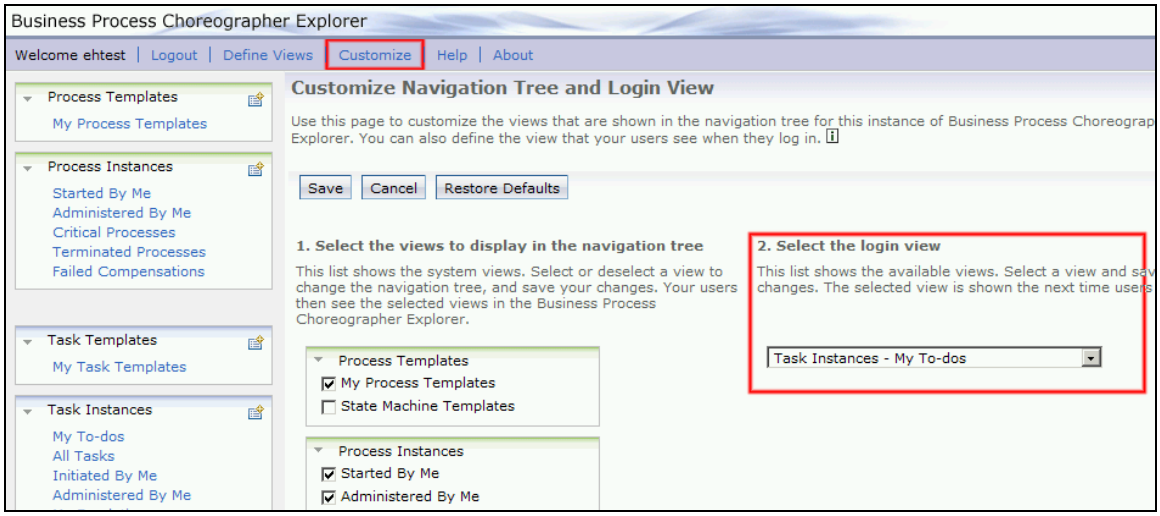


Figure 11: Customizing BPC Explorer’s login view

5.1.3 Define restrictive thresholds for default views

When setting up BPC Explorer you are asked to define a default threshold for all query results. However, in order to decrease memory consumption you may consider defining more restrictive thresholds on a fine-grained level for the default views of Explorer (e.g. “My To-dos”).
To achieve this, you have to replace the pre-defined views by new customized views (see section “Make use of customized views” in this chapter). Set up these views with the same search criteria

as in the pre-defined views but set the number of maximum results to a lower value than the default threshold.

5.2 Common Client Tuning Tips

5.2.1 Consider increasing your maximum heap size

Each client which connects to the web container naturally increases the server's memory consumption. The more clients that are connected to your server, the more objects have to be kept in memory. By increasing the maximum heap size of the server which owns the web container, more clients can be put on the system and also, response times for the clients may improve. Please note that since WPS 6.1 using large amounts of heap memory does not have the same negative effects on performance as in previous versions. The reason is that the JVM shipped with 6.1 provides very efficient garbage collection mechanisms, which means that the time required for garbage collection grows only insignificantly with a larger heap size.

5.2.2 Specify Web Container settings

By adjusting specific web container settings in WebSphere's Administrative Console, you can improve response times especially in high load scenarios. The following settings should be considered to be adjusted according to the expected load. There are no standard settings which improve performance for all scenarios so you should monitor the relevant resources (e.g. using Tivoli® Performance Viewer) while running a representative workload and then decide if and how to change them.

- **Thread inactivity timeout**
  (Servers > Application Servers > *server_name* > Thread Pools > WebContainer)
  The thread inactivity timeout specifies how much time (in ms) must elapse before a thread can be reclaimed. The default setting is 3500 ms. .

- **Minimum and maximum Web Container thread pool size**
  (Servers > Application Servers > *server_name* > Thread Pools > WebContainer)
  Once there have been more threads in the pool than specified by the minimum pool size setting, the pool size won't drop below this value. The maximum pool size specifies the upper limit for web container threads in the pool. If your CPU is not fully utilized and you want to increase concurrency, setting a higher value might improve user response times.
  Note that the maximum number of threads doesn't specify the maximum number of clients which can connect to the system. Usually, the number of maximum clients connected to the system is considerably higher since the threads can be shared between several client connections.
  To determine whether changing the container size might result in a performance gain, you can use Tivoli® Performance Viewer to monitor the load on the threads (PercentMaxed counter) and the number of active threads (ActiveThreads counter) for the Web container module. If the value of the PercentMaxed counter is consistently in double digits, then the Web container might be a bottleneck. In this case, increase the number of threads. If the number of active threads is lower than the number of threads in the pool, decreasing the thread pool size might result in a performance gain.

- **Session timeout**
  (Servers > Application Servers > *server_name* > Web container settings > Web container

> Session management)
The default session timeout is set to 30 minutes. By lowering this value, you can reduce memory consumption since inactive user sessions and their related objects are kept in the memory as long as the users don't log out or the session timeout occurs. However setting this value too low means that users often have to log on again, which results in a bad user experience and has some performance overhead.

6 Conclusion

We have described in this paper what the best practices are when tuning automatic business processes, both microflows and long-running processes. Also, Web client specific best practices – especially for the BPC Explorer – have been covered.

Another important aspect that has been left out of the picture in this paper is performance tuning Human Workflows and Human Tasks. When involving humans into business processes then a major new category of requests comes into play which the infrastructure has to handle: People querying for work available for them. Due to the nature of automatic business processes, queries did not play a role in this paper. Queries are a significant factor for Human Workflow though. Their use resulted in a set of additional performance tuning guidelines and best practices that have been documented in the white papers *Performance Tuning of Human Workflows Using Materialized Views* [HumanWorkflowPerformance] and *BPC Queries using query() and queryAll()* [BPC Queries].

As a final comment please allow us to point out that while performance tuning is a good thing to do, a well-designed application may perform well out of the box, without sophisticated performance tuning applied. To be able to create well-designed applications it is required to understand the concepts of the technology (see [Concepts]), to be familiar with the programming model (see [ProgMod]), and to know a few of the tricks about how to write efficient applications (see [EfficientBPEL]). Following the modeling best practices while creating the application and then tune it later on, after deploying it into the production runtime is going to provide the best performance.

7 Resources

Please note that the references below are in the order of their occurrence in the paper, not ordered by importance.

[BPMPerf]
WebSphere Business Process Management V6.1 Performance Tuning Red Book
http://www.redbooks.ibm.com/abstracts/redp4431.html

[Concepts]
WebSphere Process Server V6 – Business Process Choreographer: Concepts and Architecture
http://www.ibm.com/support/docview.wss?uid=swg27008938

[TuneJVM]
WebSphere Application Server InfoCenter chapter on Java Virtual Machine tuning
http://publib.boulder.ibm.com/infocenter/wasinfo/v6r1/topic/com.ibm.websphere.base.doc/info/aes/ae/tprf_tunejvm_v61.html

[DiagnoseJVM]
Diagnosing the Java Virtual Machine
http://www-128.ibm.com/developerworks/java/jdk/diagnosis/index.html

[TuneWAS]
WebSphere Application Server InfoCenter chapter on Application Server Tuning
http://publib.boulder.ibm.com/infocenter/wasinfo/v6r1/index.jsp?topic=/com.ibm.websphere.base.doc/info/aes/ae/welc6toptuning.html

[TuneStriping]
Tuning Logical Volume Striping on AIX
http://publib.boulder.ibm.com/infocenter/pseries/v5r3/index.jsp?topic=/com.ibm.aix.prftungd/doc/prftungd/tuning_log_vol_striping.htm

[QueryReopt]
Improving the performance of complex BPC API queries on DB2
http://www-1.ibm.com/support/docview.wss?rs=2307&uid=swg21299450

[WASRedbook]
ITSO, WebSphere Application Server V6: System Management and Configuration Handbook, SG24-6451-00 February 2005
http://www.ibm.com/redbooks

[ME]
WebSphere Application Server InfoCenter chapter on Messaging Engines
http://publib.boulder.ibm.com/infocenter/wasinfo/v6r1/index.jsp?topic=/com.ibm.websphere.pmc.nd.doc/tasks/tjk9999_.html

[SetupME]
WebSphere Application Server InfoCenter chapter on setting up the data store for a Messaging Engine
http://publib.boulder.ibm.com/infocenter/wasinfo/v6r1/index.jsp?topic=/com.ibm.websphere.pmc.nd.doc/tasks/tjm0030_.html

[ConfigJDBC]
WebSphere Application Server InfoCenter chapter on configuring a JDBC data source
http://publib.boulder.ibm.com/infocenter/wasinfo/v6r1/index.jsp?topic=/com.ibm.websphere.pmc.nd.doc/tasks/tjm0040_.html

[WorkManager]
WebSphere extensions
http://publib.boulder.ibm.com/infocenter/wasinfo/v6r1/index.jsp?topic=/com.ibm.websphere.base.doc/info/aes/asyncbns/concepts/casb_asbover.html

[TuneSIBus]
Medium size message SIBus tuning
http://publib.boulder.ibm.com/infocenter/wasinfo/v6r1/index.jsp?topic=/com.ibm.websphere.pmc.nd.doc\concepts\cjm0470_.html

[EfficientBPEL]
Modeling Efficient BPEL Processes by A. Robeller and G. Wilmsmann, September 2004
http://www-1.ibm.com/support/docview.wss?rs=1079&context=SS2KS6&uid=swg27005564&loc=en_US&cs=utf-8&lang=en

[ProgMod]
WebSphere Process Server V6 – Business Process Choreographer Programming Model
http://www.ibm.com/support/docview.wss?uid=swg27007157

[HumanWorkflowPerformance]
Performance Tuning of Human Workflows Using Materialized Views
http://www.ibm.com/support/docview.wss?uid=swg27009623

[BPC Queries]
BPC Queries using query() and queryAll()
http://www.ibm.com/support/docview.wss?uid=swg27010849

8 Appendix A – Tuning a small production system

8.1 Overview

This section demonstrates the tuning of a small server setup consisting of two 4-way machines, one for running the application server with WebSphere Process Server installed, the other for running the DBMS load.
Two benchmarks are being used, a microflow and a long-running process, each with service invocations using two different bindings to cover distinctive areas of business process execution.
The microflow is using the synchronous SCA binding to invoke

- Java components in the same module as the process (POJO)
- a Web Service in a different module (WS)

The long-running process uses

- an asynchronous SCA binding (JMS)
- a synchronous SCA binding to invoke a Web Service (WS).

The exercise starts with explaining the benchmarks used and describes the available machines. The setup of the database is described with respect to the distribution across the available disks and initial parameter settings.
The first round of benchmarks is run on this **un-tuned setup**. The results for microflow and long-running process runs are presented.
The system is then tuned for **JMS-based navigation** and the second round of benchmarking is conducted. The results show that the database server is not equipped with enough disk I/O capabilities to support an application server running on a 4-way machine.
Additional disks are added to the database machine and the database layout is adjusted.
The JMS-based navigation runs are repeated and the results are presented.
Finally **WorkManager-based navigation** is enabled on the application server. The long-running benchmarks are run and the results are presented.

The tuning improves microflow throughput between 27% (POJO) and 31% (WS).
The throughput for long-running processes improves by 73% (JMS) and 90% (WS) for JMS-based navigation and by 125% for synchronous SCA bindings and WorkManager-based navigation.

8.2 Description of used benchmarks and limitations

This Tuning Exercise takes up the adjustments suggested in this paper and applies them to a small server setup. Two benchmarks are used to measure system performance before and after tuning to demonstrate the effectiveness of the tuning applied.
The benchmarks consist of a microflow and of a long-running process bound to services using various bindings to cover tuning for these bindings. The services are dummy services, they simply return when called to keep the service overhead as low as possible.

8.2.1 Microflow benchmark

The microflow benchmark is a small process containing four invoke activities. Two variants of the microflow are executed:

- using synchronous SCA binding to invoke Java components (POJOs) in the same module

- using synchronous SCA Web Service binding to invoke a Web Service in another module

Microflows run within one transaction and thus cannot easily call services with an asynchronous binding like JMS.
The service bindings covered here represent the fastest synchronous invocation type, which is synchronous SCA Binding (fastest) and SCA Web Service Binding (slowest)..
The tuning aspects covered comprise

- tuning of the Java VM
- tuning of the Web container for Web Services operations

8.2.2 Long-running process benchmark

The long-running benchmark contains 24 activities overall, with 10 service invocations. Multiple paths through the benchmark are possible, but only a single path is executed for the benchmark. This path executes 11 service invocations by executing one of the invocations twice. .
Two variants of the long-running benchmark are used, they differ only in the bindings used for invoke activities:

- JMS Binding – to cover messaging tuning and asynchronous invocation style
- Web Service Binding – to cover Web Services tuning for long-running processes and synchronous invocation style

The transaction boundaries of the two variants also differ as asynchronous service invocations introduce transaction boundaries as a technical necessity. After placing the outgoing message the asynchronous service invoke commits the current transaction. When the callback arrives a new transaction is started. So for asynchronous service invokes a transaction boundary is introduced which cannot be avoided. By setting the 'Transactional Behaviour' to 'participates' an additional transaction boundary near the service invoke can be avoided which is usually beneficial for the performance of a process.
Synchronous service invocations do not introduce transaction boundaries. For the Web Services Binding variant, 'commit after' is set on service invocations to model transaction boundaries after each service invoke explicitly. This encapsulates each service invocation in a separate transaction and allows the rollback of the process to the last transaction boundary in case a service invocation fails.
As a result, the synchronous benchmark variant is executed in 25 transactions, and the asynchronous benchmark variant is executed in 15 transactions.
The difference between the two variants will also become important when WorkManager-based navigation is demonstrated.

8.2.3 Limitations

These benchmarks do not cover all areas of the product. Among the features not covered are:

- Human tasks
- Task list queries or queries for business process data

8.3 Description of machine setup and software

The benchmarks are run on the following hardware

- 3 IBM x3650, 3 GHz dual core Intel Xeon (Woodcrest), 16 GB memory, IBM ServeRAID 8i, 8 x 72 GB disks, (dual core - dual socket, small server systems)
- connected by Gigabit Ethernet switches.
- all machines run Windows Server 2003 Enterprise

- DBMS used is IBM DB2 V9.1 FP3a
- WebSphere Process Server 6.1.2 used to host the benchmarks and the Web Services.

The setup separates WPS and the database to best mirror real customer scenarios, and to allow to separately tune the performance of the database and the performance of the process engine.

8.3.1 Description of the database setup

The database setup is simplified compared to what is described in the database section of this paper. The basic disk layout consists of

- RAID-0 or 1 with 4 physical disks, stripe size 256 kB for all table spaces
- RAID-0 or 1 with 3 physical disks, stripe size 64 kB for DB2 logs

The messaging engine datastores are contained in the same database that also stores business process related data, instead of a separate one.
The database is created by issuing

DB2 CREATE DATABASE BPEDB USING CODESET UTF-8 TERRITORY en-us

Database Layout: Folder <WPS root>/dbscripts/ProcessChoreographer/DB2 contains two SQL scripts which must be executed in order to prepare the database objects. Note that these scripts must be customized before being used.
The tablespaces are created by issuing

DB22 –tf *createTablespace.sql*

The tablespace containers were placed onto the same RAID-volume as defined above. The schema for the process database is created by issuing

DB2 -tf *createSchema.sql*

The database transaction log is placed on the second RAID-volume.

It is important to set the lock timeout from infinity to a value like 30 seconds to avoid having the database waiting for locks indefinitely. This is set using

DB2 UPDATE DATABASE CONFIGURATION USING LOCKTIMEOUT 30

For version 9 of DB2 most critical parameters are self-adjusting and should assume proper values after some benchmark runs. This is true for bufferpool sizes, locklist, number of connections and the various heap areas. If automatic maintenance is enabled, statistics update should run and adjust the database statistics.

One of the most critical data points during a benchmark is the **bufferpool hit ratio**. In DB2 V9 the bufferpool hit ratio can be monitored with a SQL statement against the view `SYSIBMADM.BP_HITRATIO`. Enable bufferpool monitoring by issuing

DB2 UPDATE MONITOR SWITCHES USING BUFFERPOOL ON

from a DB2 command prompt.

The SELECT-statement

```
SELECT SUBSTR(DB_NAME,1,8) AS DB_NAME, SUBSTR(BP_NAME,1,14) AS BP_NAME,
   TOTAL_HIT_RATIO_PERCENT, DATA_HIT_RATIO_PERCENT,
   INDEX_HIT_RATIO_PERCENT, XDA_HIT_RATIO_PERCENT, DBPARTITIONNUM
   FROM SYSIBMADM.BP_HITRATIO ORDER BY DBPARTITIONNUM
```

delivers the current bufferpool hit ratios in the following form

```
DB_NAME  BP_NAME        TOTAL_HIT_RAT… DATA_HIT_RAT… INDEX_HIT_RATIO_PERCENT …
-------- -------------- -------------- ------------- ----------------------
BPETUNE  IBMDEFAULTBP            99.57         99.66                  95.21 …
BPETUNE  IBMSYSTEMBP4K               -             -                      - …
BPETUNE  IBMSYSTEMBP8K               -             -                      - …
BPETUNE  IBMSYSTEMBP16K              -             -                      - …
BPETUNE  IBMSYSTEMBP32K              -             -                      - …

  5 record(s) selected.
```

Hit ratios greater than 90% are considered optimal.

The database is pre-loaded with 11,000 process instances in state FINISHED to simulate a small production load and to allow for reliable statistics generation.
Note: Statistics update is performed only after the un-tuned runs to show the performance difference.

8.4 Monitoring the system during benchmarking

Benchmarking needs close monitoring of the machines and the software involved for two reasons:

1. to assess the performance of the system during the tuning iterations
2. to identify bottlenecks/problems during the benchmark runs

For this exercise the Windows Performance Console has been used to monitor the following data points:

- CPU utilization of application server and database server
- network data rates of application server and database server (external network and loopback)
- application server and database server disks – for each disk records of *idle time*, *bytes per second* and *transfers per second* were created

With these data points the critical condition of the disk subsystem was discovered in this exercise and the appropriate counter-measures could be taken.

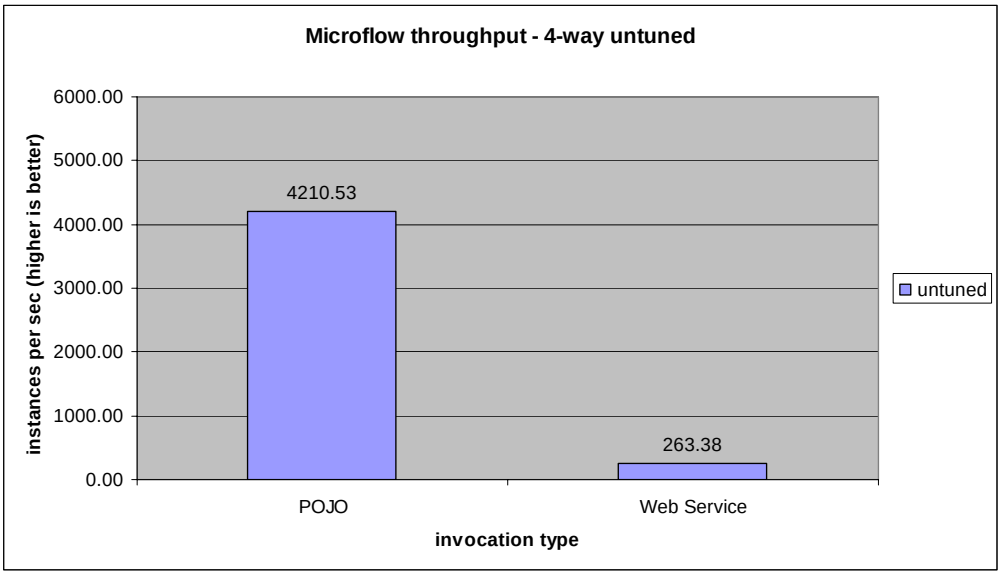
The actual throughput numbers were acquired from the benchmark harness.

8.5 Running benchmarks on un-tuned system

After installing the benchmarks on a new WPS 6.1.2 installation, and running each benchmark with the servers and database default settings, the following results have been obtained. Routinely, data for CPU, disk and network loads is gathered. The highlights are presented along with the actual raw performance data.

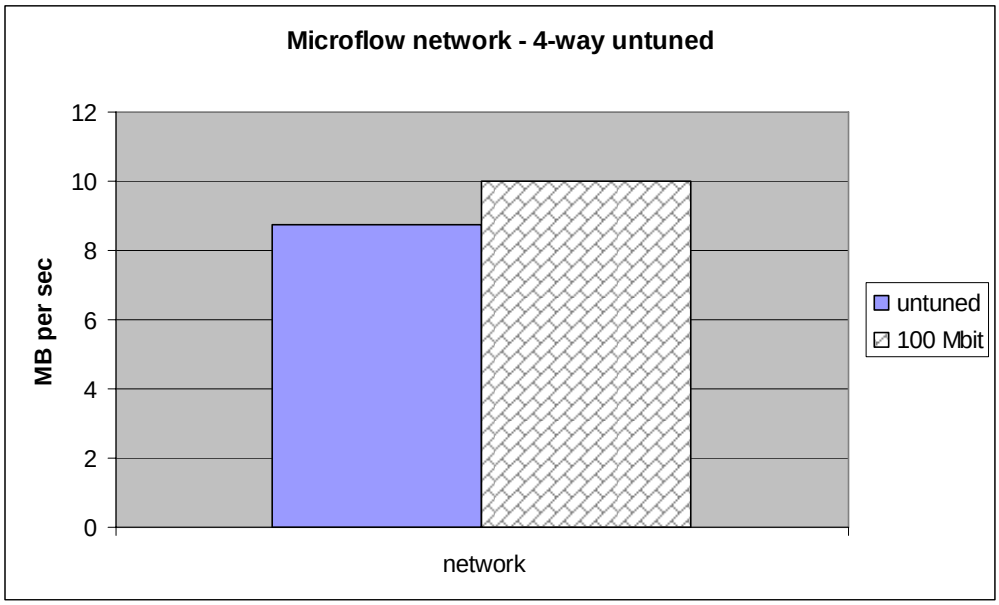
8.5.1 Microflow un-tuned

Microflow results for POJO and Web Service invocation on 4-way:



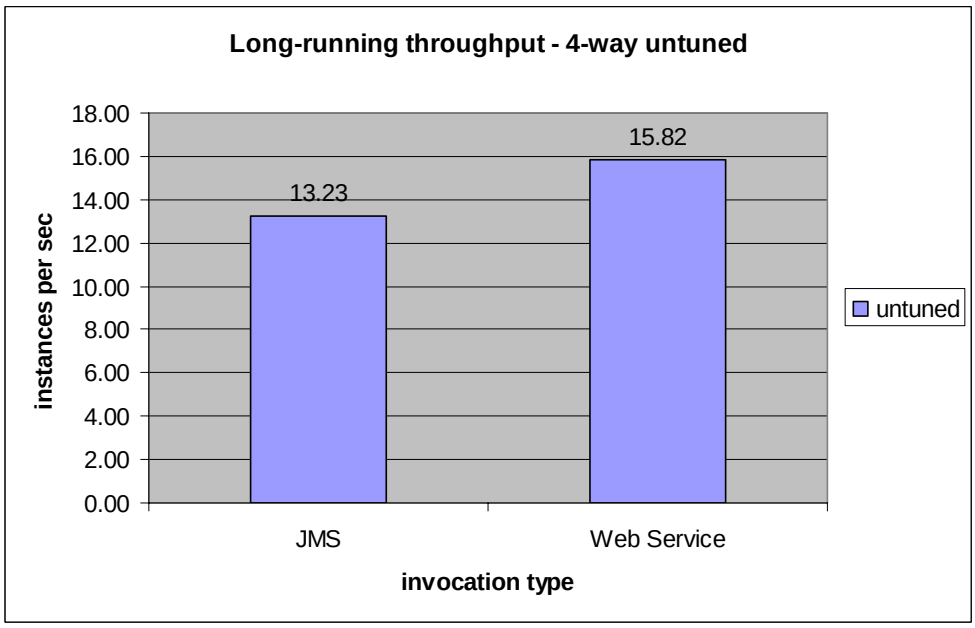
The performance difference between using a synchronous SCA Binding compared to a Web Services Binding is almost a factor of 16. This means that the costs for business process navigation are small compared to the overhead involved in a service invocation using Web Services Binding (over HTTP).

Looking at the network load for the Web Service reveals that the system is already running close to the achievable user data rate of a 100 Mbit LAN (taking away the protocol overhead for TCP/IP leaves roughly 10 MB/s). Note that this is not an issue for this setup as it uses Gigabit Ethernet, but it highlights the importance of high-speed network connectivity.

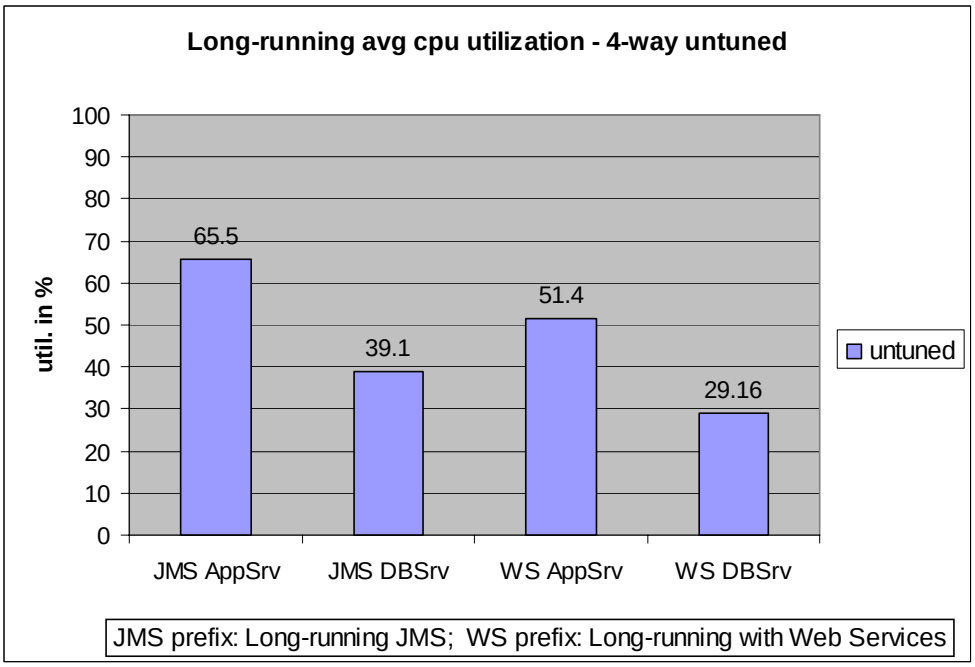


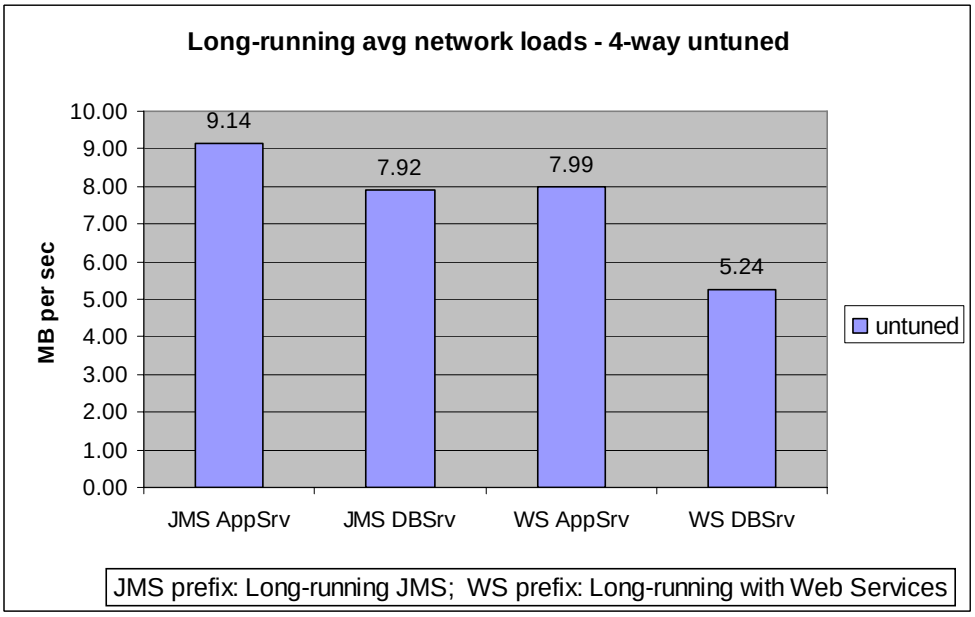
There is no load on the disks or on the network to the database server.

8.5.2 Long-running un-tuned



Asynchronous bindings are more costly than synchronous bindings due to the overhead that comes with message queuing. The JMS binding shows lower throughput than the synchronous Web Service Binding.
Comparing CPU loads, it is obvious that neither the application server machine nor the database server machine is fully utilized.





The network load for JMS is higher despite lower overall throughput. The network loads on the application server are higher than the network loads on the database server because the application server has to handle additional traffic for the Enterprise Service Bus in the JMS case and the Web Service communication for the Web Service case.

Note: The network load for JMS again approaches the limits for 100Mbit LAN.

8.6 Tuning the server for JMS-based navigation

Following the recommendations given in the main part of this paper, the following settings have been made to optimize the setup for achieving maximum throughput.

8.6.1 JVM Settings

Initial heap size	1280
Maximum heap size	1280
Generic JVM arguments	-Xgcpolicy:gencon –Xmn640M
Verbose garbage collection	enabled

threadpool	min size	max size	inactivity timeout (ms)
Default	20	80	default
SIBFAPThreadPool	20	100	default
WebContainer	50	50	1

8.6.2 Messaging Settings

On all messaging engines, the buffers for messages have been increased from the 320 kB (which is the default) to 10 MB by adding the following custom properties on each messaging engine:

custom property	value
sib.msgstore.discardableDataBufferSize	10000000
sib.msgstore.cachedDataBufferSize	10000000

The JMS connection factory ***BPECFC*** for BPC has been tuned by providing a larger connection pool with **minimum** size 50 and **maximum** size 70.

Application-specific customer-created **JMS connection factories** for JMS queues used by customer application should be treated similarly to allow for parallel operation.

The **activation specs** for the application module queues, application import queues, and the custom JMS queue for Banking JMS have been set to

- Maximum batch size: 8
- Maximum concurrent endpoints: 25

The value for the maximum batch size setting was determined in a separate experiment where the throughput was measured for batch sizes from 1 to 32 and no improvements in throughput were seen for values larger than 8.
The number of concurrent endpoints depends on the hardware and latencies in the setup. A higher degree of parallelism can mask latencies introduced by the network and disk subsystems and improve utilization and throughput. Best practice is to start with a value of 10 and increase to a point where further improvements are negligible.
The same values for all activation specs are sufficient here as the benchmark is sequential in nature and all queues are approximately loaded to the same degree. Other applications might make it necessary to size the number of concurrent endpoints differently for each activation spec in order to balance the load over all queues.

The **BPEIntQueue** (internal navigation queue) was set to use

- Maximum batch size: 8
- Maximum concurrent endpoints: 30

Setting a higher value for the concurrent endpoints helps with the Web Service invocation benchmark to mask latencies and to obtain a higher throughput.
In general, one starts with a lower number of concurrent endpoints (e.g. 10), and increases the number until the system reaches the point of maximum throughput.

8.6.3 JDBC Settings

The messaging engines are using datastores backed by DB2. Each messaging engine uses a separate JDBC datasource.
The process database is also hosted by DB2 and the process engine uses an XA datasource for connecting to the process database.
The datasources need to provide enough connections to satisfy the settings for concurrent endpoints on the activation specs. The settings used for the benchmarking have been the following:

datasource	min connections	max connections	statement cache size
BPEDatasourceDB (process database)	50	70	300
BPC_sib	50	50	40
CEI_sib	5	10	40
SCAApp_sib	50	50	40
SCASys_sib	20	40	40

Note, that CEI is not used for the benchmarks so the CEI messaging engine was not tuned.
The *SCA System Bus* is mainly used for exception handling and therefore also does not need a large number of parallel connections.

8.6.4 Database

The default bufferpool for BPEDB was manually set to 100,000 4k pages to avoid the automatic adjustment to kick in and distort benchmark runs. The process database gets tuned by updating detailed statistics with the script generated as explained in the database tuning chapter. After preloading the database with 2000 instances the script was run to update the statistics.

After creating the new statistics the dynamic statement cache has to be cleared to force the database to re-compile the incoming dynamic SQL statements using the updated statistics. This is accomplished by issueing

```
DB2 flush package cache dynamic
```

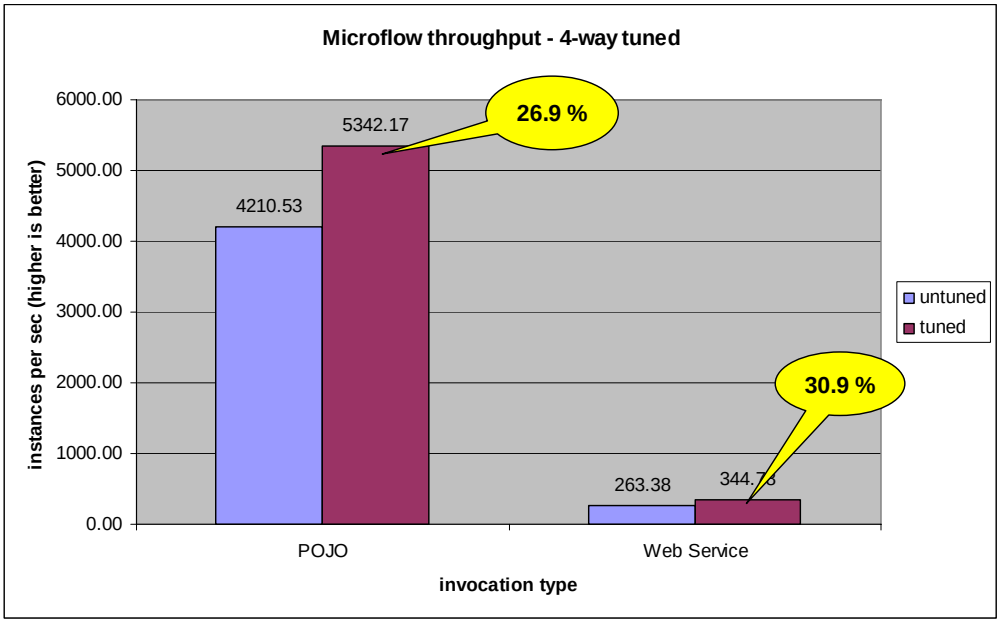
The database was running with the parameter for the number of connections (MAXAGENTS) set to AUTOMATIC, so the required number of connections was available. In case that the number of connections provided by the database is not sufficient, exceptions are thrown on the application server, and the applications are slowed down.

8.7 Running benchmarks on tuned system with JMS-based navigation

Again the microflow results will be presented first along with the un-tuned results for comparison, followed by the results for the long-running benchmarks.

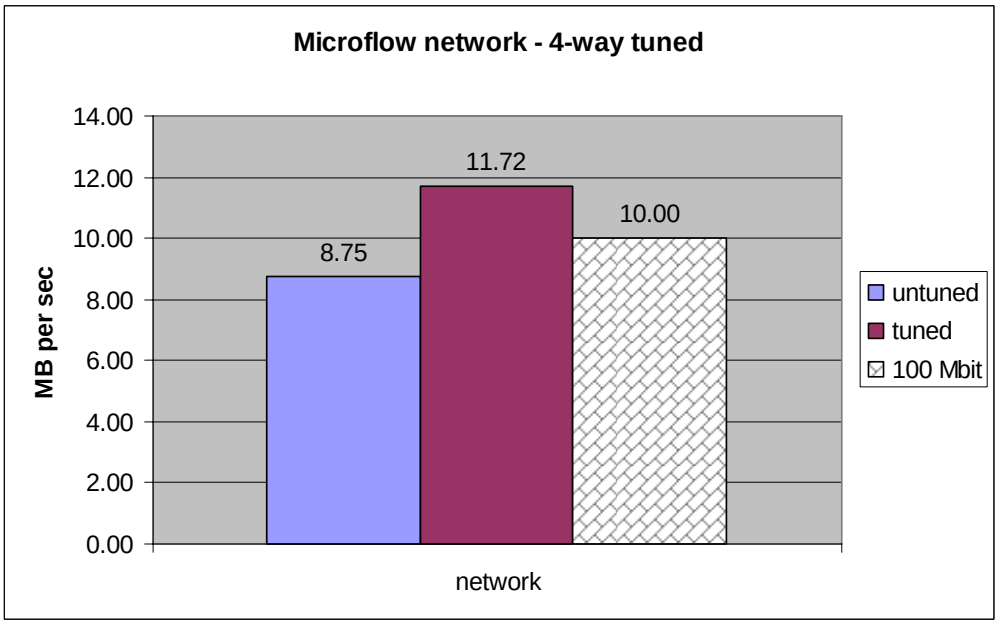
8.7.1 Microflow on system tuned for messaging

After tuning, the results for microflow performance increased considerably.



With the tuning for the Java virtual machine, improvements of 27% (POJO, synchronous SCA binding) to 31% (SCA Web Services binding) have been obtained.

The network load now exceeds the capabilities of a 100 Mbit Ethernet for the SCA Web Service microflow variant.



8.7.2 Long-running processes on system tuned for messaging

Initial tests of the tuned system showed lower throughput for 4-way operation than for the un-tuned system. In order to analyze where the bottleneck occurred the application server machine was operated 1-way, 2-way and 4-way and tested with the JMS benchmark.
The observations made where the following:

- 4-way operation of the WPS server machine leads to severe disk congestion and low performance (8.6 process instances/sec)
- 2-way operation shows same problem at a similar performance
- 1-way is the good case (best performance of all configurations - up to 10.4 process instances/sec)

A closer analysis was performed using the Windows performance console to monitor the disks. This analysis revealed that once the requested I/O data rate exceeded 2200 requests/sec at approximately 5500 Bytes/request (~ 12 MB/sec), the queue length of the volumes increased to 4, and more importantly the average request response time increased by almost 2 orders of magnitude (from 0.00012 to 0.012 s).
The drives could not recover from this load and slowed down to very low levels of performance. As a consequence the I/O load of the database had to be set up in a more optimal way.

8.7.3 Re-distributing the database load

As a result of the severe I/O performance problem on the database server three SAN drives were attached to the database server using a Fibre Channel connection.
As a second measure the messaging portion was separated into a second database. This separated the transaction logs and lowered the load on the drive hosting the original transaction logs.
The new setup looked as follows:

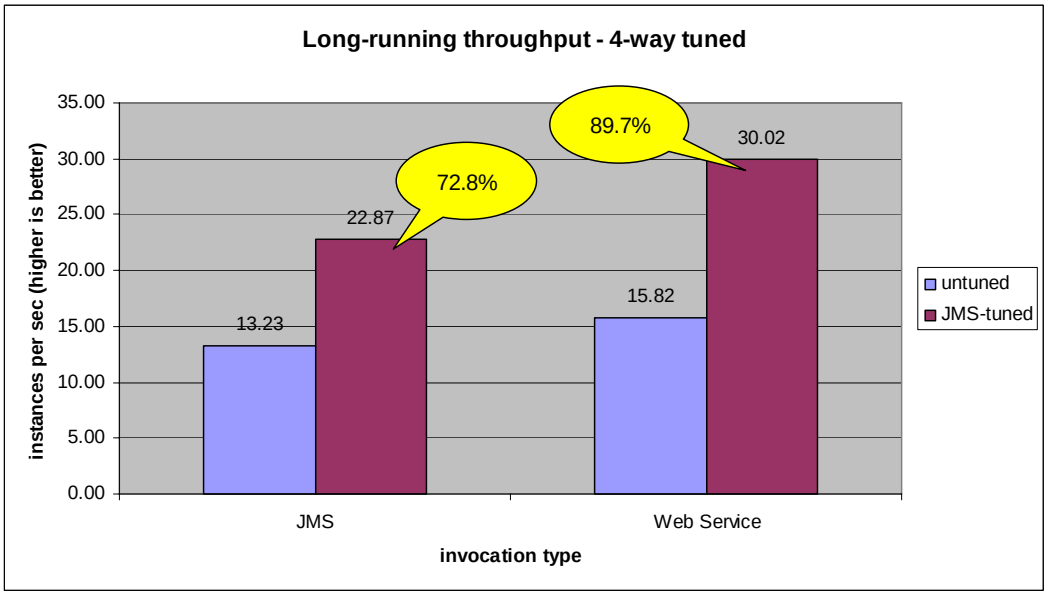
- RAID-0 with 4 physical disks, stripesize 256 kB for messaging tablespaces
- RAID-0 with 3 physical disks, stripesize 64 kB for DB2 log for messaging
- SAN, stripesize 256 kB for process database transaction log
- SAN, stripesize 256 kB for process database INSTANCE tablespace
- SAN, stripesize 256 kB for remaining tablespaces of process database

The SAN used is an IBM DS 3400, with 24 disks, two-channel connection configurable to run with either 2 or 4 Gbit bandwidth, 1 GB cache memory per channel. The SAN was attached using Fibre Channel at 4 Gbit.

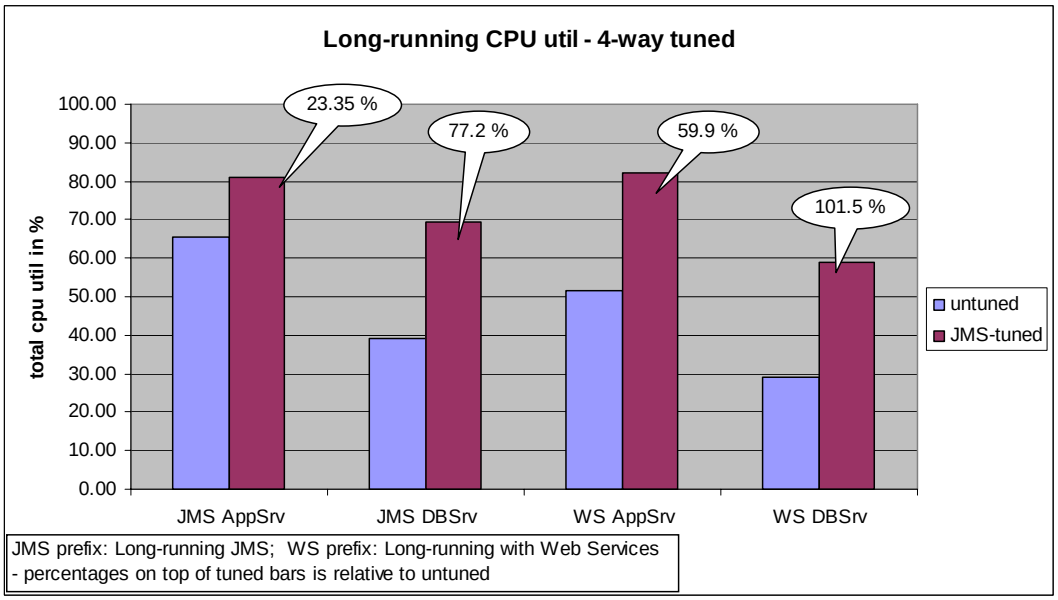
Using the SAN allowed the 4-way operation of the application server without completely overwhelming the I/O subsystem of the database server.

8.7.4 Long-running process results revisited

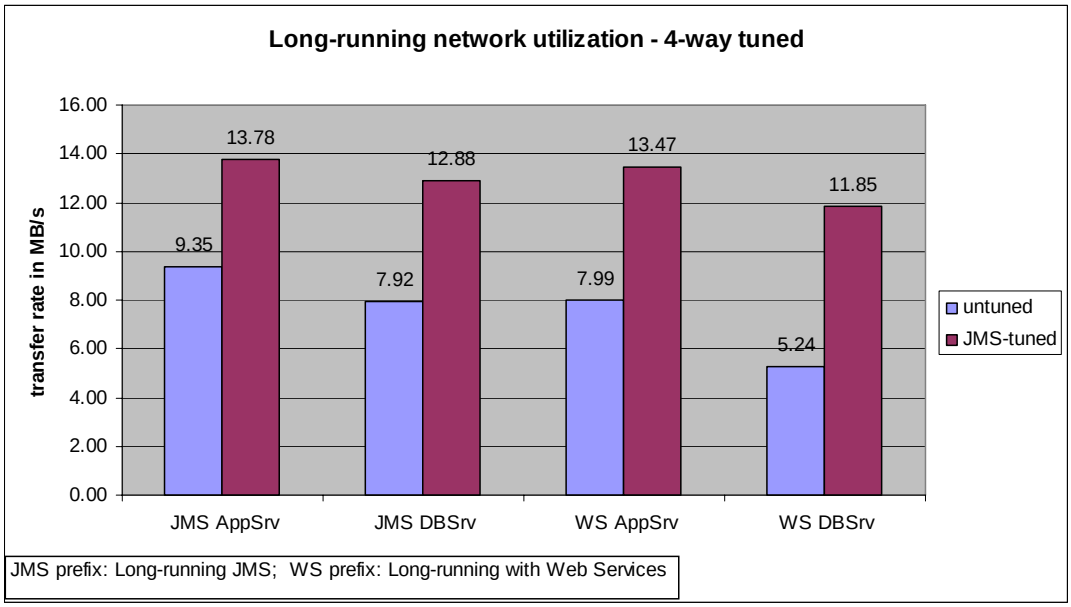
Here are the long-running results for the revised database setup.



The tuning applied to the application server in combination with an optimized database layout results in large improvements between 72% for the long-running process with JMS invocations and almost 90% for the variant with Web Service invocations.



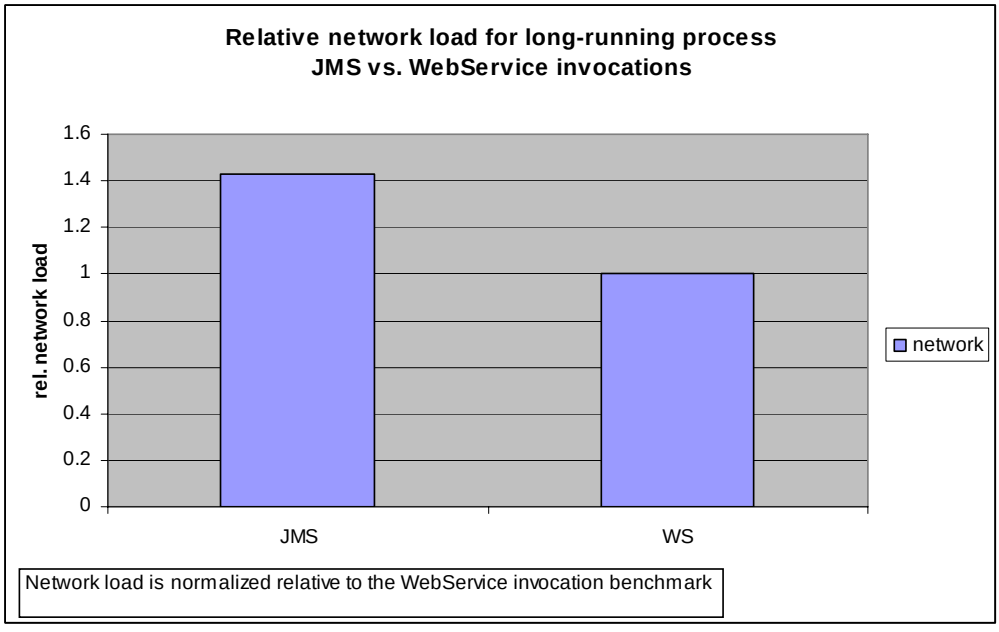
Looking at the differences in CPU utilization, the improvements in throughput roughly match the CPU utilization increases on the database server. On the application server the increase is not as pronounced (for JMS: application server increase 23.35% vs. database server 77.2%) so the tuning seems to allow for more efficient execution on the application server.



The network utilization to the database server roughly increases as the throughput does. The traffic on the application server increases a little less. Note the higher overall traffic for JMS invocations at lower throughput compared to Web Service invocation. By scaling the JMS throughput result to the Web Service result and applying the scaling factor to the network transfer rate for the database server we can estimate the relative network load for JMS compared to Web Service invocations.

scaledNetworkDB(JMS) = (throughput (WS) / throughput(JMS)) * networkDB(JMS)

Normalizing the scaled database network load for JMS using the database network load for Web Services results in the following chart:



This extrapolation shows that JMS invocations cause a roughly 40% higher network transfer rate at the same throughput compared to Web Service invocations.

8.8 Tuning the server for WorkManager-based navigation

Enabling WorkManager-based navigation is described in the main part of this document. The tuning has been performed using the values which are suggested in the respective section.

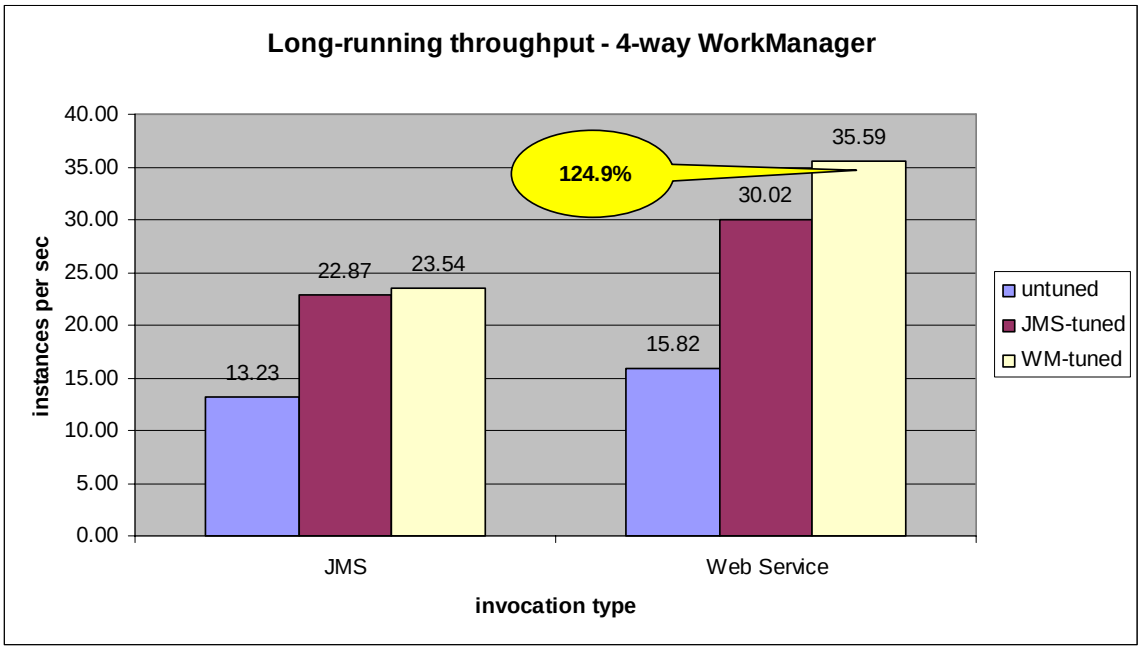
property on Business Flow Manager	**values**
allowPerformanceOptimizations	yes
interTransactionCache.size	1000
workManagerNavigation.maxAgeForStalledMessages	240
workManagerNavigation.maxProcessTimeOnThread	240
workManagerNavigation.messagePoolSize	1000
workManagerNavigation.recoveryIntervalForStalledMessages	2minutes

BPENavigationWorkManager attribute	**value**
Work request queue size	100
Minimum number of threads	10
Maximum number of threads	20

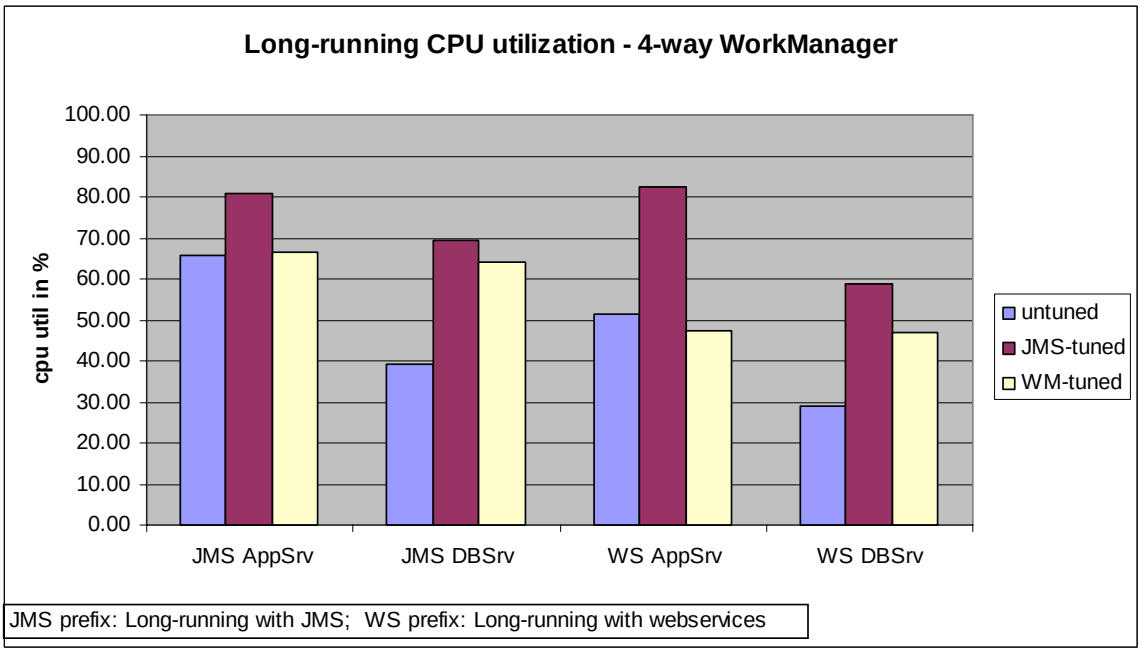
8.9 Running long-running benchmarks with WorkManager-based navigation

For the WorkManager-based navigation tests the microflow benchmarks have been skipped as this switch only affects long-running process navigation; microflow performance is not affected.

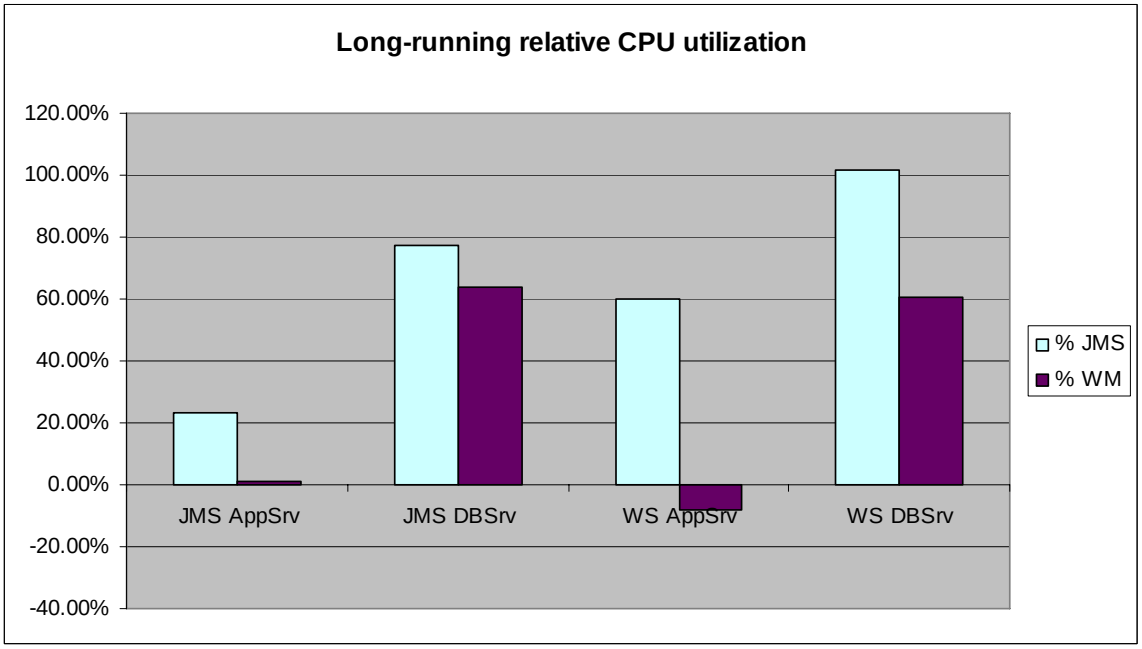
8.9.1 Long-running processes on WorkManager-based navigation



Switching to WorkManager-based navigation increases the throughput for Web Service invocations by another 18% to a total of 125% improvement over the un-tuned system. JMS does not benefit much as WorkManager-based navigation is primarily improving the internal handling of process navigation messages, which are less used when invoking asynchronous services.

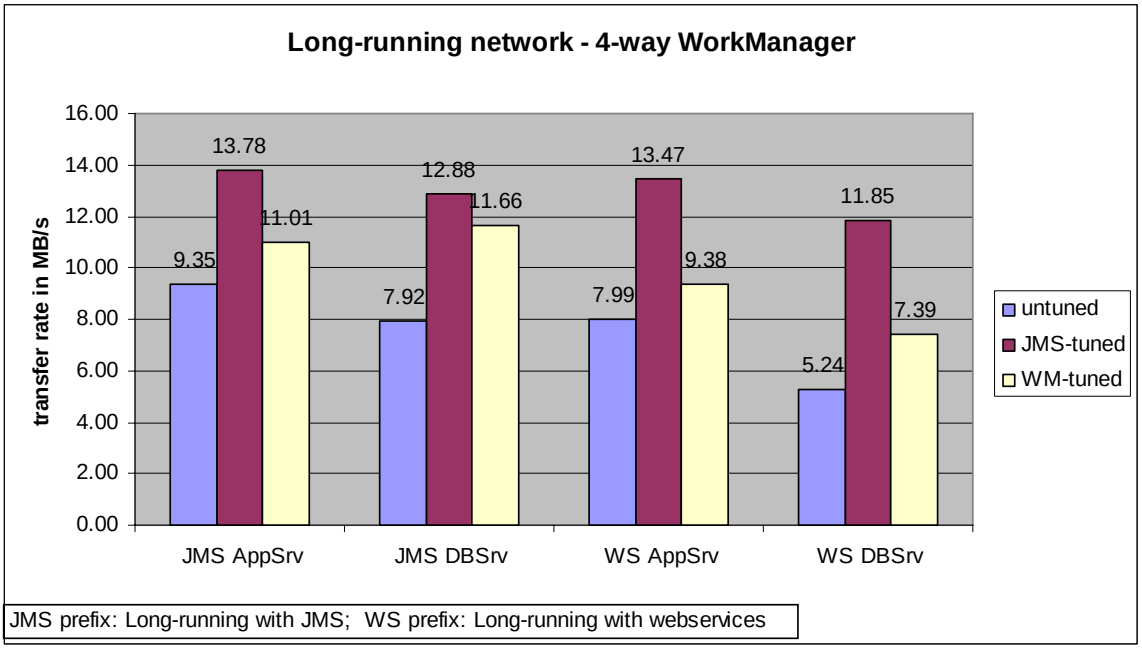


The CPU utilization under WorkManager-based navigation decreases on the application server back to the level of the un-tuned setup despite the same or higher throughput.
The utilization on the database server decreases less but noticeable.
Looking at the percentages of change relative to the un-tuned system this becomes obvious.

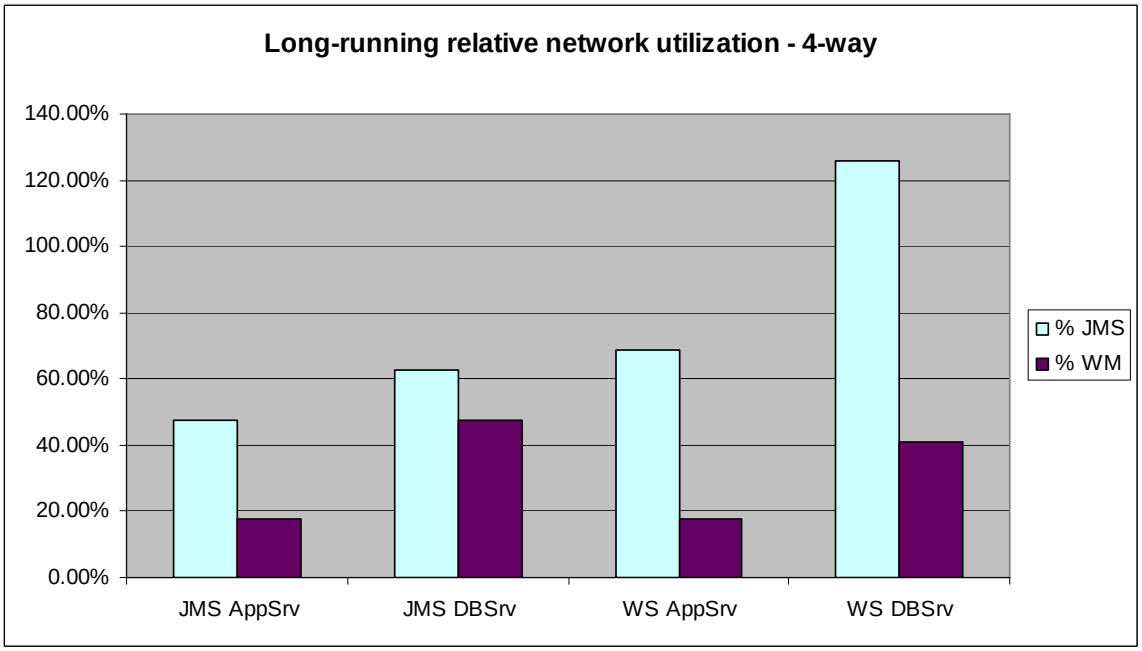


The application server CPU load returns to the level of the un-tuned system.
The database CPU load is diminished, less for JMS, much more for Web Services. This is due to the fact that WorkManager-based navigation includes the *inter-transaction cache* which lowers

the database traffic significantly for synchronous invocations. Asynchronous JMS invocations do not make as much use of this mechanism so the effect is less pronounced.



The network traffic shows a similar picture. The traffic to the database is slightly reduced when running the JMS variant of the benchmark; the traffic to the database is significantly lowered when running the Web Services variant, less data arrives at the database, the CPU load is reduced.



The relative differences chart shows this even more clearly. For WorkManager-based navigation there is 20% more network traffic from the application server over the un-tuned setup and

roughly 40% more traffic to the database consistently for both the JMS and the Web Service variant of the benchmark.
However the throughput increase compared to the un-tuned configuration is 78% for the JMS variant and 125% for the Web Services variant.

8.10 Summary

Tuning can improve the throughput capabilities of a given setup to a large degree as shown in this exercise.

Microflow 4-way (all results in process instances per sec)	**default**	**JMS-tuned**	**WorkManager-tuned**
POJO	4210	5342 (26.9 %)	5342
Web Service	263	344 (30.9 %)	344

microflow results

Long-running 4-way (all results in process instances per sec)	**default**	**JMS-tuned**	**WorkManager-tuned**
JMS	13.2	22.9 (72.8 %)	23.5 (77.9 %)
Web Service	15.8	30.0 (89.7 %)	35.6 (125 %)

long-running process results

Improvements have been as large as 125% which is a considerable factor in terms of hardware expenses and/or performance requirements.
When comparing microflow and long-running process performance with similar invocation styles (for example Web Service invocations) the long-running processes approximately run one order of magnitude slower.

The network data rate to the database is an important factor for long-running processes especially when considering clustered setups where multiple application server nodes work using the same database node. The traffic to and from the database node multiplies with the number of nodes in the application cluster.

Long-running 4-way (all results in MB / sec)	**default**	**JMS-tuned**	**WorkManager-tuned**
JMS	7.9	12.9 (62.7 %)	11.6 (47.3 %)
Web Service	5.2	11.8 (126.1 %)	7.4 (40.1 %)

long-running process network transfer rates

These results show that WorkManager-based navigation largely improves network efficiency when compared to the throughput increase. For the best case traffic to the database increases by 40 % but the throughput increases by 125 % compared to the default system.

8.11 Conclusion

The main purpose of this exercise has been to demonstrate the application of performance tuning to a given system and to present the resulting performance.

Sometimes it can be necessary to change hardware components due to incomplete initial planning or uncertainty about the necessary capacity required to properly run an application. This has been the case in this exercise where the original disk subsystem for the database server was found to be unable to offer the I/O capabilities needed for the tuned setup.

What we have also shown is the iterative nature of performance tuning. During each iteration system parameters are changed and the effects are checked using the defined benchmarks. As so often the 80-20 rule applies to system tuning – 80 % of the improvement potential can be realized in 20% of the time, the remaining 20 % take 80 % of the time.

9 * Trademarks

IBM, DB2, DB2 Universal Database, and WebSphere are trademarks or registered trademarks of the IBM Corporation in the United States, other countries, or both.

Java and all Java-based trademarks are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

Microsoft, Windows, Windows NT, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Linux is a registered trademark of Linus Torvalds in the United States, other countries, or both.

Other company, product, or service names may be trademarks or service marks of others.