IBM ®

# *SOA Performance on Linux*

**Tuning Hardware, Linux, and WebSphere for Web Services, Service Mediations, and Business Processes**

**Examining the Impact of Server Virtualization and WebSphere Virtual Enterprise on Business Processes in Linux Environment**

*Khoa Huynh*
*Steve Dobbelstein*
*Mark Peloquin*
*Vivek Kashyap*

*IBM Systems and Technology Group*

# Contents

## *Abstract*

*This paper examines the service-oriented architecture (SOA) performance in the Linux® environment. IBM has defined five SOA Foundation entry points to help businesses get started with SOA.  This paper focuses on the performance tuning for Web services (Reuse entry point), service mediations (Connectivity entry point), and business processes (Process entry point) in the Linux environment.  The paper also provides some discussion on the two remaining SOA People and Information entry points.  In addition, we also look at the impact of server virtualization and WebSphere® Virtual Enterprise dynamic operations on the performance of business processes in the Linux environment.  WebSphere Virtual Enterprise supports dynamic operations by providing the capability to manage a dynamic cluster of application servers that can be activated on a pool of physical servers in response to changes in the workload mix to meet user-defined performance goals.  We include IBM® X-Architecture® servers, as well as IBM Power servers with PowerVM™ virtualization technology, in our study.  For performance measurements, we use a benchmark that models the business processes and Web services provided for a typical automobile insurance company.*

# 1. Introduction

IBM has defined five service-oriented architecture (SOA) Foundation entry points to help businesses get started with SOA in their enterprise environment. These five entry points are *People*, *Process*, *Information*, *Connectivity,* and *Reuse* [1]. This paper focuses on the following entry points:

- Reuse entry point, which is the enablement of Web services and how they can be reused throughout the SOA implementation

- Connectivity entry point, which encompasses the transformation and routing service mediations on the Enterprise Service Bus (ESB)

- Process entry point, which encompasses the Business Process and Service Choreography aspect of SOA

This paper examines how we can tune hardware, Linux, WebSphere, and DB2[®] components to achieve optimal performance for an SOA benchmark that models the Web services, business processes, and service routing mechanisms for a typical automobile insurance company. We will conclude with a set of specific recommendations for each of the SOA entry points. Additionally, we will briefly discuss the components of the People and Information entry points and provide some references to tuning guides for these entry points.

# 2. Background information and performance evaluation methodology

In this paper, we evaluate the performance of Web services, transformation and routing mediations on the service requests, as well as business processes hosted on the WebSphere Process Server (WPS) V6.1. Let us first discuss some basic WebSphere terminology and structure.

WebSphere Process Server is an SCA-compliant runtime element that provides a standards-based process engine [3]. The foundation of WPS is the WebSphere Application Server, which is the IBM implementation of the Java™ 2 Enterprise Edition (J2EE) platform, which conforms to V1.4 of the J2EE specifications [2].

## 2.1. WebSphere Application Server structure

A *base* WebSphere Application Server (WAS) installation includes everything needed for a single application server instance. Additional server definitions can be logically grouped into *nodes*. A node can contain many application servers, but cannot span multiple physical servers. A single physical server can have multiple nodes installed on it, each with multiple managed application servers. Multiple nodes can be grouped together into a *node group* or into another logical grouping called a *cluster*. A *cell* contains one or more node groups, clusters, or both.

A WebSphere *Network Deployment* installation provides centralized administration and workload management for a cell. A cell has a master administrative repository that holds the configuration information for all nodes in the cell. There is a *Deployment Manager* through which all nodes in the cell can be managed. The Deployment Manager has a graphical interface called the *Integrated Solutions Console* through which a WebSphere administrator can perform everything from managing a node to deploying an application to any node or cluster in the cell. The Deployment Manager communicates to each node through a *node agent*. The node agent, which is a specialized application server, must be started on a node before the Deployment Manager can find it.

WebSphere *Virtual Enterprise* extends the Network Deployment environment by providing on-demand capabilities that allow a dynamic cluster of application servers to respond in real time to changing workload demands, thereby improving the operational efficiency of the servers.

## 2.2. WebSphere Process Server environment

Every WebSphere Process Server (WPS) environment involves three fundamental layers:  WPS applications, a messaging infrastructure, and one or more relational databases.  More specifically:

- WPS applications include the process server infrastructure code, such as the Business Process Choreographer (BPC) and any user applications that exploit the process server functions.  These applications require a WPS application server to be installed and run.

- A messaging infrastructure is required for WPS and uses four WebSphere Service Integration (SI) buses:
    - Two buses for the Service Component Architecture (SCA) support (SCA.SYSTEM and SCA.APPLICATION buses)
    - One bus for the BPC (BPC SI bus)
    - One bus for Common Event Infrastructure (CEI) asynchronous event publishing (CEI bus)

  When an application server (or a cluster of application servers) is added to an SI bus, a Message Engine (ME) is created.  The ME is the component in the WAS process which implements the logic of the messaging infrastructure.

- Relational databases are required for WPS and the messaging infrastructure to store certain application configurations, runtime information, and persistent data.  Two main databases must handle much of the traffic in a WPS environment:
    - The Business Process Choreographer database (BPEDB), which stores data objects related to business processes
    - The Service Integration Bus database (SIBDB), which stores data objects for events and message persistence

## 2.3. WebSphere Virtual Enterprise environment

In our setup, we also use WebSphere Virtual Enterprise (WVE) V6.1, which provides application server virtualization, resource management, and a host of advanced operational facilities, such as performance and health monitoring.  This combination of capabilities is sometimes collectively referred to as *dynamic operations*.  One key feature of dynamic operations is that they can respond in real time to changes in the workload mix (without human intervention if so desired) to ensure that performance goals set by the user can be met.  Following are the key elements and features of WVE:

- A *dynamic cluster* of application servers, which run the same applications.  Applications are first installed and configured on the dynamic cluster.  They are then propagated to all active application servers in the cluster.

- A *service policy* that defines a performance goal for one or more applications.

- An *On-Demand Router (ODR),* which is a gateway through which Web service requests flow to the back-end dynamic cluster of application servers.  The ODR's primary functions include classification of incoming requests, based on rules defined by the user, and intelligent request routing, based on sense and response mechanisms from the back-end servers.

- A *Web server,* which is placed in front of the ODR as a trusted proxy server.  In this study, we use the *IBM HTTP Server*.

- Web service requests flow through the Web server and are intelligently routed to active application servers in the dynamic cluster by the ODR, based on the performance information collected from the cluster members.  Workload is balanced across all servers in the cluster to

achieve the performance goals that are specified in the service policy.  This dynamic workload balancing is based on load distribution, service policy, and available resources.  This capability is provided by three autonomic managers associated with the ODR:  the Application Placement Controller (APC), Dynamic Workload Manager (DWLM), and Autonomic Request Flow Manager (ARFM).  They make decisions on health management, traffic shaping, and application placement for the ODR.

Dynamic clustering in the WVE environment involves the clustering of WPS applications and the messaging infrastructure.  Conceptually, the clustering of WPS applications is not very different from clustering plain J2EE applications in the WebSphere Application Server environment.  WebSphere clustering techniques in the static environment are also applicable to the dynamic WVE environment.  These techniques are discussed in *Clustering WebSphere Process Server V6.0.2*, Part 1: Understanding the topology [4]; however, the messaging infrastructure is significantly more complex.

If a cluster of application servers is added to a Service Integration (SI) bus, each server in the cluster is capable of running the Message Engine (ME) created for that cluster.  However, only one server can have an active instance of the Message Engine at any given time.  A Message Engine can be configured to work with WPS applications by using two approaches:

- The Message Engine is local to the cluster of WPS applications.  In this case, the ME runs within the same application cluster as the WPS applications.
- The Message Engine is located in its own cluster, separate from the WPS applications.  This is also called the "Silver" topology in a classification of WebSphere clustering topologies [4].

In this study, we consider both approaches.

## 2.4. Benchmark

In this study, we use a benchmark that models the business processes and Web services that are provided for a typical automobile insurance company [10].  The benchmark specifies a macro workload whose driver can generate an end-to-end workload similar to that of an actual production system in an SOA environment.  *Figure 2.1* shows the components of the benchmark.

The benchmark makes extensive use of IBM SOA platform products in the following areas:
- Enablement of Web services, using IBM WebSphere Application Server (WAS)
- Business process choreography, using integration and choreography features of IBM WebSphere Process Server (WPS)
- Integration of Web services, using IBM WebSphere Enterprise Service Bus (WESB) or DataPower appliances

Each of the areas mentioned above can be included or excluded from performance evaluations.  The use of DataPower appliances is out of the scope of this paper.

In the benchmark, the Web services are implemented as part of a *ClaimServices* application.  These Web services represent typical services that are involved in the processing of an automobile insurance claim, such as creating a claim, updating a claim, approving or denying a claim, checking insurance coverage, generating a list of approved repair shops, selecting a repair shop, and informing the customer.  Some business logic is embedded in the implementation of these services.  However, the presence of business logic might hinder us in evaluating the performance of the underlying middleware layers supporting Web services as well as investigating potential problems that might occur.  As a result, we decided to run the benchmark in *Infrastructure Mode,* which keeps the business logic in the Web services to a minimum; it only performs minimal calculations and returns responses.
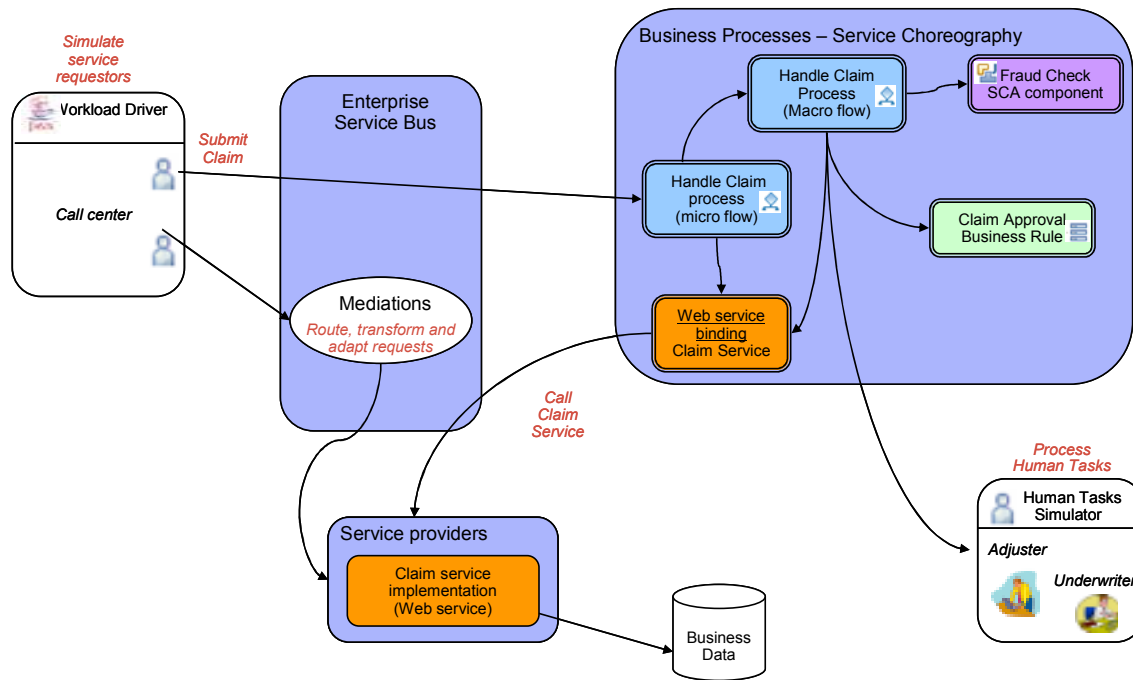
*Figure 2.1 – Architecture of benchmark used in our evaluation*

For service mediations, we consider both routing and transformation mediations. Transformation mediations can be performed on service requests, and in some cases, responses, using Extensible Stylesheet Language Transformations (XSLT). XSLT is an XML-based language used for the transformation of XML documents into other XML or human-readable documents. There are different levels of complexity for these transformations, and we will consider the following:

- *XSLT Value Transformation,* which transforms the value of a single element in the service request message using XSLT.
- *XSLT Namespace Transformation,* which transforms service requests and responses from one schema to another using XSLT. In this case, the schemas are largely the same but the name of an element differs and the two schemas have different namespaces.
- *XSLT Schema Transformation,* which transforms service request and response messages from one schema to another using XSLT. In this case, the schemas are completely different, but they contain similar data which is mapped from one to the other. In addition to the transformation, a value from the service request is transferred to the response message by storing it in a context header.
- *Composite Module* mediation consists of three mediation primitives wired together inside a single, composite mediation module. The three mediation primitives are authorization (a routing mediation that checks a password field in the body of the service request), route on body, and transform. The composite module approach is good for performance because it saves the overhead of inter-module calls, but at the expense of the ability to individually administer the pieces of the overall mediation.

Routing mediations route requests to different Web services based on content. In particular, we will examine *route-on-body* mediation, which routes each request to the appropriate Web service based on the content of a field in the body of that request.

The processing of auto insurance claims is done through a Business Process Execution Language (BPEL) application called *HandleClaim*. *Figure 2.2* illustrates the *HandleClaim* application, which is implemented using the business process choreography features that are provided by WebSphere Process Server (WPS). After a customer submits a claim, the *HandleClaim* process calls an SCA component called *FraudCheck* to determine whether the claim is fraudulent or not. If a claim is

fraudulent, it is rejected, and the processing of the claim is completed. On the other hand, if the claim is found to be valid, an *Adjuster* is called upon to evaluate and update the claim. Next there is a business rule which determines whether the value of the claim is less than $500. If so, the claim is approved automatically; otherwise, it would have to be approved in person by an *Underwriter*. The actions of Adjusters and Underwriters are simulated by another application – the *HumanTaskSimulator* – which polls and performs the roles of adjusters and underwriters. After much experimentation, we found that setting the parameters of the *HumanTaskSimulator* to the following values was more than adequate for the workloads considered in this study:

- Number of adjusters = 30
- Number of underwriters = 20
- Polling period for adjusters = 200 ms
- Polling period for underwriters = 400 ms

After a claim is approved, a check is sent to the customer, and the processing of the claim is completed. The underlying business logic of the claim services, such as creating, updating, rejecting, approving, and completing a claim, is performed by calling the Web services implemented in the *ClaimServices* application. In *Figure 2.2*, the claim services are represented by the rectangular boxes in blue.

The benchmark's *Workload Driver* is a stand-alone, multithreaded HTTP client which can generate concurrent insurance claim requests to the *HandleClaim* application using the Service Oriented Access Protocol (SOAP) implemented on top of the HTTP transport protocol [9].
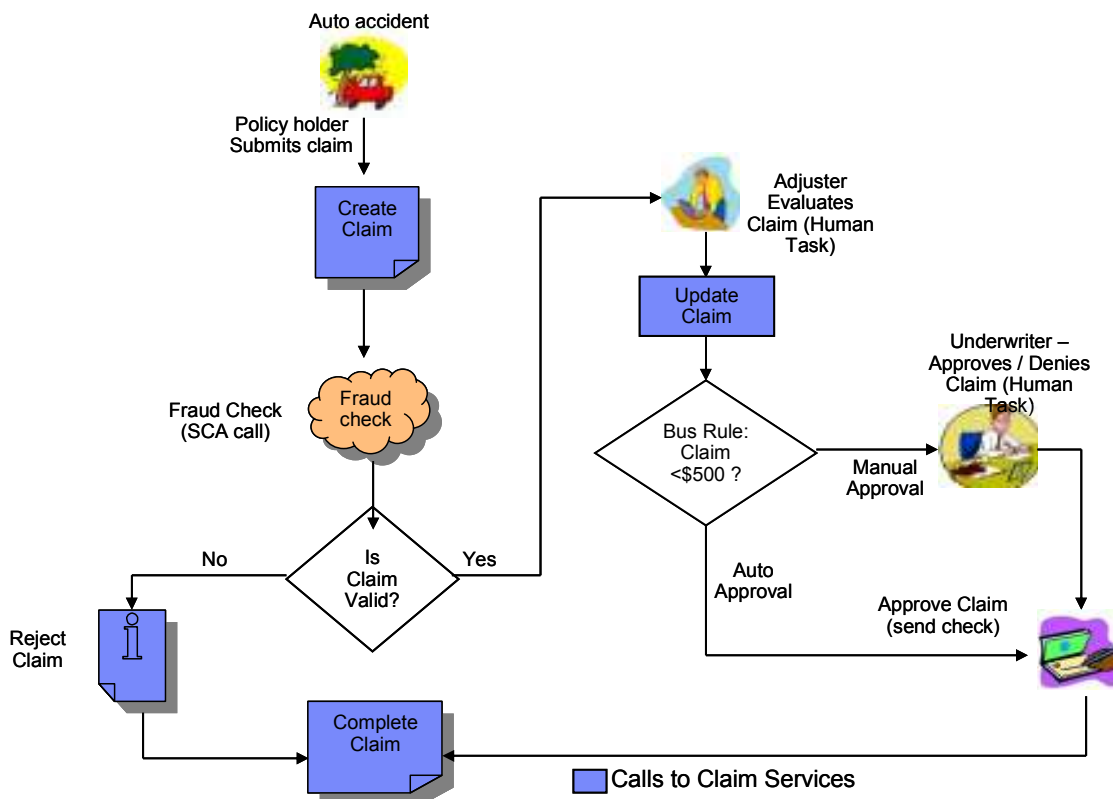


*Figure 2.2 – Insurance claim processing workflows*

In this paper, we consider both IBM X-Architecture and IBM Power Architecture® server platforms.

In most of our tests, we start warm-up runs (which can take as long as 300 seconds depending on the workload level) prior to actual data collection to ensure optimal and consistent results. Warm-up runs

were especially needed because, by default, the IBM Java Virtual Machine (JVM) in WebSphere Application Server uses a higher optimization level for compiles, thus resulting in faster runtime performance, but at the expense of slower server startups.

# 3. Systems configurations

## 3.1. Hardware

As mentioned previously, we consider both IBM X-Architecture and Power Architecture server platforms for hosting the benchmark's Web services and business processes.

### 3.1.1. IBM X-Architecture servers

In this study, we employ the IBM System x3850 M2 (*Table 3.1*), which implements the IBM eX4 chipset [7,8], to host WebSphere, DB2, and the benchmark's Web services and business process components.

| | |
|---|---|
| **Server** | IBM System x3850 M2 |
| **CPU** | 4 x 64-bit Quad-Core Intel® Xeon® Processor X7350 (2.93 GHz) |
| **Memory** | 64 GB (667 MHz DDR2) |
| **Network** | Integrated Dual-Port Gigabit Ethernet with TCP/IP offload engine |

*Table 3.1 – IBM System x3850 M2 configuration*

The benchmark's Workload Driver runs on an IBM System x3650 (*Table 3.2*).

| | |
|---|---|
| **Workload Driver** | IBM System x3650 |
| **CPU** | 2 x 64-bit Quad-Core Intel Xeon X5460 (3.16 GHz) |
| **Memory** | 24 GB (667 MHz DDR2) |
| **Network** | Integrated Dual-port Gigabit Ethernet |

*Table 3.2 – IBM System x3650 configuration*

### 3.1.2. IBM Power architecture servers

For the Power platform, we use an IBM Power 570 (*Table 3.3*) [5] with POWER6™ processors [6] to host WebSphere and the benchmark's applications in some tests and two IBM OpenPower® 720 servers (*Table 3.4*) [11] with POWER5™ processors in other tests.

| | |
|---|---|
| **Server** | IBM Power 570 |
| **CPU** | 2 x 64-bit Dual-Core IBM POWER6 (4.7 GHz), 4 MB L2 cache per core, 32 MB L3 cache shared per two cores |
| **Memory** | 32 GB (667 MHz DDR2) |
| **Network** | Dual-Port Gigabit Ethernet |
| **Internal Storage** | 1 x SAS controller with 2 x 300 GB, 15K rpm SAS drives |
| **Threading** | Simultaneous multi-threading (SMT) technology |

*Table 3.3 – IBM Power 570 configuration*

| Server | IBM OpenPower 720 |
|---|---|
| **CPU** | 2 x 64-bit Dual-Core IBM POWER5 (1.5 GHz), 1.9 MB L2 cache, 36 MB L3 cache |
| **Memory** | 16 GB |
| **Network** | Dual-Port Gigabit Ethernet |
| **Internal Storage** | 1 x SAS controller with 4 x 36 GB / 72 GB 15K rpm SAS drives |
| **Threading** | Simultaneous multi-threading (SMT) technology |

*Table 3.4 – IBM OpenPower 720 configuration*

All servers are connected to a Cisco Systems Catalyst 3750 Series Gigabit Switch (Model WS-C3750G-24TS-S).

## 3.2. Software

The Linux operating system on the IBM System x3850 M2 server is Novell SUSE Linux Enterprise Server (SLES) 10 Service Pack (SP) 1 for AMD64 and EM64T (x86_64).

The Linux operating system on the IBM Power 570 and OpenPower 720 is Novell SUSE Linux Enterprise Server (SLES) 10 Service Pack (SP) 1 for PPC (ppc64).

The operating system on the workload driver system (IBM System x3650) is Novell SUSE Linux Enterprise Server (SLES) 10 Service Pack (SP) 2 for AMD64 and EM64T (x86_64).

All servers run with WebSphere Application Server (WAS) V6.1.0.17, WebSphere Process Server V6.1.0.2, and WebSphere Virtual Enterprise V6.1.0.4.

# 4. Performance results

First let us look at how we can tune Linux and WebSphere components for Web services. We then examine how we can tune those components for routing and transformation mediations on Web services. Finally, we discuss how to create an optimal environment for business processes using those Web services.

## 4.1. Tuning Web services

For tuning Web services, let us consider the setup illustrated in *Figure 4.1.1*. The Workload Driver generates Web service requests to the benchmark's *ClaimServices* application using the Service Oriented Access Protocol (SOAP) implemented on top of the HTTP transport protocol [9]. The Workload Driver uses up to 50 concurrent threads to generate as many service requests as the *ClaimServices* application can handle. The server that hosts the Web services (that is, the one that runs WPS and the ClaimServices application) is an IBM Power 570 server with POWER6 processors. Alternatively, we also configure an x86 server – an IBM x3850 M2 server – as the Web server. While we focus on the performance results obtained on the Power 570 server, we will note the performance data obtained on the x3850 M2 server as well. The Workload Driver runs on a separate x86 server (an IBM x3650).



*Figure 4.1.1 – Setup for tuning Web services*

*Figure 4.1.2* shows the geometric means of the service request rates obtained for each of the tuning items considered in this section.
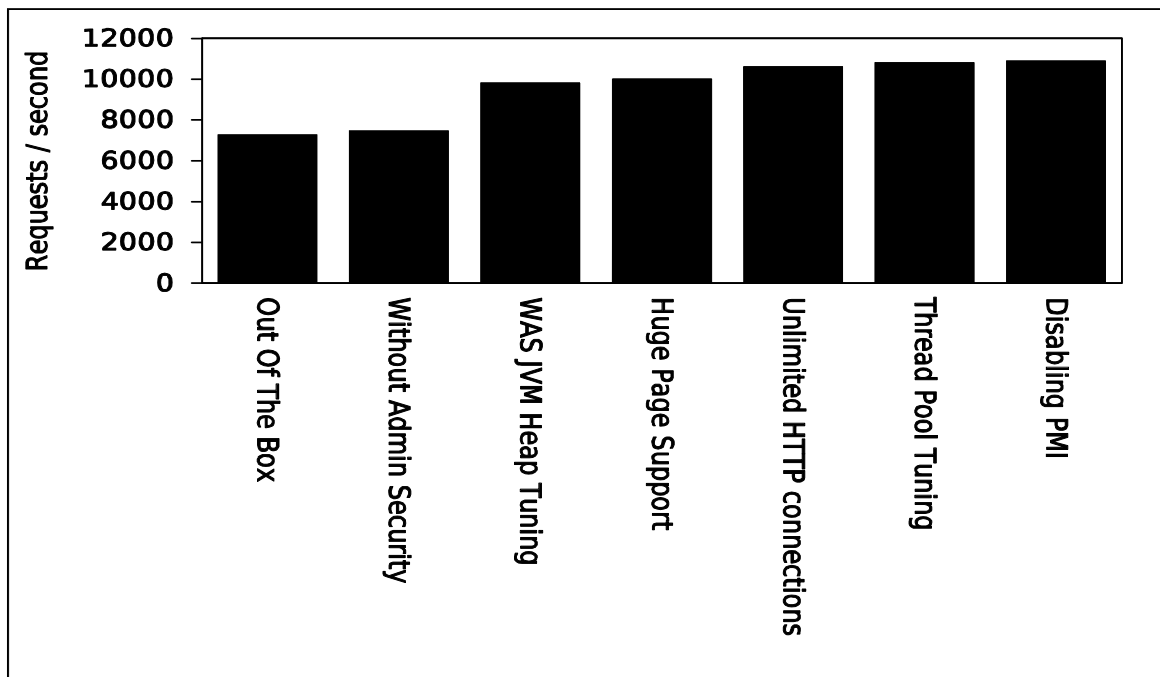


*Figure 4.1.2 – Performance impact of tuning items*

- **Out-of-the-box configuration.**  The leftmost column in *Figure 4.1.2* shows the geometric mean of the Web service request rates in the out-of-the-box configuration – with all default settings for WebSphere Application Server and Linux.  Note that administrative security is enabled in the default profile for WebSphere Application Server.  Enforcing security involves additional overhead for authentication and access validation.

  *Result*: This configuration has a geometric mean score of 7254.90 requests per second on the Power 570 server.

- **Without administrative security.**  As mentioned previously, administrative security is enabled in the default profile for WebSphere Application Server.  Enforcing security involves additional overhead for authentication and access validation.  If there is no requirement to enforce administrative security, it is better to create a WebSphere Application Server profile with administrative security disabled.

  *Result:*  As shown in *Figure 4.1.2*, the geometric mean score is 7445.44 requests per second on the Power 570 server – a 2.6 % improvement over the out-of-the-box configuration.  On an x86 server (x3850 M2), disabling administrative security yields an 8.6% improvement over the out-of-the-box configuration.

- **JVM heap tuning**.  In the Java world, the JVM heap configuration has a significant impact on performance.  Several parameters are available for tuning the JVM heap for better performance.  Two basic JVM heap parameters are the size of the heap and the garbage collection policy.

  The size of the WAS JVM heap is probably the most important factor for its performance.  A too-small heap causes garbage collection to happen more frequently.  A too-large heap causes garbage collection to happen less frequently, or to take longer to compact the large heap.  There are several tools available, such as the Tivoli® Performance Viewer (included with WebSphere), that can help analyze and monitor the heap usage and garbage collection so that the heap can be specifically tuned for a particular workload.

  The default size of the JVM heap is typically too small.  In our tests, by looking at the number of memory pages allocated to the heap that are actually in use while the Web service scenarios are being run, we find that the heap usage is between 1536 MB and 2048 MB.  As a result, we set the JVM heap size to 2048 MB.  This larger heap gives us a significant performance improvement over the default heap size.  Because the Power 570 server has 32 GB of physical memory, which is more than adequate for our tests, we set the minimum and maximum heap size to the same value (2048 MB) to avoid the overhead of frequent heap size changes and to ensure optimal performance.

  In addition to the JVM heap size, the garbage collection policy can also affect performance.  The WAS JVM supports four different garbage collection policies.  The default garbage collection policy is `optthruput`.  During a garbage collection cycle under the `optthruput` policy, all application threads are stopped for mark, sweep, and compaction if needed.  The garbage collector scans all the objects in the heap, marking any object that is in use, sweeps up the unused objects, reclaims their memory, and then compacts the remaining memory to reduce fragmentation.  The entire process can take some time.  All application threads are paused while the garbage is collected.  Consequently, the `optthruput` policy results in longer garbage collection pause times, but more often than not, it yields the best overall throughput.  However, we decided to use the `gencon` (Generation Concurrent) garbage collection policy, which handles short-lived objects differently from long-lived objects, because our testing shows that the `gencon` policy is more suitable for Web service scenarios.  Under the `gencon` policy, the heap is split into new and old segments. Long-lived objects are promoted to the old space while short-lived objects are garbage collected quickly in the new space (called a *nursery*).

  We set the size of the *nursery* to 1536 MB (75% of the total heap size) by going through the WAS

Administration Console:

1.  Go to **Servers → Application Servers →** *server name* **→ Server Infrastructure → Java and Process Management → Process Definition → Additional Properties → Java Virtual Machine.**
2.  Enter **‑ Xgcpolicy:gencon ‑Xmn1536M** in the *Generic JVM arguments* field. Note that the JVM heap's maximum and minimum sizes can also be set on this page. In our test, both the minimum and maximum heap sizes were set to 2048 MB.

*Result*: The WAS JVM Heap Tuning column in *Figure 4.1.2* shows the geometric mean of the Web service request rates on the Power 570 server with the WAS JVM heap size of 2048 MB, **gencon** garbage collection policy, and a 1536 MB nursery. The geometric mean score is 9797.22 requests per second – an improvement of more than 35% over the out-of-the-box configuration. On an x86 server (x3850 M2), increasing the WAS JVM heap size to 2048 MB results in a 75% performance improvement over the out-of-the-box configuration, and switching the garbage collection policy to **gencon** yields another 7% performance gain.

*   **Huge page support in Linux**. One of the factors to consider in system performance is memory access. Processors keep a cache of mappings from virtual addresses to physical addresses in a translation look-aside buffer (TLB) so that they do not have to walk through the page tables for every virtual address used. The TLB holds a fixed number of entries. When the processor encounters a virtual address that is not in the TLB, called a TLB miss, it must walk the virtual address through the page tables to get its physical address. The new mapping is then written to the TLB, overriding an existing entry. Linux has support for *huge pages* (sometimes also known as *large pages*). Normal pages are 4 KB in size, but the size of each huge page is 16 MB in the Power architecture and 2 MB in the Intel x86 architecture. The use of huge pages can improve memory performance in two ways. First, with huge pages, a single entry in the TLB maps to more memory; therefore, fewer TLB entries are needed to map an area of memory. With fewer TLB entries, the frequency of TLB misses is reduced. Secondly, the page table setup for mapping huge pages uses one less table to determine the physical addresses, thus making virtual-to-physical-address mapping for huge pages faster than for normal pages.

Since the WAS JVM heap size is set to 2048 MB (larger than the default size), the CPU overhead of managing and keeping track of memory in this large heap can be reduced by exploiting the huge page support that is provided in the Linux kernel.

To allow WAS JVM heap to use huge pages, we must first enable Linux to create and maintain a pool of huge pages. On the Power 570 server, we created a pool of 3 GB of huge pages (a total of 192 of 16 MB huge pages) by adding the following lines to the **/etc/sysclt.conf** file:

```
#Number of huge pages (192 x 16 MB = 3 GB)
vm.nr_hugepages = 192
#Size of shared memory is set to 4 GB (4294967296 bytes)
kernel.shmmax = 4294967296
kernel.shmall = 4294967296
```

(Note that we set the amount of shared memory to 4 GB so that there would be enough room for the 3 GB of huge pages in the shared memory pool.)

*Result*: The Huge Page Support column in *Figure 4.1.2* shows the result that is obtained by exploiting the Linux huge page support on the Power 570 server. The geometric mean of the Web service request rates is now 9999.53 requests per second, which is another 2.1% improvement over what we are able to achieve by tuning the WAS JVM heap. Cumulatively, the WAS JVM heap tuning and the huge page support yield a total of 38% performance improvement over the out-of-the-box configuration on the Power 570. On an x86 (x3850 M2) server, with the huge page size of only 2 MB, we find that huge pages only help when the Java heap is

constrained to a size that cannot accommodate all objects; if the heap could be made large enough, huge pages would not make much difference.

- **HTTP connection settings**. Several WAS Web Container settings can be tweaked for maximum concurrency in our tests. If there are not enough HTTP connections available, incoming clients cannot connect until a connection is freed. If the server's CPUs are not fully utilized, if there is no memory constraint, and if there is available network bandwidth, the number of persistent HTTP connections for each port can be increased from the default value of 100 for better performance. In our tests, setting the maximum number of HTTP persistent requests to `unlimited` gave us a noticeable performance improvement over what we had with WAS JVM tuning and huge page support. The results are shown in *Figure 4.1.2*.

  To change the number of HTTP connections available for a given port, we used the WAS Administration Console as follows:

  1. Go to **Servers** → **Application Servers** → *server name* → **Communications** and click *Ports*.
  2. Find the port number in the table and click on *View associated transports* for that port.
  3. Click on the transport chain that is listed.
  4. Click on *HTTP inbound channel (HTTP_n)*, where *n* denotes channels 1 to 4.
  5. Either click on *Maximum persistent requests per connection* and enter a number in the *Specify maximum number of persistent requests* field, or click *Unlimited persistent requests per connection*.

  <u>*Result*</u>: As shown in the Unlimited HTTP Connections column in *Figure 4.1.2*, the geometric mean of the Web service request rates on the Power 570 is 10,607.97 requests per second, which is another 6.1% improvement over what we are able to achieve by tuning the WAS JVM heap and exploiting the huge page support. Together with the WAS JVM heap tuning and huge page support, the *Unlimited HTTP connection* setting cumulatively yields a total of 46% performance improvement over the out-of-the-box configuration on the Power 570 server. On an x86 (x3850 M2) server, the *Unlimited HTTP connection* setting produces an 8.3% improvement.

- **Thread pool settings**. There is another area that we can tune for maximum concurrency: the number of threads available to service requests from the clients. For example, threads in the Web Container thread pool are used for handling incoming HTTP and Web service requests. These thread pools are shared by all applications that are deployed on the server, so in many cases, these pools need to be larger than their default sizes. We experiment with larger maximum numbers of threads in the default thread pool as well as the Web container thread pool. These numbers partly depend on the number of CPUs on the Web server because the more CPUs we have, the more threads can be executed concurrently. In our test setup, the Power 570 server has two dual-core POWER6 processors, and if we take simultaneous multi-threading (SMT) into account, we would actually have eight logical processors that can be seen by Linux. Based on recommendations from the WebSphere Performance Team, with eight logical processors, we set the maximum number of threads in the default thread pool to 200 (default value is 20) and in the Web container thread pool to 100 (default value is 50). However, we find that, in our tests, just increasing the thread pool maximum values alone would actually degrade the performance because the JVM heap would become too small to accommodate the higher number of threads that can be executed concurrently. To get any performance benefit from the larger thread pools, we would need a larger JVM heap. Our test results confirm that larger thread pools, together with a larger 3-GB JVM heap, yield a small, but noticeable, performance improvement.

  Changing the number of threads in a thread pool can be done through the WAS Administration Console as follows:
  1. Go to **Servers** → **Application Servers** → *server name* → **Additional Properties** → **Thread Pools.**

2. Click the thread pool that you want to change, and enter new values in the *Maximum Size* fields.

*Result*:  The Thread Pool Tuning column in *Figure 4.1.2* shows the cumulative performance impact of WAS JVM heap tuning (including the larger 3-GB heap), huge page support, Web container tuning, and larger thread pools on the Web services (in terms of average requests per second).  The geometric mean of the Web service request rates on the Power 570 is now 10,793.84 requests per second – another 1.8% improvement over what the preceding configuration.  All of our tuning items so far have yielded a total of 49% performance improvement over the out-of-the-box configuration on the Power 570 server.  On the x3850 M2 server, however, we find that larger thread pools do not make much performance difference.

- **Disabling Performance Monitoring Infrastructure (PMI)**.  To further optimize the performance of Web services in our tests, we next turned off all performance monitoring, tracing, and logging.  These are often necessary when setting up a server or when debugging problems or issues, but they do introduce some performance overhead.  As a result, it is recommended that tracing and monitoring be used judiciously, and whenever possible, turned off entirely to ensure optimal performance.  Disabling the WAS Performance Monitoring Infrastructure (PMI) can be done through the WAS Administration Console as follows:

  1. Go to **Monitoring and Tuning → Performance Monitoring Infrastructure (PMI) →** *server name.*
  2. Clear the *Enable Performance Monitoring Infrastructure (PMI)* box, and in the *Currently Monitored Statistic Set* box, select *None*.

  *Result*:  The *Disabling PMI* column in *Figure 4.1.2* shows the performance of Web services when we disable all performance monitoring, tracing, and logging (on top of the previous tunings that we have done).  The geometric mean of the Web service request rates on the Power 570 is now 10,878.47 requests per second – another 0.8% improvement over the preceding configuration.  On the x3850 M2 server, disabling PMI only results in a negligible 0.4% performance improvement.

*Summary*:  All of the tuning items discussed in this section have yielded a cumulative total of 50% performance improvement over the out-of-the-box configuration on the Power 570 server.  On the x3850 M2 server, the same tuning items produce 120% improvement over the out-of-the-box configuration due to the 75% performance improvement when the WAS JVM heap is increased to 2048 MB.  From the data shown in *Figure 4.1.2*, it is clear that tuning the WAS JVM heap (increasing the heap size to 2 GB, using the `gencon` garbage collection policy with a nursery size of 1.5 GB) gives us the most performance gain, followed by setting the maximum number of persistent HTTP connections for Web container ports to `unlimited.`  Other tuning items produce discernable, but not significant, performance benefits.

Finally, we want to see if there is any performance bottleneck in the most optimal configuration – with all tuning items applied – in our setup.  Although it varies a little during different Web service invocations, the CPU utilization on the Power 570 server never goes above 80%.  The network throughput between the Workload Driver and the Power 570 server always remains below 50 Mbytes/s – well within the available bandwidth of the private Gigabit Ethernet network in our laboratory.  As for the physical memory usage on the Power 570 server, the entire workload uses only 6 GB of the 32 GB of physical memory available.  The disk I/O traffic is fairly negligible – less than 100 Kbytes/sec on the Power 570 server.  Similarly, no performance bottleneck is observed on the x3850 M2 Web server.

## 4.2. Tuning routing and transformation service mediations

For tuning routing and transformation service mediations, consider the setup illustrated in *Figure 4.2.1*.  The Workload Driver generates Web service requests which go through the service mediations provided by the Enterprise Service Bus (ESB) before getting to the benchmark's *ClaimServices* application that is hosted on WebSphere Application Server.  The Workload Driver is a stand-alone, multi-threaded HTTP

client, which uses up to 50 threads with 20 maximum transactions to generate SOAP service requests to the *ClaimServices* application.   The service mediations include XSLT transformation and route-on-body mediations.  In other words, the Web service mediation tests have the following characteristics:

- Stand-alone, multi-threaded HTTP client to produce SOAP service requests
- Synchronous SOAP / HTTP request and response invocation
- XSLT transformation and route-on-body service mediations
- *ClaimServices* application hosts Web services

The Web services and the ESB's service mediations are hosted on an IBM Power 570 server with POWER6 processors.   We also consider an IBM x3850 M2 for this role, and its performance data is noted in this section along with the data for the Power 570 server.  The Workload Driver runs on another server – an IBM x3650.
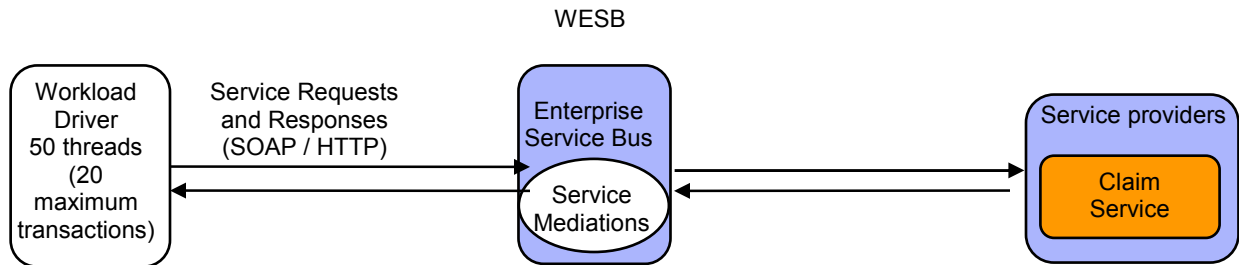


*Figure 4.2.1 – Setup for tuning Web service mediations*

Our testing indicates that WebSphere JVM heap optimization, Linux huge page support, and tuning for maximum concurrency appear to yield the most performance improvement for service mediation scenarios.  Let us first look at the service mediation performance with the payload size set to 3 KB for both service request and response messages.

- **JVM heap tuning**.  In *Section 4.1*, we found that the WAS JVM heap configuration had a significant impact on the performance of Web services; therefore, we are going to start by looking at the WAS JVM heap configuration for the mediation scenarios here.  As in *Section 4.1*, in order to change the WAS JVM heap size or set the garbage collection policy, we can use the WebSphere Application Server Administrative Console as follows:

  - Go to **Servers → Application Servers → *server name* → Server Infrastructure → Java and Process Management → Process Definition → Additional Properties → Java Virtual Machine.**
  - Enter new sizes (in MB) in the *Initial Heap Size* and *Maximum Heap Size* fields.
  - Enter appropriate JVM command line option for the garbage collection policy into the *Generic JVM arguments* field.  For example, if we want to use the `gencon` garbage collection policy with a nursery size of 1536 MB, we would enter `–Xgcpolicy:gencon –Xmn1536M` in the *Generic JVM arguments* field.

  *Result*:  With payload size for both requests and responses set to 3 KB and with the default WAS JVM settings, we could only achieve an average Web service request rate of 111.03 requests per second in the Composite Module mediation scenario, as shown in *Table 4.2.1*.  To optimize the JVM heap, we first consider the default GC policy (`optthruput`).  By setting both the initial JVM heap size and the maximum heap size to 2048 MB, we immediately get a significant performance boost to 664.26 requests per second – a 600% improvement!  The overall garbage collection overhead is 5%, which is quite acceptable.

  Increasing the JVM heap size to 3072 MB yields another 6% performance improvement.  However, increasing the JVM heap size to 4096 MB and beyond does not seem to make much difference.  As a result, we decide to keep the heap size at 3072 MB, but try to see if we could optimize the heap usage further by switching to another garbage collection policy.

The WAS JVM supports four different garbage collection policies, as shown in *Table 4.2.2*. With the JVM heap size kept at 3072 MB, the Generational Concurrent (`gencon`) garbage collection policy delivers the best performance for the Composite Module mediation scenario – as shown in *Table 4.2.1*. The `optavgpause` policy delivers the worst performance. The `subpool` policy, which is available only on IBM System p and z® servers, produces approximately the same performance as the default policy (`optthruput`), but not as good as the `gencon` policy.

Based on the data in *Table 4.2.1*, we can conclude that the `gencon` garbage collection policy, together with the JVM heap size of 3072 MB, represent the most optimal JVM heap configuration for the Composite Module mediation scenario (with payload size of 3KB for both requests and responses) on the Power 570 server. In fact, as *Figure 4.2.1* shows, this optimal WAS JVM heap configuration accounts for the largest performance gain across all Web service mediation scenarios considered in our study. However, on the x3850 M2 server, we find that the best WAS JVM heap configuration requires a much larger heap – 6144 MB – with the `gencon` garbage collection policy.

| GC Policy | JVM Heap Size (MB) | Throughput (Reqs/sec) | Overall GC Overhead | Total GC Pause |
|---|---|---|---|---|
| Default (optthruput) | 2048 | 664.26 | 5% | 11 secs |
| | 3072 | 702.57 | 5% | 8 secs |
| | 4096 | 688.94 | 4% | 6 secs |
| | 6144 | 693.48 | 4% | 6 secs |
| Gencon (nursery = 75% heap) | 3072 | 712.96 | 2% | 4 secs |
| | 4096 | 711.51 | 2% | 3 secs |
| optavgpause | 3072 | 670.41 | 2% | 4 secs |
| subpool | 3072 | 698.88 | 5% | 8 secs |

*Table 4.2.1 – Impact of tuning on Composite Module mediation scenario with payload size of 3 KB on the Power 570 server.*

| GC Policy | JVM Command Line Option | Description |
|---|---|---|
| Optimize for throughput | -*Xgcpolicy:optthroughput* (default) | This is the default policy. It is typically used for applications where raw throughput is more important than short GC pauses. The application is paused every time while garbage is being collected. |
| Optimize for pause time | -*Xgcpolicy:optavgpause* | This policy trades high throughput for shorter GC pauses by performing some of the garbage collection concurrently. The application is paused for shorter periods. |
| Generational concurrent | -*Xgcpolicy:gencon* | This policy handles short-lived objects differently from long-lived objects. Applications that have many short-lived objects can see shorter pause times with this policy while still producing good throughput. |
| Subpooling | -*Xgcpolicy:subpool* | This policy uses an algorithm similar to the default policy, but employs an allocation strategy that is more suitable for multiprocessor machines. This policy is recommended for SMP machines with 16 or more processors. This policy is only available on IBM pSeries® and zSeries® platforms. Applications that need to scale on large machines can benefit from this policy. |

*Table 4.2.2 – Garbage collection policies supported in WebSphere 6.1.*

**Impact of Tuning on WESB Mediations (Power 570)**
**(50 Client Threads, 20 Max Transactions,**
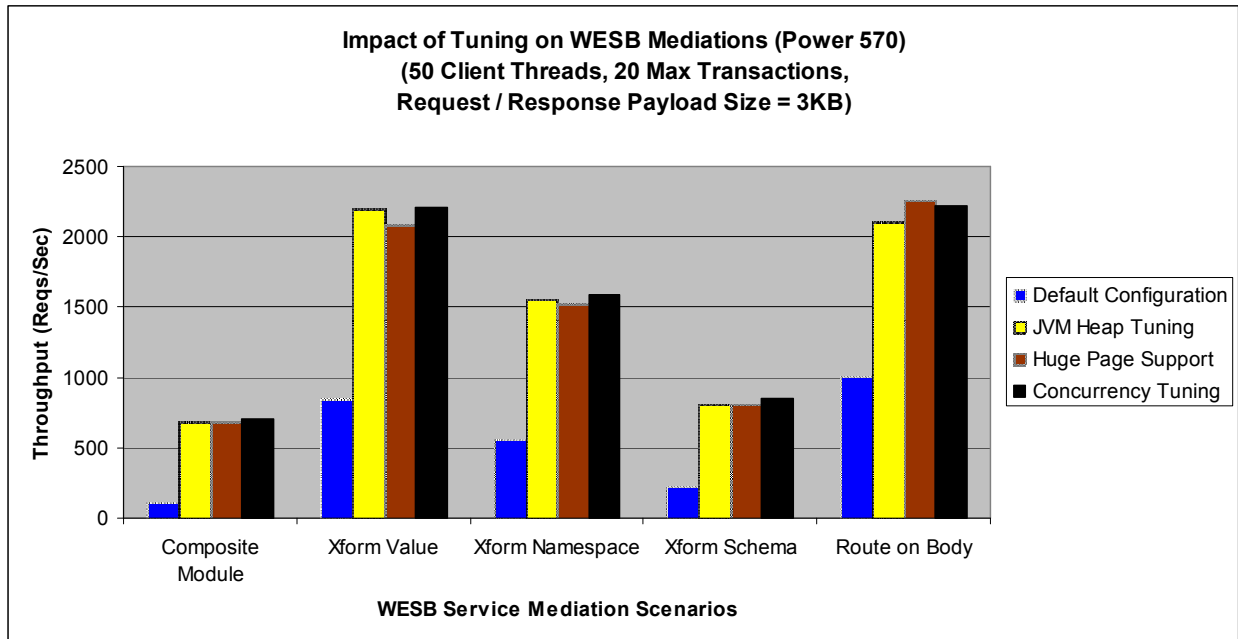**Request / Response Payload Size = 3KB)**



*Figure 4.2.1 – Cumulative performance impact of WAS JVM heap tuning, Linux huge page support, and WAS concurrency tuning*

- **Huge page support in Linux.** With the WebSphere JVM heap size now set to 3072 MB, the CPU overhead of managing and keeping track of memory in this large heap can be reduced by exploiting the Huge (Large) Page support provided in the Linux kernel. The Power processors support huge page size of 16 MB while x86 processors support only 2 MB huge page size. Huge pages are configured in Linux by adding the following lines to the **/etc/sysctl.conf** file:

```
vm.nr_hugepages = <number of pages>
kernel.shmmax = <number of bytes>
kernel.shmall = <number of bytes>
```

In our study, we want to configure enough huge pages for the WebSphere JVM heap. To allow the WebSphere JVM heap to use huge pages, we must first enable Linux to create and maintain a pool of huge pages. In our tests, we create a 4 GB pool of huge pages by adding the following lines to the **/etc/sysctl.conf** file:

```
#Number of huge pages (256 x 16 MB = 4 GB)
vm.nr_hugepages = 256
#Size of shared memory is set to 6 GB (6442450944 bytes)
kernel.shmmax = 6442450944
kernel.shmall = 6442450944
```

(Note that we set the amount of shared memory to 6 GB so that there would be enough room for 4 GB pool of huge pages in the shared memory pool.)

The WebSphere JVM can then be configured to use huge pages by adding the parameter **-Xlp** in the same "Generic JVM arguments" box where the garbage collector parameters are set.

The data in *Figure 4.2.1* shows that the huge page support does not result in any significant performance improvement for the transformation mediation scenarios that have a payload size of 3 KB, although it does yield a noticeable (8%) performance gain for the routing mediation scenario. This is somewhat expected because the transformation operations on 3 KB payloads are not expected to cross normal 4 KB page boundaries too frequently. On the other hand, route-

on-body service mediations involve fetching certain values in the message bodies, and then, based on those values, jumping to different Web services in memory at comparatively higher service request rates (more than 2,000 requests per second). This would cause more cache and TLB misses than other mediation types, so the huge page support is expected to help here. Indeed, it provides an 8% performance improvement for routing mediation scenarios. Similar behavior is also observed on the x3850 M2 server.

- **Tuning for maximum concurrency**. After optimizing the JVM heap memory usage, the next area for potential performance improvement is concurrency. There are several WebSphere Web Container settings that can be tweaked for maximum concurrency. If there are not enough HTTP connections available, incoming service requests cannot connect until a connection is freed. If the server's CPUs are not fully utilized, if there is no memory constraint, and if there is available network bandwidth, the number of persistent HTTP connections for each port can be increased from the default value of 100 to improve the server's performance. In our tests, we found that setting the maximum number of HTTP persistent requests to `unlimited` gave us a noticeable performance gain.

To change the number of HTTP connections available for a given port, we used the WAS Administration Console as follows:

1. Go to **Servers → Application Servers → *server name* → Communications** and click *Ports*.

2. Find the port number in the table and click on *View associated transports* for that port.

3. Click the transport chain that is listed.

4. Click *HTTP inbound channel (HTTP_n)*, where *n* denotes channels 1 to 4.

5. Either click *Maximum persistent requests per connection*, and enter a number in the *Specify maximum number of persistent requests* field, or click *Unlimited persistent requests per connection*.

There is another area that we can tune for maximum concurrency: the number of threads available to service requests from the clients. For example, threads in the Web Container thread pool are used for handling incoming HTTP and Web service requests. These thread pools are shared by all applications that are deployed on the server; therefore, in many cases, these pools need to be larger than their default sizes.

Changing the number of threads in a thread pool can be done through the WAS Administration Console as follows:

1. Go to **Servers → Application Servers → *server name* → Additional Properties → Thread Pools.**

2. Click on the thread pool you want to change and enter new values in the *Maximum Size* fields.

We experiment with larger maximum numbers of threads in the default thread pool as well as the Web Container thread pool. These numbers partly depend on the number of CPUs on the server because the more CPUs we have, the more threads can be executed concurrently. The Power 570 server has two dual-core POWER6 processors with simultaneous multi-threading (SMT) enabled, so that's eight logical processors working in parallel. To ensure that we have enough available threads in the thread pools, we increase the maximum number of threads in the default thread pool to 200 (default value is 20) and in the Web Container thread pool to 100 (default value is 50).

To further optimize the performance of Web service mediation scenarios in our tests, we turn off all performance monitoring, tracing, and logging. These are often necessary when setting up a server or when debugging problems or issues, but they do introduce some performance overhead. As a result, it is recommended that tracing and monitoring be used judiciously, and whenever possible, turned off entirely to ensure optimal performance.

Disabling the WebSphere Performance Monitoring Infrastructure (PMI) can be done through the WAS Administration Console as follows:

1. Go to **Monitoring and Tuning → Performance Monitoring Infrastructure (PMI) →** *server name*.

2. Clear the *Enable Performance Monitoring Infrastructure (PMI)* box, and in the *Currently Monitored Statistic Set* box, select *None*.

*Figure 4.2.1* shows the cumulative impact of increasing the maximum number of persistent HTTP connections, the maximum number of threads in the default and Web Container thread pools, as well as disabling PMI. These tweaks provide the most performance gain for the Transform Value mediation scenario – a 6% gain over what we are able to achieve with optimal JVM heap configuration and huge page support. The Transform Value mediation scenario is the least compute-intensive of all scenarios considered in our study, and therefore, it can sustain relatively high service request rates, so setting up for maximum concurrency resulted in a sizable performance boost. In contrast, the other transformation mediation scenarios are much more compute-intensive and have relatively low service request rates, so tuning for maximum concurrency does not help much. Similar observations can be made on the x3850 M2 server.

- **Impact of large payload sizes**. Let us now look at the performance of the Web service mediation scenarios with payload (message) sizes greater than 3 KB. *Figures 4.2.2* through *4.2.6* show the impact of the tuning items on these mediation scenarios with payload sizes greater than 3 KB.

As payload size increases, the Web service rates (throughput) drop, as expected. The size of the drop in throughput depends on the amount of processing that is required to perform the mediation on the payload in the service requests and responses. For example, *Figure 4.2.2* shows that the Composite Module mediation scenario posts approximately the same throughput for both {Request = 3 KB, Response = 10 KB} and {Request = 10 KB, Response = 3 KB} configurations because the same amount of mediation processing is done on both requests and responses. On the other hand, the fact that the Transform Value mediation scenario posted approximately the same throughput for both {Request = 10 KB, Response = 3 KB} and {Request = 10 KB, Response = 10 KB} configurations, as shown in *Figure 4.2.3*, indicates that the Transform Value mediation is only performed on requests (not responses) because the difference in the response payload size does not affect throughput. The relatively high throughput for the Transform Value and Route-On-Body mediation scenarios indicates that the amount of processing required to perform these mediations is less than the processing required by the other mediations.

It is important to note that the default WebSphere settings do not have enough capacity to even support payload sizes greater than 3 KB (all scenarios would fail after some time). Setting the initial and maximum JVM heap sizes to 1024 MB is the first step that allows all mediation scenarios to complete successfully. Indeed, the JVM heap optimizations discussed earlier in this section account for the largest performance improvement across all mediation scenarios – regardless of payload sizes.

For many mediation scenarios, such as Composite Module and Transform Schema, the Linux huge page support provides larger performance gains at payload sizes greater than 3 KB. This is expected because performing transformations on payloads larger than 3 KB would cross normal 4 KB page boundaries more frequently; therefore, huge pages would help in these cases.

Concurrency tuning (which includes setting the maximum persistent HTTP connections to *unlimited*, increasing the sizes of the default and Web Container thread pools, as well as disabling all performance monitoring, tracing, and logging) provides larger benefits at higher throughput. In fact, based on the data in *Figures 4.2.2* through *4.2.5*, it appears that concurrency tuning only provides noticeable performance gains at service request rates higher than 600 requests per second.

*Figure 4.2.7* shows the optimized performance for transformation mediation scenarios at different payload sizes. As the payload size increases, the Web service request rate drops due to larger computational requirements for performing transformation and routing mediations on larger payloads. In fact, as we increase the payload size to 100 KB for both requests and responses, the eight processor cores on the Power 570 server start to become the limiting factor: the CPU utilization approaches 99%, preventing us from obtaining higher throughput.

Similar observations can also be made on the x3850 M2 server – with the exception of the Transform Value mediation scenario. The Transform Value scenario requires the least amount of computational power because it is the simplest among the mediation scenarios considered in our study; therefore, it can sustain relatively high service request rates. On the x3850 M2, with its 16 processor cores, the network actually becomes the performance limiting factor *before* the CPUs in the Transform Value scenario with 100-KB payloads: the network traffic between the client and the server is measured approaching 500 Mbps, limiting the CPU utilization on the x3850 M2 to less than 80%.

**Summary**: On the Power 570 server, the default WebSphere and Linux settings can only support 3 KB payloads in the service requests and responses for the Web service mediation scenarios considered in our study. Setting the WAS JVM heap size to 3072 MB and using the Generational Concurrent (**gencon**) garbage collection policy yield more than 600% performance improvement. The Linux huge page support provides another 8% performance boost for routing mediations, and the concurrency tuning yields an average of 6% performance improvement for all mediation scenarios across different payload sizes. However, as we increase the payload size to 100 KB, the CPU utilization on the Power 570 server approaches 99%, so the server CPUs become the limiting factor, preventing us from obtaining higher throughput. Similar observations can be made for the x3850 M2 server – except for the Transform Value mediation scenario where the network actually becomes the performance bottleneck before the CPUs as the payload size is increased to 100 KB.
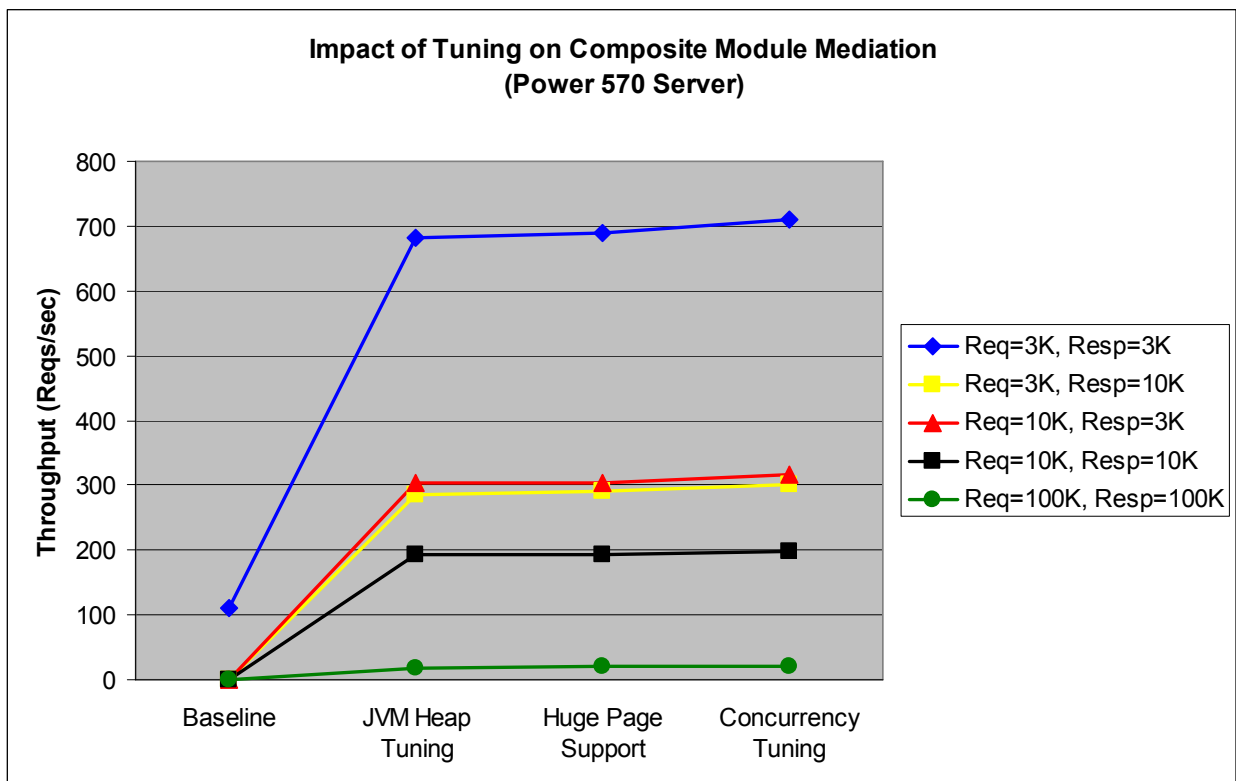


*Figure 4.2.2 – Impact of tuning on Composite Module mediation at different payload sizes*
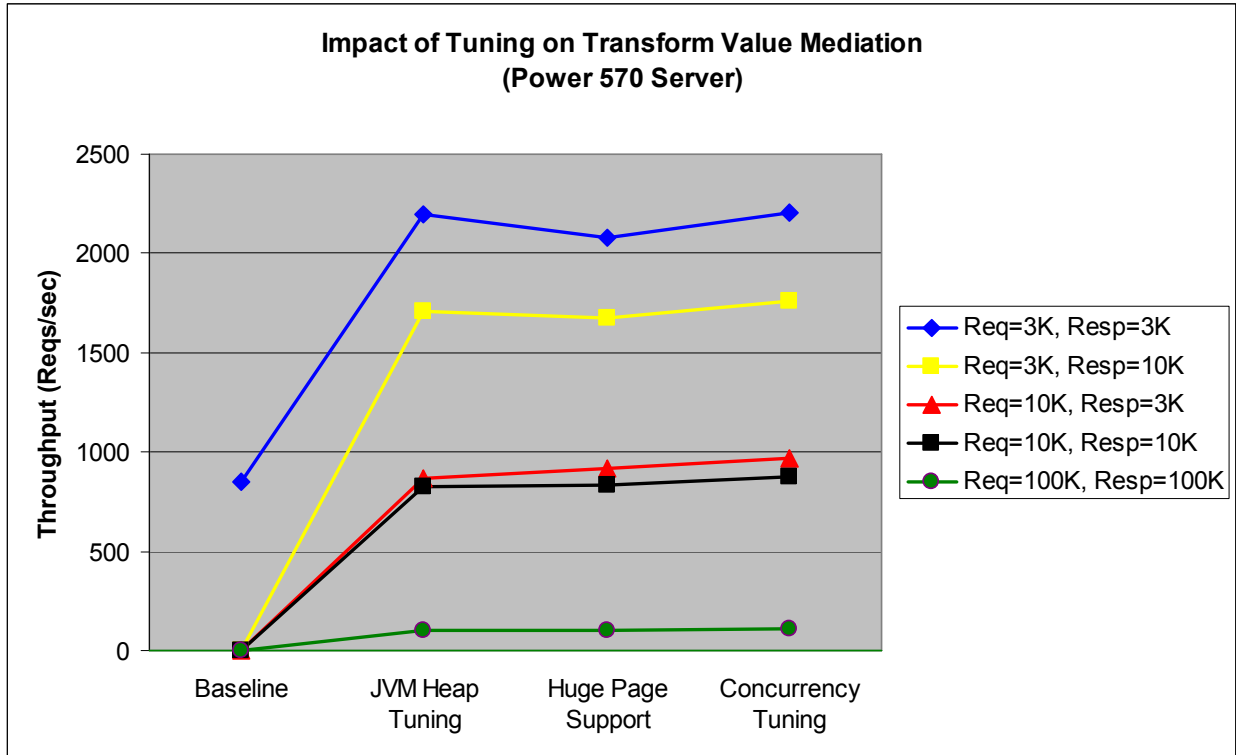
**Impact of Tuning on Transform Value Mediation
(Power 570 Server)**

*Figure 4.2.3 – Impact of tuning on Transform Value mediation at different payload sizes*

**Impact of Tuning on Transform Namespace Mediation
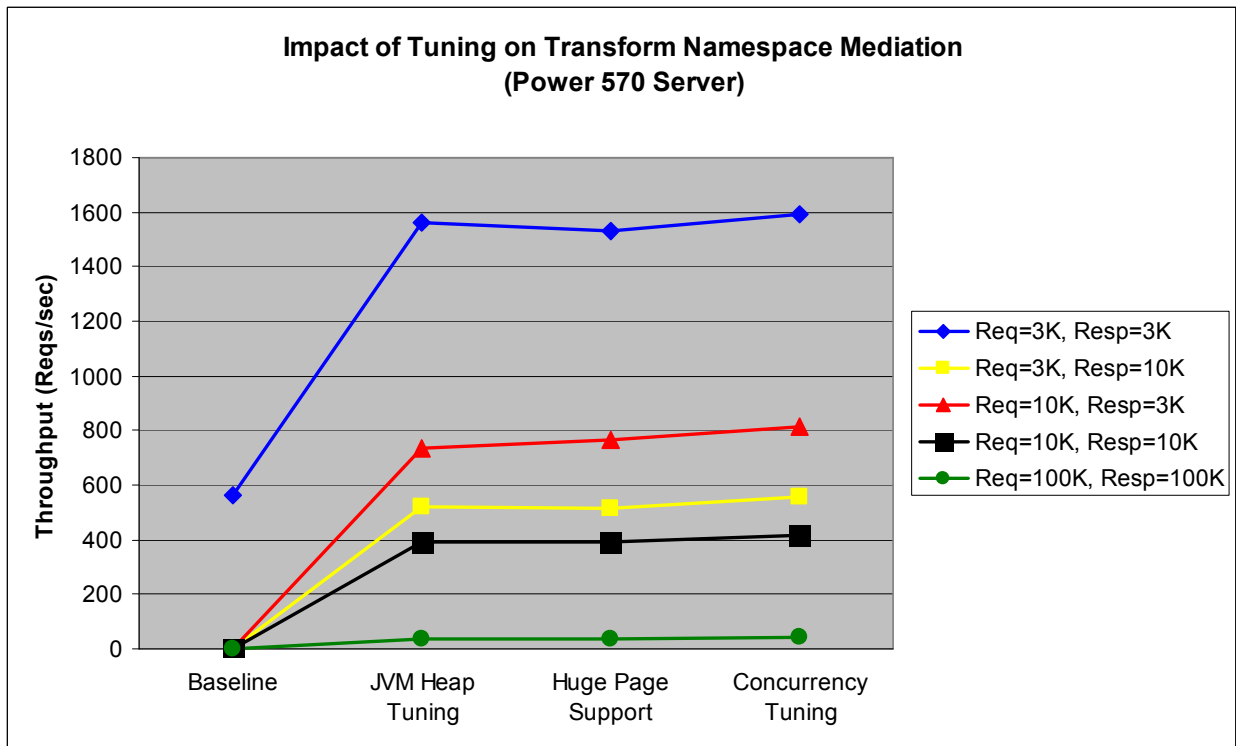(Power 570 Server)**

*Figure 4.2.4 – Impact of tuning on Transform Namespace mediation at different payload sizes*
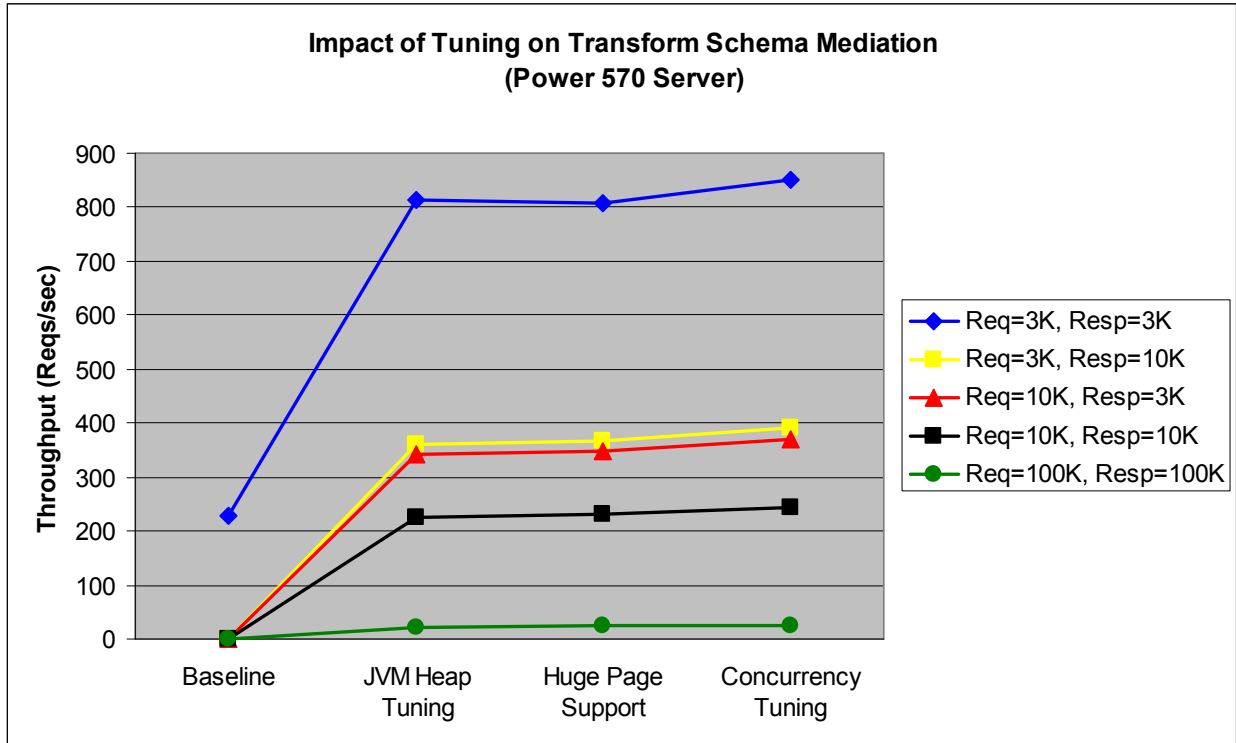
*Figure 4.2.5 – Impact of tuning on Transform Schema mediation at different payload sizes*
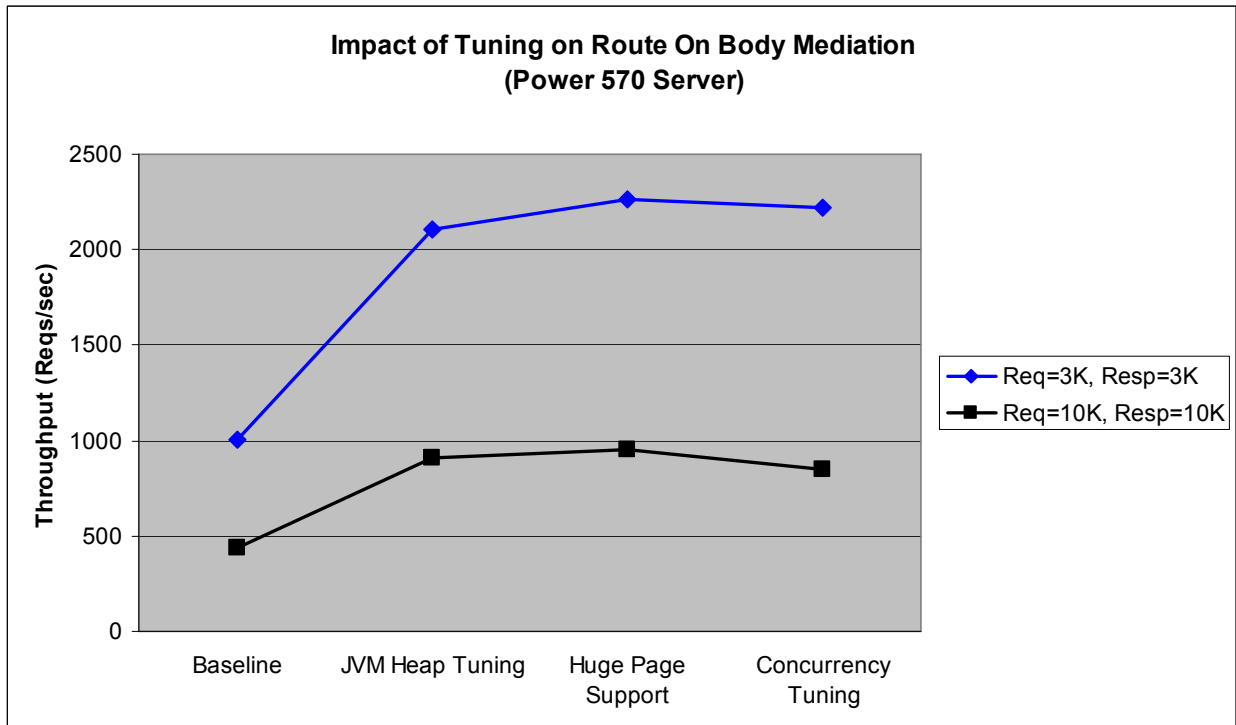


*Figure 4.2.6 – Impact of tuning on Route On Body mediation at different payload sizes*

**Optimized Performance of Transform Mediations
(Power 570 Server, 50 Client Threads, 20 Max Transactions)**

Figure showing throughput (Reqs/sec) on y-axis (0 to 2500) vs Request - Response Payload Size on x-axis (3K, 3K; 3K, 10K; 10K, 3K; 10K, 10K; 100K, 100K). Legend: Composite Module, Transform Value, Transform Namespace, Transform Schema.
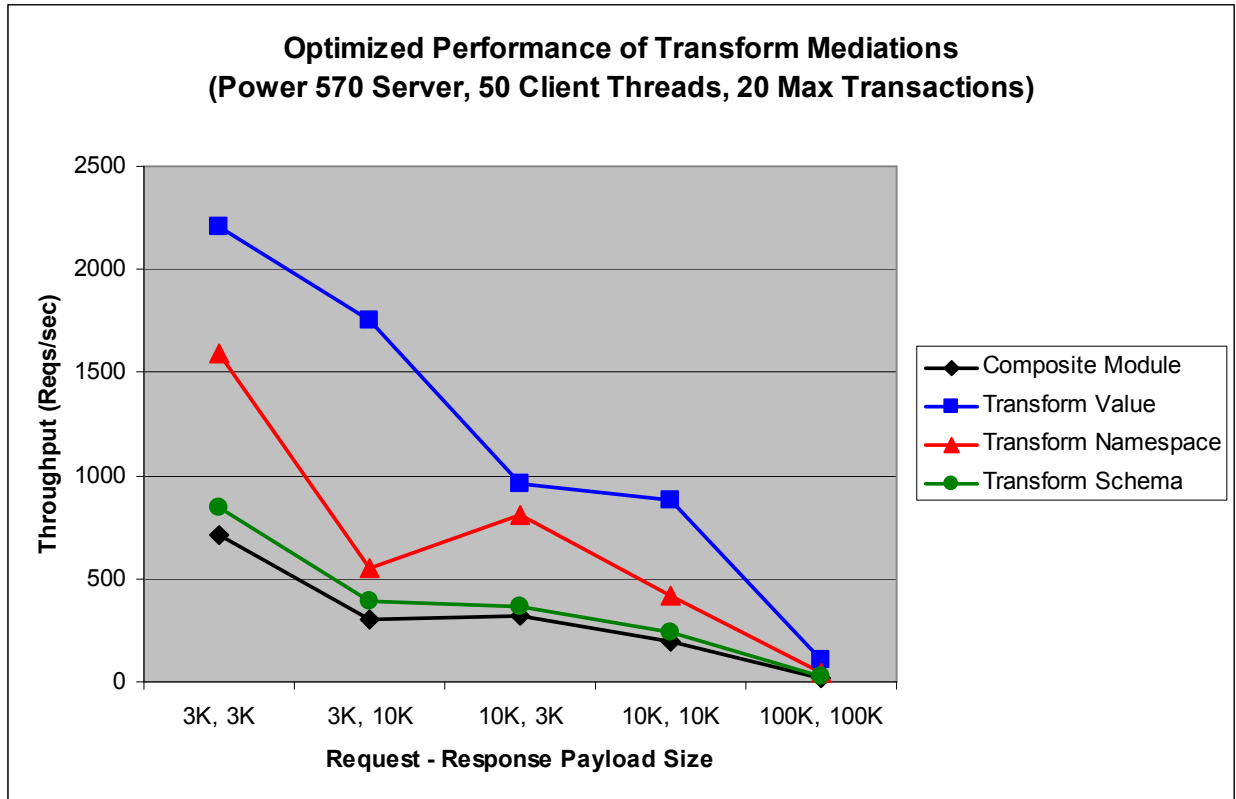
*Figure 4.2.7 – Optimized performance for transformation mediation scenarios across different payload sizes*

## 4.3. Tuning business process management (no clustering)

First, let us consider the setup illustrated in *Figure 4.3.1* where everything runs on a single IBM Power 570 server. In this setup, both WPS and DB2 are installed on an IBM Power 570 with dual POWER6 processors (a total of four *Processing Units* or cores). Both *HandleClaim* and *ClaimServices* applications are also installed and executed on this server. There is a single disk where the BPEDB, SIBDB, and all WPS logs for transaction and compensation services reside. The data for this configuration is shown in the first two rows in *Table 4.3.1*, under *Configurations 1* and *1.1*.
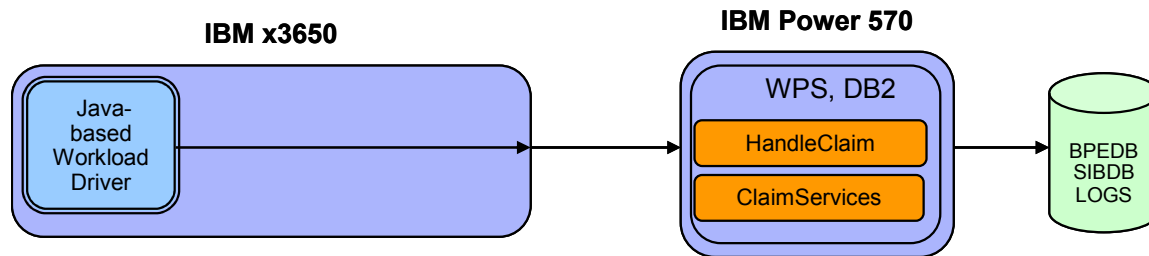
**IBM x3650**      **IBM Power 570**

Java-based Workload Driver → WPS, DB2 [HandleClaim, ClaimServices] → BPEDB SIBDB LOGS

*Figure 4.3.1 – Single server setup (Configurations 1 and 1.1)*

| Config | Max Tx | BTPS | CPU Utilization | | | DB2 Databases | | | | WPS Logs | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | Single Server | WPS | DB2 | Single Disk | BPEDB | BPE Log | SIBDB | Single Disk | Transaction | Compensation |
| 1 | 80 | Error | N/A | N/A | N/A | N/A | N/A | N/A | N/A | N/A | N/A | N/A |
| 1.1 | 80 | 2.59 | 53% | N/A | N/A | 99.60% | N/A | N/A | N/A | N/A | N/A | N/A |
| 2 | 80 | 21.7 | N/A | N/A | N/A | N/A | N/A | N/A | N/A | N/A | N/A | N/A |
| 2.1 | 80 | 45.7 | N/A | N/A | N/A | N/A | N/A | N/A | N/A | N/A | N/A | N/A |
| 2.2 | 80 | 46.28 | N/A | 97% | N/A | N/A | N/A | N/A | N/A | N/A | 67% | 60% |
| 3 | 80 | 61.12 | N/A | 93% | 30% | N/A | 30% | 13% | 12% | N/A | 79% | 75% |
| 3.1 | 80 | 55.84 | N/A | 89% | 26% | N/A | 47% | 11% | 13% | N/A | 81% | 75% |
| 3.2 | 80 | 44.56 | N/A | 78% | 30% | N/A | 43% | 10% | 9% | N/A | 70% | 60% |
| 3.3 | 80 | 41.79 | N/A | 75% | 30% | N/A | 70% | N/A | 73% | N/A | 63% | 66% |

*Table 4.3.1 – Tuning business process management (non-cluster)*

As you can see in *Table 4.3.1*:

- **Default configuration:** *Configuration 1* shows the out-of-the-box configuration where there is no tuning.

  *Result*:  In this configuration, the benchmark cannot even run.  The default configuration for the WAS JVM is too small to handle the necessary processing for the benchmark used in our study.

- **JVM heap tuning:** *Configuration 1.1* shows what happens when we tune the WAS JVM heap configuration.  Two basic JVM heap parameters are the size of the heap and the garbage collection policy.  Several tools are available, such as the Tivoli Performance Viewer (included with WebSphere), that can help analyze and monitor the heap usage and garbage collection so that the heap can be specifically tuned for a particular workload. In our tests, by looking at the number of memory pages allocated to the heap that are actually in use while the benchmark scenarios are being run, we find that the heap usage is usually at 2048 MB.  As a result, we set the JVM heap size for WAS at 2048 MB.  Because the Power 570 server has 32 GB of physical memory, which is more than adequate for our tests, we set the minimum and maximum heap size to the same value (2048 MB) to avoid the overhead of frequent heap size changes.

  In addition to the JVM heap size, the garbage collection policy can also affect performance.  The WAS JVM supports four different garbage collection policies.  The default garbage collection policy is `optthruput`.  However, we decided to use the `gencon` (Generation Concurrent) garbage collection policy, which handles short-lived objects differently from long-lived objects, because our earlier testing for Web service scenarios showed that the `gencon` policy was more suitable for Web service scenarios [12].  Under the `gencon` policy, the heap is split into new and old segments. Long-lived objects are promoted to the old space while short-lived objects are garbage collected quickly in the new space (called a *nursery*).  We also set the size of the nursery to 1536 MB (75% of the total heap size) through the *Integrated Solution Console (ISC):*

  1. Go to **Servers → Application Servers → *server name* → Server Infrastructure → Java and Process Management → Process Definition → Additional Properties → Java Virtual Machine.**
  2. Set the *Initial Heap Size* and *Maximum Heap Size* fields to 2048 MB.
  3. Enter `- Xgcpolicy:gencon –Xmn1536M` in the *Generic JVM arguments* field.

  *Result*:  Tuning the WAS JVM heap configuration allows the benchmark to run, but its overall throughput is only 2.59 business transactions per second (BTPS).  The data in *Table 4.3.1* for *Configuration 1.1* shows that the CPU utilization for the IBM Power 570 server is 53% while the disk utilization for the single disk is 99.6%.  This single disk is obviously the system performance bottleneck.

- **Adding another server for DB2:** *Configuration 2* in *Table 4.3.1* shows the result of our effort to remove the single disk as the system performance bottleneck.  First, we use another server – the IBM x3850 M2 – as the DB2 server and move the BPEDB and SIBDB databases over to the disk

arrays attached to the x3850 M2.  This configuration is illustrated in *Figure 4.3.2*.  More specifically, we partition the workload as follows:

- o The x3850 M2 server runs the *HumanTaskSimulator*, DB2, and hosts the databases on its disk arrays:
    - ▪ BPEDB on a 24-disk array attached to the IBM System Storage™ DS3400 controller
    - ▪ BPE logs on a 12-disk array attached to the IBM ServeRAID MR10M controller
    - ▪ SIBDB on a 12-disk array attached to the same IBM ServeRAID MR10M controller

- o The Power 570 server now only runs WPS and the benchmark's *HandleClaim* and *ClaimServices* applications.  We also make some additional improvements on this server's configuration:
    - ▪ One disk for transaction logs only
    - ▪ Another separate disk for compensation (recovery) logs only
    - ▪ The WAS JVM heap size increased from 2048 MB to 4096 MB with the nursery size set to 3072 MB (75% of the heap size)

**Result**:  With these changes, the overall benchmark throughput is now 21.7 BTPS – a significant improvement over 2.59 BTPS for *Configuration 1.1*.
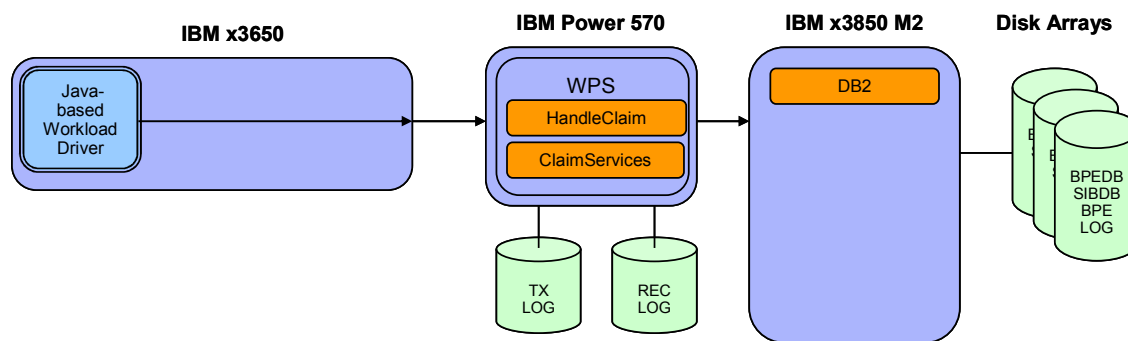


*Figure 4.3.2 – Separate WPS and DB2 servers (Configurations 2, 2.1, 2.2, and 3)*

- **WAS/WPS concurrency optimization:**  *Configuration 2.1* in *Table 4.3.1* shows the performance impact of additional WAS and WPS tuning that maximizes concurrency.  These are recommended by the WebSphere Performance Teams [13].  In particular, since the Power 570 server has a total of four cores or eight logical CPUs with simultaneous multi-threading (SMT), we follow the recommendations for eight CPUs in setting the following parameters:
    - o Thread pools:
        - ▪ Default thread pool max = 200
        - ▪ Web container's thread pool max = 100
    - o Connection pools:
        - ▪ BPEDB data source's connection pool max = 150
        - ▪ CEI ME data source's connection pool max = 80
        - ▪ SIBDB system data source's connection pool max = 30
        - ▪ SIBDB BPC data source's connection pool max = 50
    - o J2C connection factories:
        - ▪ BPECF connection pool max = 40
        - ▪ BPECFC connection pool max = 40
        - ▪ HTMCF connection pool max = 20
    - o J2C activation specifications:
        - ▪ BPEInternalActivationSpec → Custom properties → maxConcurrency = 60
        - ▪ <Benchmark> → Custom properties → maxConcurrency = 40

*Result*:  With the concurrency optimizations, we essentially double the benchmark's overall throughput to 45.7 BTPS (as compared to 21.7 BTPS for *Configuration 2*).

- **DB2 tuning:**  *Configuration 2.2* in *Table 4.3.1* shows the performance impact of applying DB2 tunings to optimize the log data rates.  We set the following DB2 parameters:
    - o  Max storage for lock list (4 KB) = 400
    - o  Log buffer size (4 KB) = 512
    - o  Log file size (4 KB) = 8000
    - o  Number of primary log files = 10
    - o  Number of secondary log files = 10
    - o  Maximum number of active applications = 250
    - o  Average number of active applications = 50
    - o  Percent log file reclaimed before soft checkpoint = 300
    - o  Percentage of lock lists per application = 20
    - o  Buffer pool size (pages) = 64000

  *Result*:  As can be seen in *Table 4.3.1*, tuning DB2 delivers a benchmark throughput rate of 46.28 BTPS – only a small improvement (1.3%) over what we are able to get for *Configuration 2.1*.  *Table 4.3.1* also shows that this *Configuration 2.2* has the CPU utilization of the WPS server (Power 570) at 97% and the disk utilization at 67% and 60%, respectively, for transaction and compensation (recovery) logs.  This indicates that the system performance bottleneck is no longer the disk subsystem, but rather, the Power 570's two dual-core POWER6 processors.  Adding more processors to the WPS Server would certainly increase the benchmark throughput – provided that some J2C Connection Factories and Activation Specifications would also be increased to accommodate the higher number of processors as recommended by the WebSphere Performance Team [13].

- **Cache mirroring in the IBM System Storage DS3400**:  *Configurations 3* and *3.1* in *Table 4.3.1* show the performance impact of the cache mirroring capability in the DS3400.  Starting with *Configuration 3*, we also use a more powerful Power 570 server.  Cache mirroring, which is the default setting on the DS3400, can remain enabled for reliability or availability reasons.  In our setup, the BPEDB disk array is attached to the DS3400 controller.  In *Configuration 3*, in order to achieve higher benchmark throughput, we disable cache mirroring.  In *Configuration 3.1*, we leave the cache mirroring enabled (the default setting).

  *Result*:  As we can see in *Table 4.3.1*, with cache mirroring enabled, the disk array utilization for BPEDB jumps from 30% to 47%, and the benchmark throughput drops from 61.12 BTPS to 55.84 BTPS – an 8.6% performance drop.  This is a classic trade-off between performance and reliability/availability.

- **Reducing WAS JVM heap size in memory-constrained environment**:  *Configuration 3.2* in *Table 4.3.1* shows the impact of reducing the WAS JVM heap size from 4096 MB to 2048 MB.  This could be necessary in memory-constrained environments.

  *Result*:  Reducing the WAS JVM heap from 4096 MB to 2048 MB drops the benchmark's overall throughput from 55.84 BTPS to 44.56 BTPS, a significant 20% performance drop.  As a result, it is highly recommended that the WAS JVM heap be adjusted accordingly after adding CPUs to the WPS server.

- **Write-back caching in IBM ServeRAID MR10M controller**:  *Configuration 3.3* in *Table 4.3.1* shows what happens when we use write-through caching policy in the MR10M disk array controller (instead of write-back).  Write-back caching is always good for the performance of disk writes because the disk write operations are considered done as soon as the data is written to the cache (instead of physical disks).  Data in the cache would then be written ("flushed") to physical disks at a later time.  However, in an environment where there is no battery backup for the controller or for the disk subsystem, write-through caching could help protect the integrity of data

when there is a power outage.  In our setup, the SIBDB and BPE log disk arrays are attached to the MR10M controller.

**_Result_**:  Using the write-though caching policy for the SIBDB disk array results in a 6% performance drop – from 44.56 BTPS with write-back caching in *Configure 3.2* to 41.79 BTPS with write-through caching in this *Configuration 3.3*.  Note that, for *Configuration 3.3* in *Table 4.3.1*, the utilization of the disk array for BPEDB jumps to 70% because we now use a single 24-disk array for both BPEDB and BPE logs.  However, our test data (not shown in *Table 4.3.1*) indicates that the 70% utilization of the disk array does not have any noticeable impact on the benchmark throughput.  This is because, even at 70% utilization, the disk I/O queue depth for the 24-disk array is still less than 1, indicating that the disk array is not the performance bottleneck.  In fact, this is lower than the CPU utilization (75%) for the WPS server.

- **Linux logical disk striping**:  In *Configurations 3* and *3.1*, the utilization for the disks holding the transaction log and compensation (recovery) log on the WPS server is quite high – around 80% for transaction log and 75% for compensation log.  While this is still lower than the CPU utilization, it would be interesting to try to lower the disk utilization and see if that would have any impact on the benchmark's overall throughput.  We do not have any extra disk array controller that could be installed in the WPS server, but we have two extra disks that could be used to set up a four-disk logical array through the Linux Logical Volume Manager (LVM) to handle both transaction and compensation log traffic.

  **_Result_**:  *Table 4.3.2* shows the results.  For a four-disk logical array, the stripe size of 64 KB provides better performance than smaller stripe sizes of 4 KB and 16 KB, given that the size of each log data write is between 4 KB and 5 KB.  The utilization of the device manager is found to be over 90% while the utilization of each individual disk ranges from 27% to 36%.  However, dedicating a single disk to each log, without any logical disk striping, produces the best benchmark throughput, even though we use only two disks and each disk is used more than 75% of the time.  This indicates that the overhead of the Linux logical volume manager in managing a logical disk array has a negative performance impact when it comes to transaction and compensation logs on the WPS server.  As a result, for the rest of our study, we dedicate a single physical disk for each WPS log.

| Disk Setup | Stripe Size (KB) | Benchmark Throughput (BTPS) |
|---|---|---|
| Two Single Disks | N/A | 55.84 |
| 4-Disk Logical Array | 4 | 44.79 |
| | 16 | 49.79 |
| | 64 | 51.38 |

*Table 4.1.2 – Logical disk striping (through Linux Logical Volume Manager)*

- **Linux large (huge) page support**:  We configure large (huge) pages (16 MB page size) on the Power 570 server (WPS server) for the WAS JVM heap and find that it does not result in any performance improvement.  In fact, with large page support, the benchmark's overall throughput decreases from 61.12 BTPS (*Configuration 3*) to 59.70 BTPS, a performance degradation of about 2%.  As a result, for the remainder of this study, we do not configure large pages.

*Figure 4.3.3* shows the cumulative performance improvement as we go from the default (out-of-the-box) configuration to tuning DB2.  The tuning items are shown in a chronological order from left to right in the figure.

It should also be noted that the x3650 server that hosts the benchmark's Workload Driver does not have any impact on our performance results because the CPU utilization of this server is less than 5% and the utilization of the only disk used on this server is less than 6%.
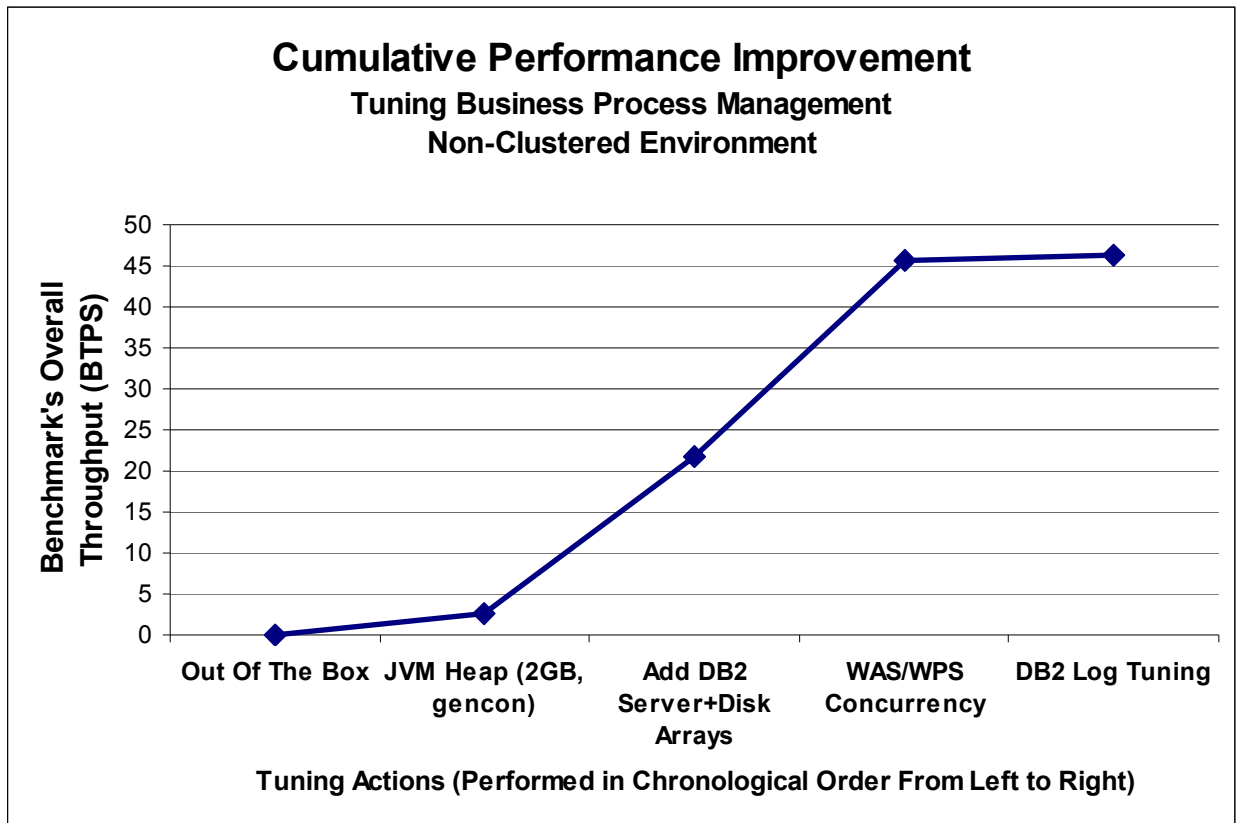
## Cumulative Performance Improvement
### Tuning Business Process Management
### Non-Clustered Environment

**Benchmark's Overall Throughput (BTPS)** (y-axis: 0, 5, 10, 15, 20, 25, 30, 35, 40, 45, 50)

x-axis categories: Out Of The Box, JVM Heap (2GB, gencon), Add DB2 Server+Disk Arrays, WAS/WPS Concurrency, DB2 Log Tuning

**Tuning Actions (Performed in Chronological Order From Left to Right)**

*Figure 4.3.3 – Cumulative Impact of BPM Tuning Actions in Non-Clustered Environment*

## 4.4. Tuning dynamic operations with WebSphere Virtual Enterprise (WVE)

In this section, we evaluate a virtual, goals-driven, autonomic environment for SOA applications like the benchmark used in this study. We use WebSphere Virtual Enterprise (WVE) to create this environment, which is illustrated in *Figure 4.4.1.*

The dynamic cluster of WPS servers, called *application cluster* in *Figure 4.4.1*, consists of WPS servers on which the benchmark's applications are deployed. These applications include *HandleClaim* and *ClaimServices* applications, denoted as B and S, respectively. In response to an increase in workload demands or an on-going failure to meet a service policy's performance objective, WebSphere Virtual Enterprise's autonomic managers can start an instance of WPS server on the x3850 M2 server or on any of the two logical partitions (LPARs) on the OpenPower 720 server. Based on the results in *Section 4.3*, each instance of the WPS server has a single disk for transaction log and another single disk for compensation (recovery) log. In *Figure 4.4.1*, the LPARs are virtualized with PowerVM technology, so I/O operations to these (virtual) disks must go through the Virtual I/O Server (VIOS). The LPARs use shared, capped virtual CPUs. In general, it is recommended that we configure shared, uncapped virtual CPUs across the LPARs; however, in this study, the number of (virtual) CPUs that each LPAR is allowed to use is capped so that we can more accurately measure the workload impact on each LPAR and the performance impact of logical partitioning.

By default, there can only be a maximum of one WPS server instance per node in the dynamic application cluster.

The service policy in WVE only allows two types of performance goals to be specified – the average response time or the percentile response time goals. In this study, we use the average response time goal for simplicity. To set a reasonable response time goal, we examine the response times reported in

*Table 4.4.1* for all configurations that we evaluate in this study.  We note that the best response time that we are able to achieve is about 14 ms.  As a rule of thumb (best practices), the average response time goal should be at least two times higher than the best response time observed on a lightly loaded cluster.  Consequently, we set the average response time goal in the service policy to be 30 ms.  This should enable the autonomic managers to activate additional WPS servers (instances) when the average response time for claim requests exceeds 30 ms (the performance goal).  Setting the average response time goal too close to 14 ms would never result in additional servers to be started because the autonomic managers would determine that starting additional servers would not improve the chances of meeting the performance goal.

We specify the service policy to be applicable to HTTP and SOAP requests that are handled by the benchmark's applications.  We also assign the highest priority to this service policy.

We use the default settings for the On-Demand Router (ODR) and its associated autonomic managers.

As described in *Section 2.3*, there are two approaches to cluster the Message Engines (MEs):

- The Message Engines are local to the dynamic application cluster.
- The Message Engines are in their own cluster (called *ME Cluster*), separate from the dynamic application cluster.

In this section, we consider both approaches and see which one performs better.  In those configurations where the ME Cluster is configured, it runs on the x3850 M2 server.  As such, the ME Cluster is a static cluster.
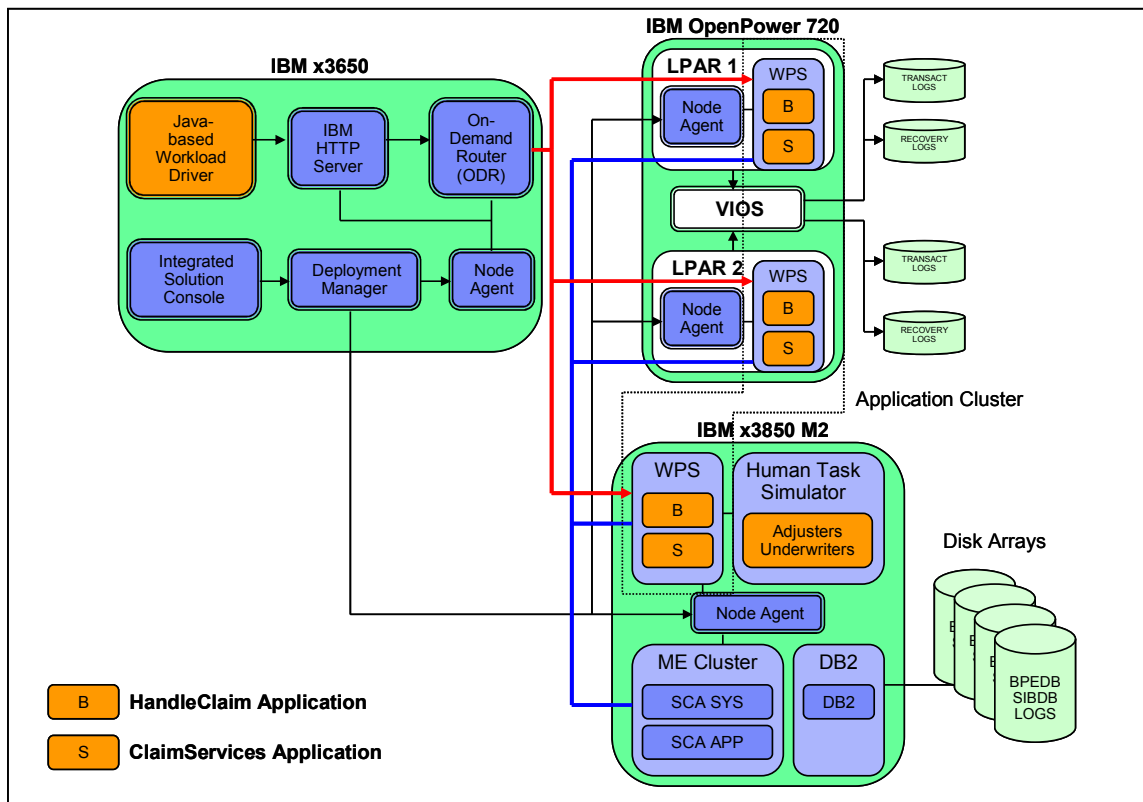


*Figure 4.4.1 – Dynamic application cluster with separate cluster for Message Engines*

In general, for the configurations considered in this section, the physical servers host the following components (see *Figure 4.4.1*):

- IBM x3650 server
    - Benchmark's Java Workload Driver
    - WebSphere Deployment Manager
    - IBM HTTP Server (Web Server)
    - WebSphere On-Demand Router (ODR)
    - WebSphere Node Agent

- IBM x3850 M2 server
    - DB2, hosting SIBDB and BPEDB databases (including their logs)
    - Benchmark's *HumanTaskSimulator*, simulating the actions of adjusters and underwriters
    - WPS server (initial instance – unless otherwise specified), hosting the benchmark's *HandleClaim* and *ClaimServices* applications
    - WebSphere Node Agent
    - ME Cluster (if configured)

- IBM OpenPower 720 server (VIOS with 0.5 POWER5 core, LPAR1 with 1.5 POWER5 cores, LPAR2 with 2.0 POWER5 cores)
    - Virtual I/O Server (VIOS)
    - LPAR1: WPS server (one instance maximum) and WebSphere Node Agent
    - LPAR2: WPS server (one instance maximum) and WebSphere Node Agent

On the x3850 M2 server, the BPEDB database is hosted on a 24-disk array managed by the IBM System Storage DS3400 controller. Because of the large amount of disk writes, the BPEDB log resides on a separate 12-disk array that is attached to the IBM ServeRAID MR10M adapter. The SIBDB database and its log are hosted on another 12-disk array that is attached to the same MR10M adapter. As in *Section 4.3*, each WPS server has two disks, one dedicated for the transaction log and one for the compensation (recovery) log. For optimal performance, cache mirroring is disabled for the DS3400 controller and the write-back caching policy is used for the MR10M controller. In addition, all of the WPS and DB2 tuning items, discussed in *Section 4.3,* are applied to the WVE environment.

For the remainder of this paper, we denote the maximum number of concurrent insurance claim requests generated by the benchmark's Workload Driver as *Max Tx* and the number of threads used to generate this traffic as simply *Threads*. We will look at two performance measurements: the number of *business transactions per second (BTPS),* as reported by the benchmark, and the *average response time* for service requests, as reported by WebSphere Virtual Enterprise.

Let us first consider the configurations with the Message Engines in a separate cluster (ME Cluster). The performance results for these configurations are shown in *Table 4.4.1*.

- **Single node performance (with separate ME Cluster):** *Configuration 5* in *Table 4.4.1* shows what can be achieved with a single node (the x3850 M2 server) in the dynamic cluster. The dynamic application cluster is put in *Manual Mode*, so that the autonomic managers do not attempt to activate additional WPS servers on the other nodes – even when the average response time exceeds the performance goal of 30 ms.

    *Result*: With the Workload Driver using two threads to generate a maximum of 60 concurrent requests (Max Tx = 60, Threads = 2), the x3850 M2 is able to handle an average of 28.29 BTPS. The CPU utilization on the x3850 M2 is 83% and the disk subsystem still has quite a bit of bandwidth available (the disk I/O queue depth is less than 0.60). The average response time for the claim requests ranges from 22.85 ms with a Max Tx of 20 to 52.74 ms with a Max Tx of 60. Setting the maximum number of concurrent requests higher than 60 introduces some instability: only a few runs are completed successfully as the CPU utilization moves above 90%.

| Config | Workload | | BTPS | CPU Utilization | | | Min Response Time (ms) | | | Max Throughput (BTPS) | | | Max Disk | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | Max Tx | Threads | | Node 1 | Node 2 | Node 3 | Node 1 | Node 2 | Node 3 | Node 1 | Node 2 | Node 3 | Queue Depth | Util |
| 5 | 20 | 1 | 18.23 | 49.60% | N/A | N/A | 22.85 | N/A | N/A | 20.03 | N/A | N/A | 0.29 | 19% |
| | 40 | 1 | 25.06 | 73.50% | N/A | N/A | 32.09 | N/A | N/A | 27.88 | N/A | N/A | 0.5 | 28% |
| | 60 | 1 | 25.62 | 75% | N/A | N/A | 37.27 | N/A | N/A | 28.6 | N/A | N/A | 0.55 | 30% |
| | 60 | 2 | 28.29 | 83% | N/A | N/A | 52.74 | N/A | N/A | 31.45 | N/A | N/A | 0.53 | 28% |
| 5.1 | 20 | 1 | 15.24 | 35% | 45% | 55% | 18.77 | 49.41 | 50.08 | 10.91 | 3.59 | 3.61 | 0.39 | 38% |
| | 40 | 1 | 20.72 | 44% | 72% | 78% | 16.02 | 57.64 | 55.61 | 14.08 | 4.91 | 5.08 | 0.55 | 53% |
| | 60 | 1 | 21.88 | 40% | 75% | 78% | 17.76 | 66.08 | 64.83 | 14.49 | 5.25 | 5.35 | 0.64 | 61% |
| | 60 | 2 | 26.02 | 58% | 81% | 85% | 19.22 | 82.99 | 81.4 | 17.04 | 5.98 | 6.15 | 0.64 | 61% |
| | 80 | 2 | 30.86 | 65% | 91% | 92% | 25.51 | 91.37 | 89.45 | 19.62 | 6.53 | 7.13 | 0.76 | 71% |
| | 80 | 3 | 34.29 | 75% | 95% | 95% | 35.3 | 103.62 | 102.78 | 24.04 | 6.49 | 6.87 | 0.67 | 68% |
| | 100 | 3 | 33.71 | 77% | 92% | 92% | 40.47 | 196.22 | 162.2 | 22.35 | 5.01 | 8 | 0.78 | 67% |
| | 160 | 4 | 35.56 | 86% | 92% | 93% | 82.34 | 263.48 | 212.12 | 26.99 | 4.83 | 6.67 | 1.04 | 70% |
| 5.2 | 160 | 4 | 38.73 | 86% | 72% | N/A | 73.05 | 121.23 | N/A | 28.68 | 12.58 | N/A | 1.24 | 50% |
| | 160 | 6 | 40.47 | 91% | 80% | N/A | 68.63 | 145.3 | N/A | 29.73 | 14.1 | N/A | 1.26 | 50% |
| 6 | 20 | 1 | 23.55 | 40% | N/A | N/A | 16.18 | N/A | N/A | N/A | N/A | N/A | 0.38 | 24% |
| | 40 | 1 | 37.36 | 64% | N/A | N/A | 20.28 | N/A | N/A | N/A | N/A | N/A | 0.66 | 39% |
| | 60 | 1 | 45.73 | 77% | N/A | N/A | 25.88 | N/A | N/A | N/A | N/A | N/A | 0.95 | 47% |
| | 60 | 2 | 46.55 | 80% | N/A | N/A | 30.38 | N/A | N/A | N/A | N/A | N/A | 0.96 | 48% |
| 6.1 | 20 | 1 | 9.4 | 13% | 92% | N/A | 14.97 | 70.41 | N/A | N/A | N/A | N/A | 0.43 | 40% |
| 6.2 | 20 | 1 | 11.27 | 13% | 92% | N/A | 14.68 | 54.82 | N/A | N/A | N/A | N/A | 0.43 | 42% |
| 6.3 | 20 | 1 | 15.41 | 15% | 55% | N/A | 13.6 | 37.46 | N/A | 5.22 | 10.38 | N/A | 0.51 | 50% |

*Table 4.4.1 – Performance data for dynamic cluster configurations*

- **Multi-node performance (with separate ME Cluster)**: *Configuration 5.1* in *Table 4.4.1* shows that we can achieve higher service rates when we put the dynamic application cluster in *Automatic Mode*. This allows the On-Demand Router's autonomic managers to activate additional servers on the OpenPower 720's LPARs automatically (without human intervention) to help handle the workload generated by the benchmark's Workload Driver. This configuration is illustrated in *Figure 4.4.1*.

  Initially, after we turn on the benchmark's Workload Driver (with Max Tx = 20, Threads =1), the initial WPS server running on the x3850 M2 takes a while to warm up. During this warm-up period, the average response time for service requests exceeds the service goal of 30 ms for more than 30 seconds, as shown in *Figure 4.4.2*. The autonomic managers sense this condition, and generate a runtime task to activate additional WPS servers on the cluster's remaining nodes. The runtime task appears on the System Administration View of the Integrated Solutions Console (ISC) at approximately 2 minutes after the start of the run. Since the application cluster is in Automatic Mode, the runtime task starts immediately without further human intervention after it appears on the ISC. It takes about 2 minutes for the new WPS servers to be activated on the two LPARs on the OpenPower 720 server. Finally, about 4 minutes after the start of the run, the WPS servers on the LPARs start running. Of course, these new servers need time to warm up and that is the reason for the initial high response times recorded on these servers.

  In the mean time, as the new WPS servers are starting on the LPARs, their Message Engines are starting to run in the ME cluster on the x3850 M2, causing the average response time on the x3850 M2 server to jump higher as well. However, after about 90 seconds, the average response time on the x3850 M2 starts to stabilize and falls to about 18.77 ms – well below the service goal of 30 ms and better than the best average response time in the single-node configuration (*Configuration 5*). This shows the benefit of offloading the workload to new additional servers in the dynamic cluster. However, on the OpenPower 720, the LPARs (with 1.5 POWER5 cores for LPAR1 and 2.0 POWER5 cores for LPAR2) that host the new servers are not powerful enough to achieve a response time of less than 30 ms for the service requests sent to them. Because of this, the maximum throughput data collected at each node indicates that the On-Demand Router routes more service requests to the more powerful x3850 M2 server than to the two POWER5 LPARs.
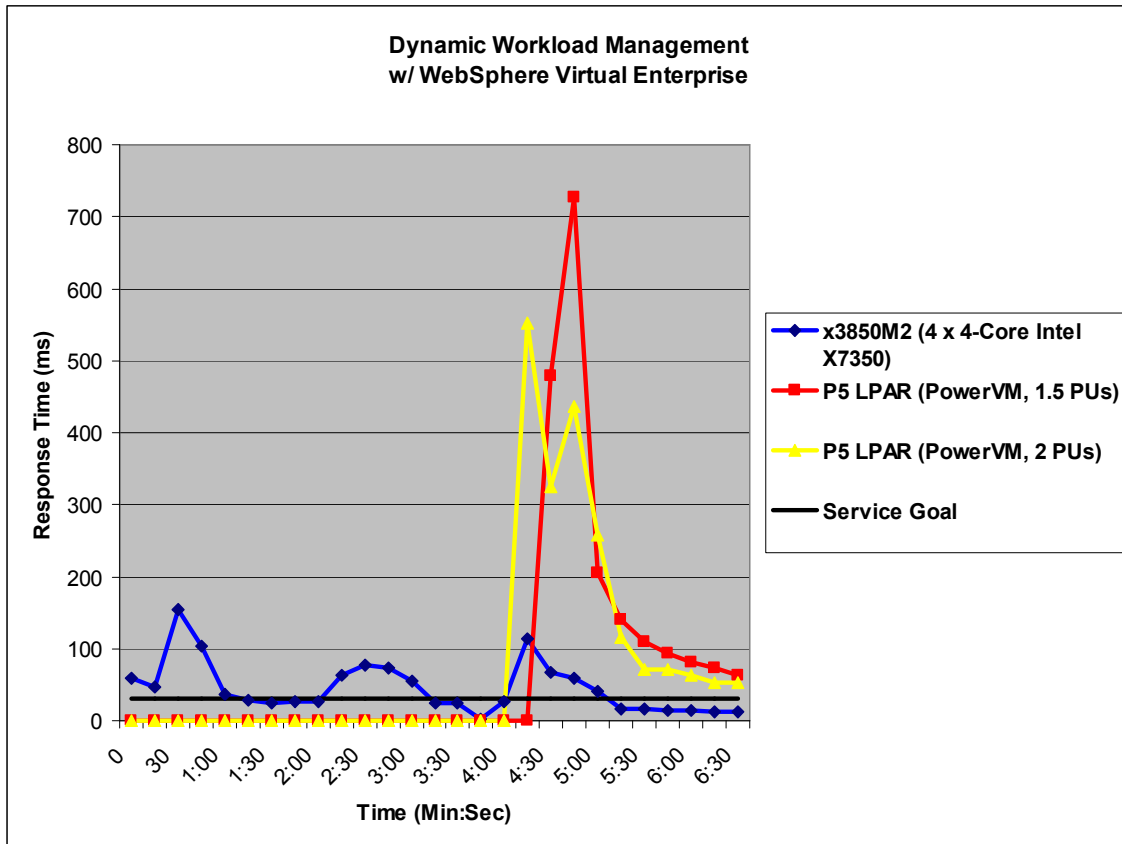
**Dynamic Workload Management
w/ WebSphere Virtual Enterprise**

Figure 4.4.2 – *WebSphere Virtual Enterprise's autonomic managers activating additional WPS servers*

After everything settles down, the best benchmark throughput that we can get for this multi-node configuration is 35.56 BTPS, a 25% improvement over the single-node configuration. In addition, as can be seen in *Table 4.4.1*, the multi-node configuration provides more stability at much higher workloads (up to Max Tx = 160, Threads = 4) than the single-node configuration. At the highest workloads (Max Tx = 160, Threads = 4), the CPU utilization is 92%, 93%, and 86% for LPAR1, LPAR2, and x3850 M2 servers, respectively, indicating that we are running out of CPU bandwidth on the LPARs. The disk subsystem is still okay, with the maximum I/O queue depth of 1.04 and the disk with the highest utilization (70%) being the compensation log disk in LPAR2.

*Figure 4.4.3* shows the throughput that is contributed by each node in the dynamic cluster. As the workload increases, the throughput at the LPARs begins to level off, running out of CPU bandwidth. However, with 16 processor cores, the x3850 M2 just keeps going while the ODR sends more requests to it.

- **Multi-node performance (with separate ME Cluster and without PowerVM)**: *Configuration 5.2* in *Table 4.4.1* shows the benchmark throughput when we effectively remove the PowerVM virtualization layer from the OpenPower 720 server. In the previous configuration (*Configuration 5.1)*, because we are running out of CPU bandwidth and the disk utilization is high on the LPARs with Max Tx = 160 and Threads = 4, removing the virtualization layer might result in higher benchmark throughput. To do this, we only configure a single LPAR on the OpenPower 720 server, assign all CPUs (4 cores) and memory to this single LPAR, and remove the Virtual I/O Server (VIOS). All I/O operations now go directly to the disk subsystem without going through the VIOS, reducing the I/O latency. This configuration is illustrated in *Figure 4.4.4.*
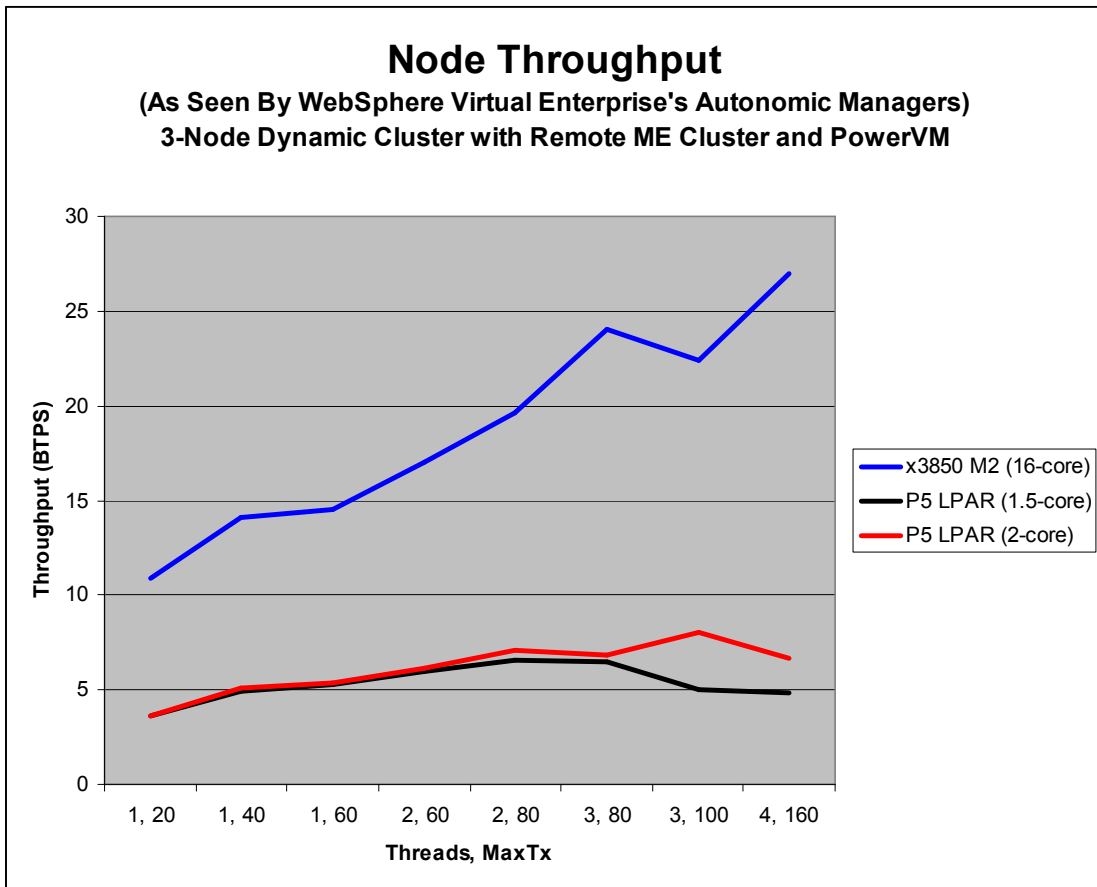
**Node Throughput**

**(As Seen By WebSphere Virtual Enterprise's Autonomic Managers)**

**3-Node Dynamic Cluster with Remote ME Cluster and PowerVM**

*Figure 4.4.3 – Node throughput as reported by WebSphere Virtual Enterprise*

**Result**: As can be seen in *Table 4.4.1*, this configuration is now able to support up to 6 threads with MaxTx = 160, and the overall benchmark throughput is now 40.47 BTPS, a 14% improvement over *Configuration 5.1*. The CPU utilization on the x3850 M2 server now moves above 90%, but the disk subsystem still has available bandwidth because the disk I/O queue depth is 1.26 and the highest disk utilization is only 50%. If we look at the data for MaxTx = 160, Threads = 4 and compare it against the data for the same workload in the previous configuration, we see that the average response time on the single 4-core LPAR is less than half the response times observed on the 1.5-core LPAR1 and 2-core LPAR2 in the previous configuration. While the maximum throughput observed at the single 4-core LPAR is about 10% higher than the sum of maximum throughput for both LPAR1 and LPAR2 in the previous configuration, it is interesting to note that the performance of the WPS server running on the x3850 M2 is also better (that is, lower average response time, higher maximum throughput). This indicates that a single 4-core LPAR can offload more workload from the x3850 M2 than both the 1.5-core LPAR1 and 2-core LPAR2 can in the previous configuration. Later on in this paper, we will measure the performance impact of the VIOS more accurately.

- **Single-node performance (with local Message Engines)**: *Configuration 6* in *Table 4.4.1* shows the performance data when using only a single node (x3850 M2) with local Message Engines running in the same dynamic cluster as the WPS servers.

  **Result**: The data indicates that, for this single-node configuration with local Message Engines, the benchmark throughput is much higher, and the average response time is much lower, than in *Configuration 5* where we have remote Message Engines located in a separate cluster. Depending on the workload level, the extra latency in going to Message Engines in a remote cluster results in 30% - 60% drop in the benchmark throughput and 25% - 40% jump in the average service response time. Consequently, having the Message Engines running locally in

the same cluster as the applications is faster in the single-node configuration, but it severely limits the scalability of a multi-node configuration, a fact that is very obvious when we consider the next configuration.
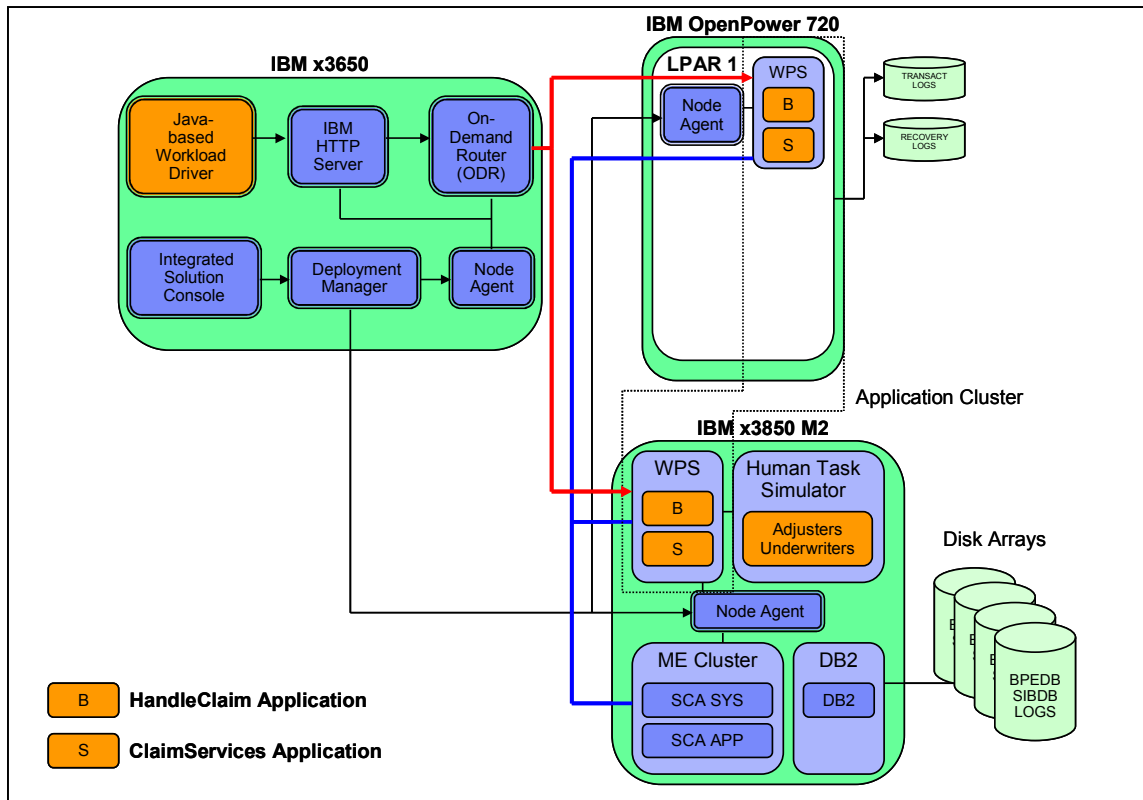


*Figure 4.4.4 – Dynamic application cluster with separate ME Cluster but without PowerVM*

- **Multi-node performance (with local Message Engines)**: *Configuration 6.1* in *Table 4.4.1* shows the performance data obtained for a two-node application cluster with local Message Engines. This application cluster is illustrated in *Figure 4.4.5*. When we start the Workload Driver, the initial WPS server runs on a two-core POWER5 LPAR. As the CPU utilization on this LPAR quickly rises above 90%, and the average service response time is far above the service goal of 30 ms, the autonomic managers generate a runtime task to activate the remaining node (the x3850 M2) in the cluster. However, even after a new WPS server is started on the x3850 M2, most (approximately 80%) of the workload still goes to the initial node (the two-core POWER5 LPAR), leaving the second node (the x3850 M2) under-utilized – the CPU utilization on the x3850 M2 is only 13%. The benchmark's overall throughput (9.4 BTPS) is much smaller than in the case with remote Message Engines (35.56 BTPS). This is because of a design limitation for the Service Integration (SI) buses: if a Message-Driven Bean (MDB) has a local ME available, it is always forcibly "bound" to it, even if the ME is inactive. Because only one ME instance can be active at any given time, only one WPS server can have active instances of that MDB. In other words, all but one of the WPS servers in the cluster is essentially in stand-by mode. This imposes serious limitation on the overall scalability of the dynamic cluster if we have long-running business processes or asynchronous SCA invocations. As a result, for multi-node dynamic clusters, we recommend configuring the Message Engines in a separate cluster. The WebSphere clustering topologies are classified into three types [4]:
  - *Bronze* topology, where all components (WPS applications, Message Engines, CEI) are run in a single cluster
  - *Silver* topology, where there are two separate clusters: one for WPS applications and CEI, and one for the Message Engines

- *Gold* topology, where there are separate clusters for WPS applications, CEI, and Message Engines

Because the benchmark that is used in this study does not generate many events, the Silver topology is the most suitable topology. *Figure 4.4.6* shows the impact of messaging infrastructure clustering on the benchmark throughput.

In this *Configuration 6.1*, the WPS server running on the two-core POWER5 LPAR is the initial node; therefore, with local Message Engines, most of the traffic goes there and the benchmark throughput is only a meager 9.40 BTPS. What if we made the x3850 M2 the initial node? In this case, most of the traffic would be routed to the x3850 M2. Because the x3850 M2 is much more powerful than the two-core POWER5 LPAR, the benchmark's throughput is 34.62 BTPS – almost *four times higher* than in the case of the two-core POWER5 LPAR being the initial node (with the exact same settings for the Workload Driver). The CPU utilization is 65% and 9% on the x3850 M2 and the POWER5 LPAR, respectively, confirming that most of the traffic is indeed routed to the initial x3850 M2 node.

- **Virtual I/O Server (VIOS) impact**: We know that the VIOS introduces additional latency to disk I/O operations. In **Configuration 6.2**, we remove the VIOS partition; therefore, disk writes to the transaction and compensation logs no longer go through VIOS. At the same workload driver settings (Max Tx = 20, Threads = 1), the data in *Table 4.4.1* indicates that removing the VIOS reduces the average service response time from 70.41 ms to 54.82 ms, a 22% improvement over the configuration with VIOS. Let us take a closer look at the transaction and compensation log disks which are attached to the POWER5 LPARs. *Table 4.4.2* shows the disk performance statistics for *Configuration 6.1* (with VIOS) and *Configuration 6.2* (without VIOS). The number of disk writes and the amount of written log data are similar in both configurations, as expected. However, *without* VIOS, we are able to obtain:

  - Higher overall throughput for the benchmark (11.27 BTPS vs. 9.40 BTPS)
  - Lower disk I/O queue depth
  - Lower I/O service times
  - Low disk utilization (as seen from Linux)

However, it should be noted that the PowerVM virtualization technology also allows us to configure the log disks as *dedicated* I/O devices to the LPARs so that disk writes to those disks do not have to go through VIOS. We will look at the performance of the disk subsystem in more detail later in the paper.

- **Adding more CPUs (or using shared, uncapped LPAR mode)**: With the initial WPS server running on a two-core POWER5 LPAR, most of the workload traffic goes there, pushing the CPU utilization on the LPAR close to 100%. **Configuration 6.3** in *Table 4.4.1* shows what we are able to get when we assign more POWER5 cores to the LPAR. This is effectively the same as if we configure the LPAR to run in shared, uncapped mode where the LPAR could use the remaining (idle) processor cores on the OpenPower 720 server if needed. Indeed, this is one of the key advantages of the PowerVM virtualization technology: we can configure an LPAR with a small number of processor cores initially, and then, as the workload demand increases, this LPAR can grow to the full size of the physical server. In this configuration, the LPAR now has a total of 4 POWER5 cores, and as a result, the CPU utilization drops from 92% to 55% while the average service response time on this LPAR drops from 54.82 ms to 37.46 ms, given the same settings for the Workload Driver.

| Config | Max Tx | Threads | BTPS | Server | Data Type | Disk Type (Single Disk) | writes/sec | wKB/sec | avgrq-sz (sectors) | avgqu-sz | await (ms) | svctm (ms) | util |
|--------|--------|---------|-------|--------------|-----------|-------------------------|------------|---------|--------------------|----------|------------|------------|------|
| 6.1 | 20 | 1 | 9.4 | 2-core P5 LPAR | TX LOG | VIOS | 118 | 554 | 9.41 | 0.41 | 3.44 | 3.42 | 40% |
| | | | | | REC LOG | VIOS | 84 | 424 | 10.14 | 0.43 | 5.13 | 4.08 | 40% |
| 6.2 | 20 | 1 | 11.27 | 2-core P5 LPAR | TX LOG | No VIOS | 112 | 495 | 8.89 | 0.37 | 3.43 | 3.32 | 37% |
| | | | | | REC LOG | No VIOS | 84 | 416 | 9.1 | 0.26 | 2.93 | 2.86 | 25% |

*Table 4.4.2 – Impact of VIOS*
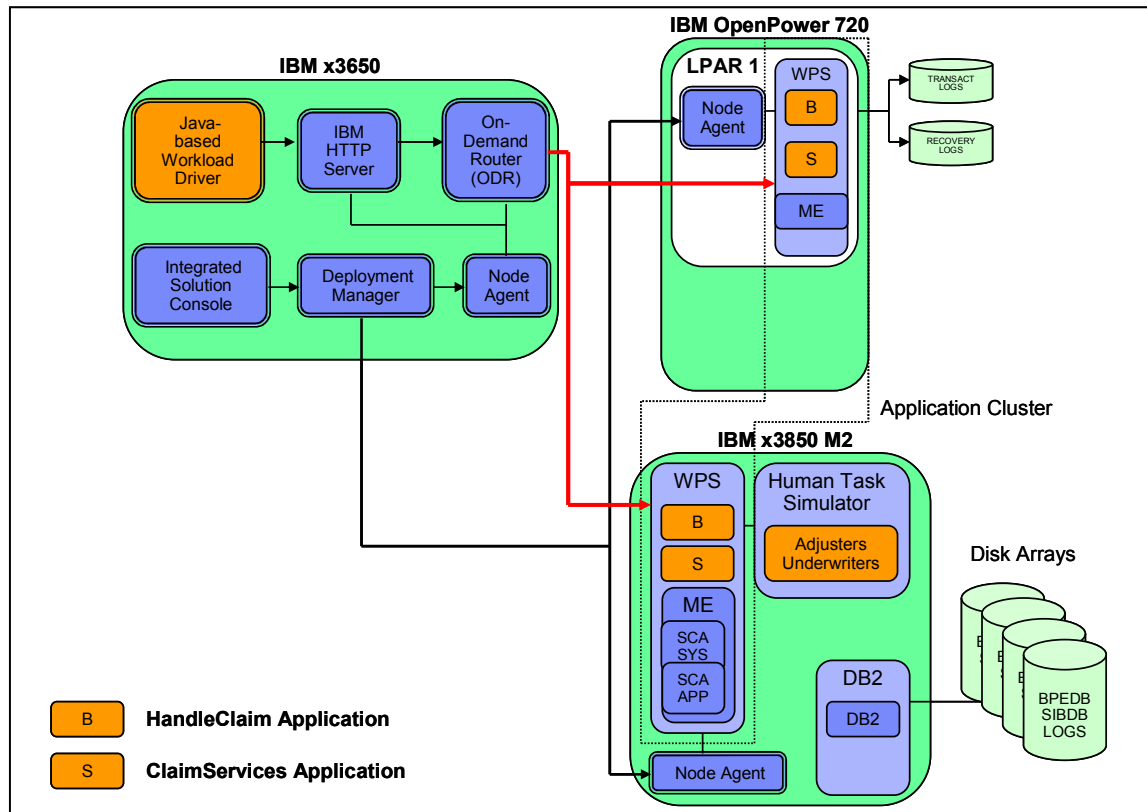


*Figure 4.4.5 – Dynamic application cluster with local Message Engines*

- **Disk I/O performance**: *Table 4.4.3* provides the disk I/O statistics for *Configuration 5.1* (three nodes: 1.5-core POWER5 virtualized LPAR, 2-core POWER5 virtualized LPAR, 16-core x3850 M2) and *Configuration 5.2* (two nodes: 4-core POWER5 LPAR without VIOS, 16-core x3850 M2). The data is provided for both light workload (Max Tx = 20, Threads = 1) and heavy workload (Max Tx = 160, Threads = 4). Both configurations have Message Engines in a separate cluster (ME Cluster). The data indicates that, for an SOA application such as the benchmark used in this study, from a performance perspective, it is better to use the entire OpenPower 720 server as a single partition without PowerVM technology than configuring LPARs with VIOS. In addition, we note the following from the data:

  o For the disks attached to the virtualized LPARs, the extra latency in going through the Virtual I/O Server (VIOS) results in higher I/O service times and higher disk utilization (as seen from Linux). With VIOS, the I/O service times can reach as high as 15 ms, whereas without VIOS, the I/O service times are always less than 4 ms. Similarly, the disk utilization data (collected at the operating system level) can reach as high as 70% with VIOS while it always stays below 50% without VIOS. However, the disk I/O queue depth in all cases is less than 1, indicating that this is strictly an I/O latency issue and that the disk subsystems are not really the performance bottleneck in the configurations that we consider in this study.

  o Even under the heaviest workload conditions considered in this study, there is still quite a bit of available bandwidth left on the disk arrays: the disk I/O queue depth is mostly less than 1 and the utilization is mostly less than 50%. Even under the heaviest workload, the queue dept is only 1.24.

  o Data writes to the BPEDB database tend to be "bursty" in nature (based on the disk I/O queue depth and the disk utilization data). In contrast, data writes to the SIBDB database and to the logs (BPE, transaction, compensation) occur evenly over time (that is, the disk I/O queue depth data matches well with the disk utilization data).

  o Data writes to the disk arrays are in the sub-millisecond range, indicating that write-back caching is in effect and very effective, especially since there is not much I/O waiting in the queues. However, the MR10M controller delivers better performance (in terms of I/O service times) because its cache is located much closer to the server than in the case of the DS3400 controller where the controller is located remotely from the server over the fiber links.



**Impact of Messaging Infrastructure Clustering**

**3-Node = 16-core x3850 M2 + 1.5-core P5 LPAR + 2-core P5 LPAR**

**2-Node = 16-core x3850 M2 + 4-core P5 LPAR (no VIOS)**
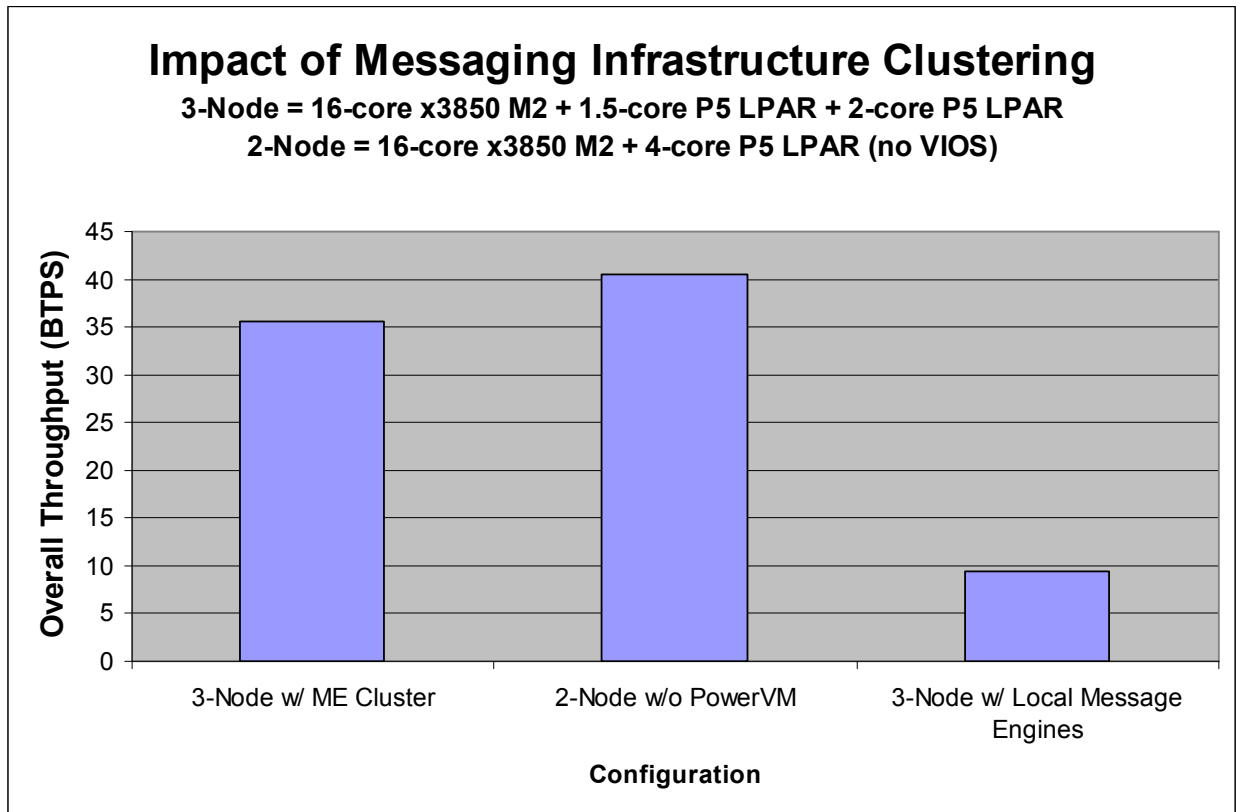
*Figure 4.4.6 – Impact of Message Engine clustering on overall throughput*

- **Tuning the On-Demand Router (ODR)**: The WVE test team recommends that the JVM heap size for the ODR should be at least 1024 MB and that the garbage collection policy should be *Generational Concurrent (gencon)* with the *noAdaptiveTenure* setting. However, because the service request rates in this study are fairly small (less than 100), this ODR tuning does not make much difference in the benchmark's overall throughput. For this study, the default JVM heap configuration for the ODR works just as well.

- **Workload driver**: The benchmark's Workload Driver, the IBM HTTP Server, the On-Demand Router, and the WebSphere Node Agent are hosted on the x3650 server. The CPU utilization of this server is less than 5% and the utilization of the only disk used on this server is less than 6%, indicating that this server does not have any negative impact on our performance results.

| Config | Max Tx | Threads | BTPS | Server | Data Type | Disk Type | writes/sec | wKB/sec | avgrq-sz (sectors) | avgqu-sz | await (ms) | svctm (ms) | util |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 5.2 | 160 | 4 | 38.73 | 16-core x3850M2 | BPE LOG | 12-disk MR10M | 1177 | 9261 | 15.74 | 0.06 | 0.06 | 0.05 | 6% |
| | | | | | SIBDB | 12-disk MR10M | 1451 | 6634 | 9.14 | 0.06 | 0.05 | 0.04 | 6% |
| | | | | | BPEDB | 24-disk DS3400 | 1555 | 6656 | 8.56 | 1.24 | 0.8 | 0.32 | 50% |
| | | | | | TX LOG | 12-disk DS3400 | 254 | 1121 | 8.52 | 0.17 | 0.68 | 0.68 | 17% |
| | | | | | REC LOG | 12-disk DS3400 | 186 | 891 | 9.58 | 0.12 | 0.64 | 0.64 | 12% |
| | | | | 8-core P5 | TX LOG | SCSI disk | 143 | 646 | 9.05 | 0.47 | 3.31 | 3.28 | 47% |
| | | | | | REC LOG | SCSI disk | 135 | 651 | 9.68 | 0.46 | 3.42 | 3.16 | 43% |
| | 20 | 1 | 26.36 | 16-core x3850M2 | BPE LOG | 12-disk MR10M | 1509 | 8257 | 10.94 | 0.07 | 0.04 | 0.04 | 7% |
| | | | | | SIBDB | 12-disk MR10M | 1978 | 8061 | 8.15 | 0.08 | 0.04 | 0.04 | 8% |
| | | | | | BPEDB | 24-disk DS3400 | 1014 | 4290 | 8.96 | 0.71 | 0.7 | 0.33 | 34% |
| | | | | | TX LOG | 12-disk DS3400 | 184 | 808 | 8.78 | 0.11 | 0.61 | 0.61 | 11% |
| | | | | | REC LOG | 12-disk DS3400 | 128 | 607 | 9.5 | 0.08 | 0.61 | 0.61 | 8% |
| | | | | 8-core P5 | TX LOG | SCSI disk | 129 | 577 | 8.95 | 0.42 | 3.3 | 3.24 | 42% |
| | | | | | REC LOG | SCSI disk | 101 | 485 | 9.6 | 0.31 | 3.1 | 2.96 | 30% |
| 5.1 | 160 | 4 | 35.56 | 16-core x3850M2 | BPE LOG | 12-disk MR10M | 972 | 8491 | 17.47 | 0.07 | 0.07 | 0.07 | 7% |
| | | | | | SIBDB | 12-disk MR10M | 1283 | 6076 | 9.47 | 0.08 | 0.06 | 0.06 | 8% |
| | | | | | BPEDB | 24-disk DS3400 | 1344 | 5980 | 8.9 | 1.04 | 0.77 | 0.34 | 46% |
| | | | | | TX LOG | 12-disk DS3400 | 226 | 1003 | 8.55 | 0.13 | 0.58 | 0.57 | 13% |
| | | | | | REC LOG | 12-disk DS3400 | 183 | 879 | 9.6 | 0.12 | 0.66 | 0.66 | 12% |
| | | | | 3.5-core P5 | TX LOG | VIOS disk | 67 | 330 | 9.62 | 0.53 | 10.85 | 10.77 | 57% |
| | | | | | REC LOG | VIOS disk | 42 | 228 | 10.62 | 0.64 | 15.83 | 14.56 | 65% |
| | | | | 4-core P5 | TX LOG | VIOS disk | 78 | 357 | 9.19 | 0.46 | 5.97 | 5.79 | 45% |
| | | | | | REC LOG | VIOS disk | 52 | 288 | 11.02 | 0.74 | 14.24 | 13.47 | 70% |
| | 20 | 1 | 15.24 | 16-core x3850M2 | BPE LOG | 12-disk MR10M | 1067 | 5558 | 10.41 | 0.04 | 0.04 | 0.04 | 4% |
| | | | | | SIBDB | 12-disk MR10M | 1481 | 5998 | 8.1 | 0.07 | 0.05 | 0.05 | 7% |
| | | | | | BPEDB | 24-disk DS3400 | 494 | 2064 | 8.35 | 0.3 | 0.61 | 0.37 | 18% |
| | | | | | TX LOG | 12-disk DS3400 | 91 | 395 | 8.72 | 0.04 | 0.46 | 0.46 | 4% |
| | | | | | REC LOG | 12-disk DS3400 | 65 | 308 | 9.42 | 0.04 | 0.56 | 0.56 | 4% |
| | | | | 3.5-core P5 | TX LOG | VIOS disk | 52 | 275 | 10.32 | 0.25 | 4.77 | 4.66 | 24% |
| | | | | | REC LOG | VIOS disk | 39 | 196 | 9.83 | 0.37 | 9.34 | 9.04 | 36% |
| | | | | 4-core P5 | TX LOG | VIOS disk | 70 | 324 | 9.31 | 0.28 | 3.95 | 3.87 | 27% |
| | | | | | REC LOG | VIOS disk | 43 | 213 | 9.93 | 0.39 | 9.07 | 8.89 | 38% |

*Table 4.4.3 – Performance statistics for disk subsystems*

# 5. SOA People entry point

The SOA People entry point is a starting point for SOA, enabling people to interact with applications and information services that support business processes. Enabling people to access multiple applications and information sources and allowing them to work together efficiently – the main pillars of the People entry point – are critical to the success of SOA implementations.

The People entry strategy to SOA can help:

- Accelerate productivity
- Reduce costs for access to multiple applications and information sources
- Reduce time to deployment for new services
- Increase access to process flexibility and orchestration
- Enable collaboration inside and outside the enterprise

IBM has introduced the *IBM Portal Pack for SOA Configurations* [14], which includes essential WebSphere Portal family components for optimal people interactions and collaboration in an SOA environment.  More specifically, the IBM Portal Pack includes the following products:

- *WebSphere Portal Server* for portal services and development capabilities
- *IBM Lotus ActiveInsight* for dashboarding and scorecarding
- *IBM Lotus Forms Server* for intuitive human-centric forms interface

For the People entry point, the most important component to be tuned for performance is the *WebSphere Portal Server*.  There is a comprehensive performance tuning guide for *WebSphere Portal V6.1* [15],

which discusses tuning parameters and performance considerations for the base portal, Web 2.0 theme, DB2 database, Web content management, clustering, and portal caches in the AIX®, Linux, and Microsoft® Windows® 2003 environments.

# 6. SOA Information entry point

Information as a service is an entry point to SOA that offers information access to complex, heterogeneous data sources within an enterprise as reusable services. By establishing information as a service, we can improve the availability and consistency of information, while simultaneously removing traditional barriers to information sharing.  Through this entry point, we can ensure consistent definitions, packaging, and governance of key business data.  Information services that can be easily reused across processes would lead to more business flexibility and increased IT resource productivity.  More specifically, the SOA Information entry point can help:

- Collect, clean and make data accessible:
    - Develop a unified view of businesses with in-line access to analytical data for improved transparency and business insight
    - Generate and govern authoritative master data records with shared metadata and data quality services for master data management

- Reduce cost and risk:
    - Reduce costs that are associated with infrastructure rationalization and migration by decoupling information from individual information sources
    - Reduce risk exposure through in-line analytics and auditable data quality for risk and compliance initiatives

- Increase business agility:
    - Increase the agility for business transformation by providing reusable information services, spanning structured and unstructured information that can be plugged into applications, business processes, and portals
    - Reduce development costs that are associated with accessing and transforming data

There is a whitepaper which focuses on designing appropriate solutions for high-value information services [16].  There is also an audio podcast from the International DB2 User Group (IDUG) on tuning a *DB2 Data Warehouse* [17], which is a key component for implementing information services.

## 7. Conclusions and recommendations

Based on the data analysis in *Section 4*, we can draw the following conclusions and recommendations:

- **Tuning Web services (SOA Services and Reuse entry point)**

  o Of the various tuning parameters examined in this paper, the size of the JVM heap in the WebSphere Application Server (WAS) has the greatest performance impact on both x86 and Power Web servers. It is critical to have the heap sized appropriately for the workload. It is also important to use a garbage collection policy that is suitable for the workload. For the workload considered in this paper, the `gencon` garbage collection policy appears to be the most effective.

  o The performance benefit of huge (large) page support in Linux is mixed in our testing. On the Power Web server, we find that allocating huge 16 MB pages for the WAS JVM heap improves performance. However, on an x86 Web server which has a much smaller huge page size (2 MB), we find that the use of huge pages for the heap is only beneficial if the heap is constrained; if an appropriately sized heap is configured, huge pages would be of little benefit.

  o Configuring adequate persistent HTTP connections delivers a modest improvement in performance on both Power and x86 platforms.

  o Disabling logging and tracing in the WebSphere Application Server has a small benefit after the other tuning items are in place.

  o In our testing, increasing the size of the thread pools in the WebSphere Application Server does not appear to yield much performance gain on the x86 Web server. On the Power Web server, increasing the size of the thread pools helps performance only if the WAS JVM heap is configured large enough to accommodate more concurrent threads. This shows that it is important to consider the tuning of all parameters as a whole because their effects can interact with one another.

- **Tuning service mediations (SOA Connectivity entry point)**

  o For transformation and routing service mediations, the default WebSphere and Linux settings can only support payload (message) size of 3 KB or less for service requests and responses.

  o Optimizing the WebSphere JVM heap configuration delivers the greatest performance gain – up to 23X improvement on the x3850 M2 server that hosts the mediations. It is therefore very important to have the JVM heap sized appropriately for the workload. Generational Concurrent (`gencon`) garbage collection policy appears to be the most suitable for Web service mediations.

  o For routing mediations, the huge page support in Linux delivers 16% and 8% performance improvement on the x3850 M2 and Power 570 servers, respectively. For transformation mediations, however, the huge page support does not yield any noticeable performance gain with a payload size of 3 KB or less. At larger payload sizes, it does yield a small, but noticeable, performance improvement for relatively complex mediation scenarios, such as Composite Module and Transform Schema, where mediation operations on large payloads often cross the normal 4 KB page boundaries.

  o On the x3850 M2 server, tuning for maximum concurrency only delivers noticeable performance gain when the service request rate is relatively high – more than 600 requests per second. (Concurrency tuning includes setting the maximum number of

persistent HTTP connections to *unlimited*, increasing the sizes of the default and Web Container thread pools, as well as disabling all performance monitoring, tracing, and logging.) On the Power 570 server, only the Transform Value mediation scenario could sustain a high enough service request rate for concurrency tuning to yield a small (6%) performance improvement. The remaining mediation scenarios did not see any performance gain.

o As the payload size in the service request and response messages increases, the Web service rates (throughput) drops, as expected. The size of the drop in throughput depends on the amount of processing that is required to perform the mediation on the message payloads. On the x3850 M2 server, as we increase the payload size to 100 KB for all request and response messages, the server CPU utilization surges above 90% and becomes a performance bottleneck for all WESB service mediation scenarios – with the exception of the Transform Value scenario. In the Transform Value scenario, the network traffic and latency become the limiting factor before the server CPUs because of the high rate of service requests and responses, each with a large 100 KB payload. However, on the Power 570 server, the server CPU utilization is always very high, and as the payload size increases to 100 KB, the server CPUs become the absolute bottleneck, preventing us from getting higher throughput.

- **Tuning business process management in a nonclustered environment (SOA Process entry point)**

  o As with other WebSphere facets, the WebSphere JVM heap configuration for the WPS server is important. For the BPM scenarios considered in this paper, the default JVM heap size is too small. At least 2048 MB is recommended. In some cases, increasing the JVM heap size from 2048 MB to 4096 MB can result in a 20% performance gain. The JVM heap size should also be adjusted accordingly after adding resources, such as processors, to the server or partition that hosts the WPS server. It is also recommended to use the Generational Concurrent (`gencon`) garbage collection policy because it is most suitable for Web service transactions.

  o With a large volume of disk write traffic to the SIBDB database, BPEDB database, and BPE log, it is recommended that they be hosted on physical disk arrays, such as those attached to the IBM ServeRAID MR10M and IBM System Storage DS3400 controllers used in this study. If the disk arrays are attached to a host controller through a fiber-channel interface (such as the DS3400), it would be better to host the BPEDB database on its own separate disk array. This is because, in our testing, the average disk I/O queue depth becomes greater than 1.0 under peak workload conditions even when we dedicate a 24-disk array entirely to BPEDB.

  o For best performance, cache mirroring and write-through caching in the disk array controllers should be disabled. However, this is a trade-off between performance and reliability/availability.

  o For data writes to WPS databases and logs, the IBM ServeRAID MR10M disk subsystem delivers better performance than the IBM System Storage DS3400 disk subsystem. This can be attributed to the fact that the write-back cache on the MR10M is much closer to the WPS server than the cache in the DS3400, which is located remotely from the server over the fiber links.

  o It is better to have the DB2 component run on a separate physical server from the WPS server, especially when there are multiple WPS servers in a cluster configuration.

  o Business process modeling applications, such as those used in our benchmark, are generally processor-intensive. Adding more processors to the WPS server would certainly increase the throughput of those applications, provided that some J2C

Connection Factories and Activation Specifications would also be increased to accommodate the higher number of processors as recommended by the WebSphere Performance Team [13].

o   The transaction and compensation logs for each WPS server should be hosted on separate disks if physical disk arrays are not available.  These disks (or disk arrays) should be local to the WPS server.  Hosting the transaction and compensation logs on a remote server, even on disk arrays, would result in very low performance due to the large amount of disk writes involved and the latency requirements. Logical disk striping through the Linux Logical Volume Manager (LVM) is also not very effective in handling the log traffic.

o   The WAS and WPS concurrency parameters, such as thread pools (e.g. default, web container), connection pools (for BPEDB, CEI ME, SIBDB System, and SIBDB BPC data sources), J2C connection factories (for BPECF, BPECFC, HTMCF), and J2C Activation Specifications, should have appropriate values based on the number of processors (as seen by the operating system).  In this study, giving these parameters appropriate values essentially doubles the benchmark's overall throughput.

o   Tuning DB2 log parameters, such as the maximum storage for lock list, log buffer size, and log file size, yields only minor (less than 2%) improvement in the overall business transaction rate.

o   The Linux Huge (Large) Page Support is not effective for BPM scenarios.

- **Tuning dynamic operations in a WebSphere Virtual Enterprise (WVE) environment (SOA Process entry point)**

  o   WebSphere Virtual Enterprise, with its On-Demand Router (ODR) and associated autonomic managers, creates a virtual, goal-driven, autonomic cluster environment for SOA applications.  Such an environment can optimize resource usage and provide the capability of responding in real-time to spikes in workload demands (without human intervention if *Automatic Mode* is used).  This capability includes the activation of new application server instances on available physical servers to handle the peak workload.

  o   Through the use of Java, the application servers are really separated from the physical server hardware.  In our setup, application server instances are activated on different server hardware platforms (Power and x86) to handle the same stream of service requests.  A single DB2 server instance on an x86 server can support multiple WPS server instances on that x86 server as well as on other Power servers.

  o   For messaging infrastructure clustering, having the Message Engines running locally in the same dynamic cluster as the WPS applications is faster, but it severely limits the overall scalability of a multi-node configuration.  For multi-node configurations, it is recommended that the Message Engines run in their own cluster, separate from the application cluster.  Although desirable, it is not necessary for the Message Engine cluster to run on different physical servers from those that host the application clusters to achieve good scalability.  In our study, the Message Engine cluster runs on the x3850 M2 server, which hosts DB2 and is also part of the dynamic application cluster.

  o   Care should be taken when setting the performance goal in the service policy.  WVE V6.1 only supports service goals that are related to response times.  Setting the service goal too close to the minimum service response time on a lightly loaded configuration would never result in additional servers to be activated because the autonomic managers would determine that starting additional servers would not improve the capability of meeting the service goal anyway.  In our study, we set the service goal to be two times the minimum service response time.

o It takes at least 3 to 4 minutes from the time a service goal is considered *breached* until additional servers are started and running. Even then, it might take a few more minutes for the new servers to warm up and be able to handle service requests at optimal rates.

o From a pure performance perspective, if a Power server is used to host WPS servers, it is better to configure a single LPAR that has all the resources on the physical server and use dedicated disks for transaction and recovery logs than configuring multiple LPARs with PowerVM virtualization technology. For the benchmark in this study, a single LPAR that has all four POWER5 cores on an OpenPower 720 server delivers 14% more business transactions per second than two virtualized LPARs on the same server. Alternatively, with PowerVM virtualization technology, we can start small and put the LPAR in *shared, uncapped* mode so that the LPAR can grow into the full size of the physical server as workload demand increases.

o The Virtual I/O Server (VIOS) results in higher I/O service times and higher disk utilization. We have seen the virtual I/O service time as high as three times the dedicated I/O service time. The VIOS overhead is also responsible for 20% increase in the average service response time in this study. However, the PowerVM virtualization technology does allow us to configure the disks as dedicated I/O devices to the LPARs so that I/O operations to these disks do not go through VIOS. For WPS servers, if we need to use the PowerVM virtualization technology for availability and other virtualization benefits, we should host the transaction and compensation logs on physical disks dedicated to each LPAR.

o As we can see from the performance data in this study, the PowerVM virtualization technology does have some performance overhead. However, its features and benefits are quite many, among them:

- Server consolidation of existing workloads onto Power servers, thereby addressing the challenges of reducing space, power, and overall life-cycle costs
- High availability solutions with Live Partition Mobility (available for POWER6 processors or later only)
- The capability of WebSphere Virtual Enterprise to work with server provisioning solutions, such as the Tivoli Provisioning Manager (TPM) or Tivoli Intelligent Orchestrator (TIO), to dynamically allocate new (physical or virtual) servers on which application server instances can be activated to handle unexpected surges in workload demands
- Low-cost application security and isolation with EAL4+ certification
- Far better granularity than hardware partitioning techniques (for example, multiple LPARs can be allocated within a single processor to a granularity of $1/100^{th}$ of a processor, with $1/10^{th}$ of a processor being the minimum for an LPAR)
- The capability of virtualizing and reconfiguring I/O, as well as support for a mix of shared and dedicated I/O across LPARs
- The shared and uncapped mode, which allows an LPAR to grow to the full size of the physical server as long as resources are available to handle peak workload demands

The last two features – the support for dedicated I/O and shared, uncapped LPAR mode – can help reduce some performance overhead of the PowerVM virtualization technology in our setup. More specifically, for WPS servers, the transaction and compensation logs should be hosted on physical disks dedicated to each LPAR. Instead of adding more processor cores to an LPAR, we can start small and put the LPAR in shared, uncapped mode where it can use additional available processors on the physical server as necessary.

- For the business process scenarios considered in this study, the benchmark's Workload Driver, the IBM HTTP Server, and the On-Demand Router do not consume much CPU or I/O resources at run time. The default configurations for the IBM HTTP Server and the On-Demand Router are found to be more than adequate to support the request rates of less than 100 requests per second in this study.

- **SOA People and Information entry points**: The SOA People entry point enables people to access multiple applications and information sources and allowing them to work together efficiently while the SOA Information entry point offers information access to complex, heterogeneous data sources within an enterprise as reusable services. Both of these entry points are critical to the success of SOA implementations. The two key components for these entry points are the *WebSphere Portal Server* and the *DB2 Data Warehouse*. Performance tuning of the WebSphere Portal Server in the Linux environment can be found in the comprehensive performance tuning guide for WebSphere Portal Server V6.1 [15]. There is also an audio podcast from IDUG on optimizing the performance of a DB2 Data Warehouse [17].

# *References*

[1] Services Oriented Architecture (SOA) Entry Points,
http://www-306.ibm.com/software/solutions/soa/entrypoints/index.html?S_TACT=107AG01W&S_CMP=campaign

[2] WebSphere Application Server,
http://www-306.ibm.com/software/webservers/appserv/was/

[3] WebSphere Process Server: IBM's New Foundation for SOA,
http://www.ibm.com/developerworks/websphere/library/techarticles/0509_kulhanek/0509_kulhanek.html

[4] Clustering WebSphere Process Server V6.0.2, Part 1: Understanding the topology,
http://www.ibm.com/developerworks/websphere/library/techarticles/0704_chilanti/0704_chilanti.html

[5] IBM Power 570, http://www-03.ibm.com/systems/power/hardware/570/

[6] IBM POWER6 Microarchitecture, IBM Journal of Research and Development,
http://www.research.ibm.com/journal/rd/516/le.html

[7] IBM System x3850 M2,
http://www-07.ibm.com/systems/includes/content/x/pdf/XSD03019USEN.pdf

[8] IBM System x3850 M2 Enterprise Server's X4 Technology,
http://www-03.ibm.com/systems/x/hardware/enterprise/x3850m2/x4/info.html

[9] Nghiem, Alex (October 8, 2002). *The Basic Web Services Stack: IT Web Services*: "A Roadmap for the Enterprise", Prentice Hall.

[10] SOABench 2005, Version 0.18, Document Owner: Andrew Schofield, Internal IBM Confidential Document (April 18, 2006).  (This document is available only with the approval of the document owner.)

[11] IBM OpenPower 720, http://www.ibm.com/systems/p/hardware/openpower/720/index.html

[12] SOA Performance on Linux:  Services and Reuse Entry Point, by Steve Dobbelstein, Khoa Huynh, Vivek Kashyap, and Mark Peloquin, Internal IBM Document (June 2008)

[13] WebSphere Process Server (WPS) 6.1.0, WebSphere Enterprise Service Bus (WESB) 6.1.0, WebSphere Business Monitor (Monitor) 6.1.0, WebSphere Adapters (WA) 6.1.0 Performance Report, by WebSphere Process Server, Enterprise Service Bus, Adapter, and Monitor Performance Teams, Internal IBM Document (April 2008)

[14] IBM Portal Pack for SOA Configurations, http://www-01.ibm.com/software/lotus/portal/packforsoa/?S_TACT=107AG01W&S_CMP=campaign

[15] IBM WebSphere Portal 6.1 Performance Tuning Guide, http://www-01.ibm.com/support/docview.wss?rs=688&uid=swg27013972, by IBM WPLC Performance Team (October 2008)

[16] Delivering High-Value SOA Information Services, SOA Solutions Whitepaper,
ftp://ftp.software.ibm.com/software/solutions/soa/pdfs/WSW11332-USEN-00_SOA_BUSINESS_WP_0806A.pdf (2007)

[17] Tuning Your DB2 Data Warehouse, IDUG Podcast, http://www.idug.org/podcast-downloads/344.html