WebSphere Business Integration Message Broker for Linux V5

Performance report

Version 1.0

January, 2004

Tim Dunn

Kevin Braithwaite

Messaging Technologies Performance
IBM UK Laboratories
Hursley Park
Winchester
Hampshire
SO21 2JN

Take Note!

Before using this report be sure to read the general information under "Notices".

First Edition, January 2004.

This edition applies to *WebSphere Business Integration Message Broker for Linux V5 CSD2.*

Notices

This report is intended for Architects, Systems Programmers, Analysts and Programmers wanting to understand the performance characteristics of **WebSphere Business Integration Message Broker for Linux V5 CSD2 level of code**. The information is not intended as the specification of any programming interfaces that are provided by WebSphere MQ or WebSphere Business Integration Message Broker for Linux – V5. It is assumed that the reader is familiar with the concepts and operation of WebSphere Business Integration Message Broker V5.

References in this report to IBM products or programs do not imply that IBM intends to make these available in all countries in which IBM operates.

Information contained in this report has not been submitted to any formal IBM test and is distributed “asis”. The use of this information and the implementation of any of the techniques is the responsibility of the customer. Much depends on the ability of the customer to evaluate these data and project the results to their operational environment.

The performance data contained in this report was measured in a controlled environment and results obtained in other environments may vary significantly.

Trademarks and service marks

The following terms, used in this publication, are trademarks of the IBM Corporation in the United States or other countries or both:

- IBM
- Netfinity
- WebSphere MQ
- WebSphere MQ Integrator
- WebSphere Business Integration Message Broker
- DB2

The following terms are trademarks of other companies:

- Red Hat, Red Hat Inc.

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

Summary of Amendments

Date	Changes
06 January 2004	Initial Release

Feedback

This report and other tools that are produced by the performance group are produced in order to help you understand the performance characteristics of WebSphere Business Integration Message Broker and to assist you with Capacity Planning. It is important that the reports and tools are effective in what they do and it is very useful to have feedback on the content and style of the information which is produced. Your comments, both positive and negative, are therefore welcome.

Your answers to the following questions are particularly interesting:

- What are your most common performance questions?
- Do the reports provide what is needed?
- Is there any other performance information which is required to help you do your job?
- Would you like to see any other aspects of WBIMB performance discussed?

Please supply feedback to the WBIMB Performance Group user ID (email address WMQPG@uk.ibm.com).

Contents

1 Introduction

The purpose of this report is to illustrate the processing characteristics of WebSphere Business Integration Message Broker (WBIMB). This has been done by measuring the message throughput which is possible for a number of different message nodes. The test cases used have been made deliberately trivial in order to be able to report the cost of using WBIMB, rather than to report the cost of running a particular application.

The effect of the WBIMB broker queue manager has been minimized where possible. This has meant using predominately non persistent messages as well as having a compensating program to ensure that the WebSphere MQ queue manager queue cache did not overflow to disk. If these two actions were not taken, the throughput possible with WBIMB would have been constrained by the necessary I/O processing of the WebSphere MQ queue manager with which the WBIMB broker was associated.

The performance measurements have focused on the throughput capabilities of the broker using different processing node types. The aim of the measurements was to be able to answer questions such as how many messages a second can be processed with each of the node types and what are the relative costs of the different node types.

The following node types have been measured:

- Aggregation
- Compute
- Database
- Filter
- FlowOrder
- MQInput and MQOutput
- Publication
- RouteToLabel

These nodes give a cross section of the possible node types and should be sufficient to cover most basic types of message transformation and distribution. All the nodes measured used minimal processing where it was possible (apart from the investigation into complex node processing) so the results presented represent the best throughput that can be achieved for that node type. This should be borne in mind when performing capacity planning.

All measurements are for a single instance of a message flow in a single execution group unless otherwise noted. Although this does not show the maximum throughput possible with each type of node on the measurement machine it does provide a common methodology and shows the relative costs of nodes.

Recommended minimum specification machines are given for each of the WBIMB components. This information is given in Chapter 5, Recommended Minimum Specification.

All measurements were conducted in the same measurement environment. This is described in Chapter 7, Measurement Hardware and Software.

All measurements were obtained using the same methodology. This is described in Chapter 8, Evaluation Method.

2 Maximum Message Broker Throughput

WBIMB provides the capability to process messages in a variety of message formats, those with legacy data structures, XML or tagged delimited strings for example. Using WBIMB it is possible to both parse such messages when they are an input message and write them when they are an output message. Once an input message has been parsed it is possible to perform many different types of processing. Typical uses of WBIMB are message routing, message enrichment, message transformation and publish/subscribe.

WBIMB provides the capability to run more than one copy of a message flow within a broker. Multiple copies of a message flow are usually run in order to increase message throughput. Using the mechanisms provided in WBIMB it is possible to fully utilize a server machine and so achieve much higher message throughput rates than would be obtained with a single copy of the message flow.

WBIMB provides an easy method to increase the number of copies of a message flow which are running. This is an operational consideration rather than a design or coding issue for the message flow. As such you can change the number of copies of a message flow which are run to meet differing requirements.

The recommended way of running multiple copies of a message flow on Linux is to assign a copy of the message flow to more than one execution group. In order to run multiple copies of the message flow multiple execution groups within the broker are required.

The following sections illustrate the message rates that are possible for a variety of different types of message processing when fully utilizing the server machine on which the performance measurements were taken. These flows are simple in nature but illustrate the magnitude of processing that is possible with WBIMB. The results shown are an indication only. The use of faster or slower processors or more complex message processing will have a corresponding effect on message throughput.

2.1 Message Routing

Through the use of a filter node it is possible to implement simple message routing. The filter node is able to apply an ESQL expression against the content of the input message. Based on the result of the expression evaluation a message can be passed through one of two node terminals. The first terminal (True) could lead to one message queue, via an MQOutput node. The second terminal (False) could lead a second message queue or another filter node.

In order to illustrate this type of processing a message flow consisting of an MQInput node, a Filter node and an MQOutput node was defined. This is the same processing that is described in Section 3.5, Filter Node. Sufficient copies of the message flow were run in order to fully utilize the server machine. With this message flow it was possible to process approximately **4039** 1KB non persistent msgs/second.

2.2 Message Enrichment from a Database

In some applications there is a need to read information from a database. This might be to validate the contents of a field in an input message or to include data from the database in the output message. Such processing requires one or more rows to be read from a database.

In order to illustrate this type of processing a message flow consisting of an MQInput node, a Database node and an MQOutput was defined. This is the same processing that is described in

Section 3.4.3, Database Read. Sufficient copies of the message flow were run in order to fully utilize the server machine. With this message flow it was possible to process approximately **2628** 1KB non persistent msgs/second.

2.3 Message Update

ESQL provides the means to perform message manipulation. The level of complexity of such processing varies considerably as does the cost of running such processing. Simple processing requires less CPU and so is capable of running at higher message rates than more complex processing.

In order to illustrate a form of message processing a message flow consisting of an MQInput node, a Compute node and an MQOutput was defined. This is the same processing that is described in Section 3.3.4, Very Complex Compute . Sufficient copies of the message flow were run in order to fully utilize the server machine. With this message flow it was possible to process approximately **266** 1KB non persistent msgs/second.

2.4 Message Transformation

A typical use of WBIMB is to perform message transformation. For example it may be necessary to reorder message fields as two communicating applications expect the fields to be in a different sequence. In order to do such processing the input message must be read in, the fields reordered and the output message written.

To illustrate simple message transformation an XML input message was reordered by a message flow consisting of an MQInput node, a Compute node and an MQOutput node. This is the same processing that is described in the Generic XML to Generic XML test in Section 3.7, Message Conversion. Sufficient copies of the message flow were run in order to fully utilize the server machine. With this message flow it was possible to process approximately **986** 4KB non persistent msgs/second.

2.5 Complex Message Processing

In some situations WBIMB is required to perform complex processing on the contents of a message. This may involve reformatting, reordering and calculation to produce for example a financial statement.

In order to illustrate such processing an XML message was processed by a message flow consisting of an MQInput node, a Compute node and an MQOutput node. This is the same processing that is described in Section 3.3.5, Nested SELECT FROM Compute . Sufficient copies of the message flow were run in order to fully utilize the server machine. With this message flow it was possible to process approximately **714** 4KB non persistent msgs/second.

2.6 Summary of Message Broker Throughput

In the preceding sections there were five examples of different types of processing which vary in the complexity and function performed. The figure below summarizes the results into a single chart.

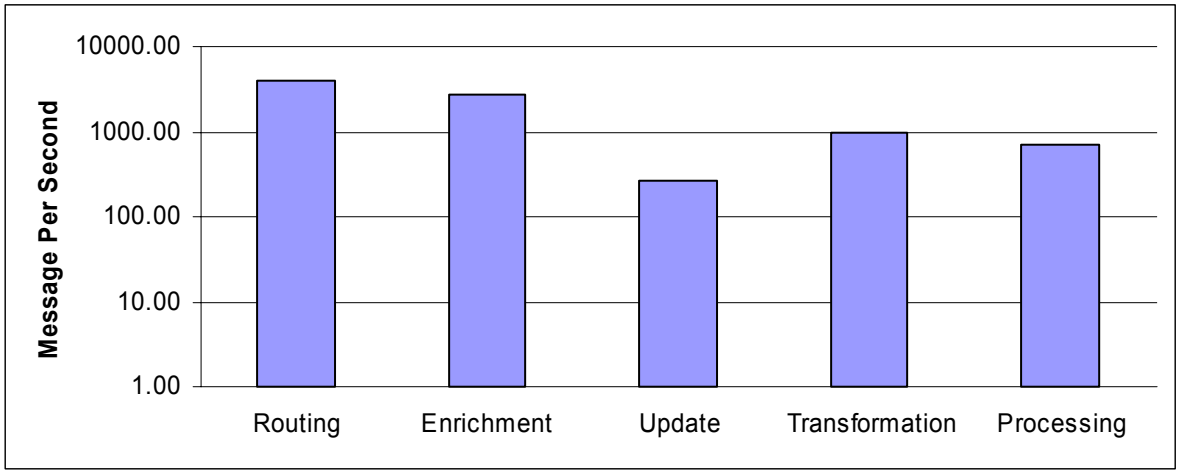

Figure 1: Summary of Message Throughput Tests

Note the difference between the maximum and minimum message rates which were obtained even though the same hardware and software was used for all of the tests. The message rates which you will achieve in practice will also be very different. They will be dependent on the resources which are allocated to running the message flow and the complexity of the message flow. Complexity is measured not only by the number of processing nodes along the critical path also by the nature of the processing within the nodes.

3 Message Node Processing Profiles

There are many different nodes available within WBIMB. These vary significantly in function and level of performance. This is understandable as they provide a wide range of functions. For example the Filter node can be used to test the value of one or more fields in a message and route messages at a high rate, whilst the aggregation nodes can be used to perform coordination of requests to a variable number of external applications. This coordination is clearly more involved than the testing of a single field in a message and the message rate achieved with the aggregation node is consequently lower.

Similarly the processing within a given node type can significantly affect the message rate which is achievable with that node. Take a Compute node for example. Within a Compute node it is possible to code ESQL. The quantity and complexity of the ESQL have a very direct effect on the message rate which is possible as you will see in the Compute node tests.

The purpose of this section is to profile some of the most commonly used processing nodes. Each of the nodes is implemented in a simple message flow and the maximum message rate possible with a single copy of that message flow is measured. The effect on message throughput of using different message sizes and level of persistence are shown. The results for each test are presented with a brief description of the test case followed by a graph of the measurement results. A table containing the actual measurement results is presented in Chapter 10, Measurement Data.

When examining the results of these tests it is important to understand whether the test was CPU or I/O bound. For any given test a higher message rate will be obtained if the test is CPU bound rather than I/O bound. For example CPU utilizations lower than 28% on a 4 CPU machine would indicate that the test is I/O bound to some extent. The 28% CPU consists of a CPU utilization of 25% (full utilization of one processor) for the single message flow running in an Execution Group and at least 3% for the Broker queue manager listener process through which messages arrive and depart the system (or local application if this is used instead). In a high volume environment this processing by the Broker queue manager listener or local application is likely to be even higher. CPU utilizations for each test are reported in Chapter 10, Measurement Data.

The extent to which processing is I/O bound can be reduced through the use of a faster I/O device than a SCSI disk, such as a device with a non volatile cache. Use of a faster I/O device in most circumstances leads to an improvement in message rate (however it may not always be possible to make a message flow CPU bound). As a general rule those tests which have more involved message flow processing are CPU bound even when processing persistent messages. In the testing on this platform SCSI disks where used for the WebSphere MQ queue manager and database manager logs.

3.1 A Trivial MQInput/MQOutput Message Flow

A message flow consisting of a single MQInput and MQOutput node represents the simplest form of message input and output. Measuring the throughput achievable with such a message flow shows the maximum message rate that can be achieved using WBIMB to move messages between WebSphere MQ queues.

A single message flow was defined, consisting of an MQInput node and MQOutput node. The transaction mode for the MQInput and MQOutput nodes was set to automatic.

The figure below shows the results that were obtained when running the message flow with varying message sizes and persistence. There was a single instance and single execution group running the message flow

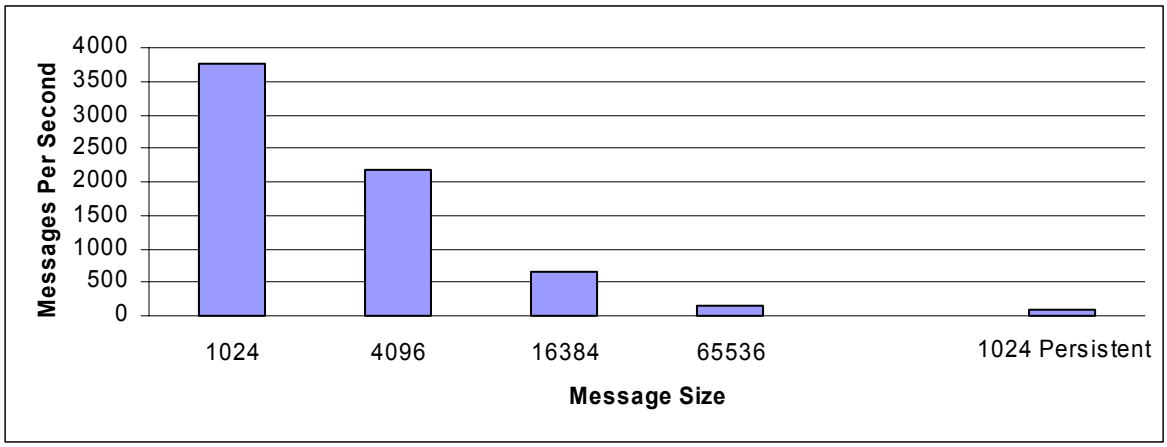

Figure 2: MQInput/MQOutput Throughput Results

With a 1KB non persistent message it was possible to process approximately 3753 msgs/second. The message throughput rate declined with size reflecting the increased volume of data.

With 1KB persistent messages it was possible to process approximately 91 msgs/second. This lower rate in comparison to the non persistent message processing is due to the fact that messages are processed within a WebSphere MQ unit of work with the consequent commit processing which involves I/O processing to the WebSphere MQ queue manager log.

The measurement data for this test is shown in Chapter 10, Measurement Data, Section 10.1.

Note: The results from this test indicate the best possible message throughput that will be achieved with a message flow on any given machine. This is because the messages are copied from the input queue to the output queue with the absolute minimum of processing. Such processing is not typical of WBIMB usage.

3.2 Aggregation Node

The aggregate node function in WBIMB provides an implementation of the request/reply processing model. The implementation is sufficiently flexible to allow a variable number of requests and replies to be coordinated. Use of the supplied aggregate nodes can reduce the complexity of the logic of the message flow thus allowing the user to concentrate on business logic and freeing them from the need to provide co-ordination logic. The nodes are typically used to coordinate one or more requests to back-end applications.

The aggregation support within WBIMB consists of a 'fan-out' phase in which one or more request messages are issued to applications and a 'fan-in' phase in which responses or reply messages are collected together. A consolidated reply can then be generated. Aggregation support is provided through the AggregateControl, AggregateRequest and AggregateReply nodes.

Testing consisted of a fan-out message flow, a simple C application which copied messages from one queue to another and a fan-in message flow. The fan-out message flow generated four request messages which where written to an intermediate request queue. The simple C application emulated a back-end application and copied the messages from the intermediate request queue to an intermediate reply queue. The fan-in message flow processed the messages appearing on the intermediate reply queue and produced a consolidated output message.

Coding of the message flows and configuration of the broker environment followed the recommendations given in **SupportPac IP05, WebSphere MQ Integrator V2.1 - Optimizing Use of Aggregation Nodes** which is available at
http://www.ibm.com/software/integration/support/supportpacs/individual/supportpacs/ip05.pdf

The figure below shows the results that were obtained when running the aggregation test with varying message sizes and persistence. One copy of the fan-out message flow, four copies of the fan-in message flow and one copy of the simple C application were run for maximum message throughput.

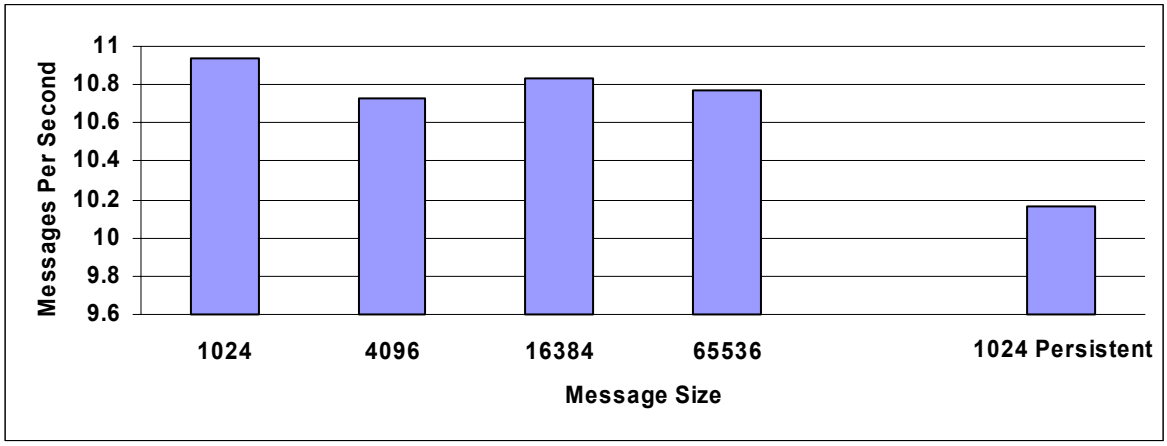

Figure 3: Aggregation Node Throughput Results

With a 1KB non persistent message it was possible to process approximately 11 msgs/second. The rate of insert/delete activity did not change with message size. This was due to the message being heavily I/O bound, only utilizing 6% CPU.

With 1KB persistent messages it was possible to process approximately 10 msgs/second. This lower rate in comparison to the non persistent message processing is due to the fact that messages are

processed within a WebSphere MQ unit of work with the consequent commit processing which involves I/O processing to the WebSphere MQ queue manager log.

Message throughput with the aggregation nodes is heavily dependent on the speed and type of disks used for the database log (in this case SCSI disks were used). This is because each message which is sent by the fan-out message flow and received by the fan-in message flow must be inserted into a database as a BLOB. It is possible to increase message throughput by increasing the number of copies of the fan-out and fan-in message flows. SupportPac IP05 has more details on this (see the start of this section for details on where to find this).

The measurement data for this test is shown in Chapter 10, Measurement Data, Section 10.2.

3.3 Compute Node

A compute node provides the capability to derive an output message from an input message and also optionally include user specified processing as well as data values from an external relational database. The compute node has the potential to vary from simple to complex in its processing. The degree of complexity specified has a direct bearing on the message throughput rates that can be achieved using nodes of that type. A series of measurements were taken using varying numbers of compute nodes as well as different levels of user specified processing in order to illustrate these effects.

The following cases were measured:

- A simple compute node that copied the input message to an output message. The purpose of this measurement was to show the message throughput that is achievable when copying a message and modifying a single field. A single field was modified in order to ensure that the compute node built a new output message based on the input. If no field is modified WBIMB optimizes the process and simply repeats the input message which can give an unrealistic message rate**.** This represents the simplest form of compute node.

- A single complex compute node that contained user specified ESQL processing as well as the copying of the input message to an output message. The purpose of this measurement was to show the effect that additional CPU bound processing has on message throughput.

- Multiple complex compute nodes that consisted of five of the complex compute nodes connected in sequence. The purpose of this measurement was to establish the cost of using multiple complex compute nodes.

- A single very complex compute node that consisted of five times the processing of the single complex compute node. The purpose of this measurement was to illustrate the benefit that can be obtained by combining processing within a single compute node.

- A single compute node containing a nested ESQL SELECT statement. The purpose of this measurement was to illustrate the throughput possible when using more complex ESQL.

The ESQL used for each of the tests is given in Chapter 9, Compute Node ESQL.

3.3.1 Simple Compute Test

The figure below shows the results that were obtained when running the simple compute node with varying message sizes and persistence. There was a single instance and single execution group running the message flow.

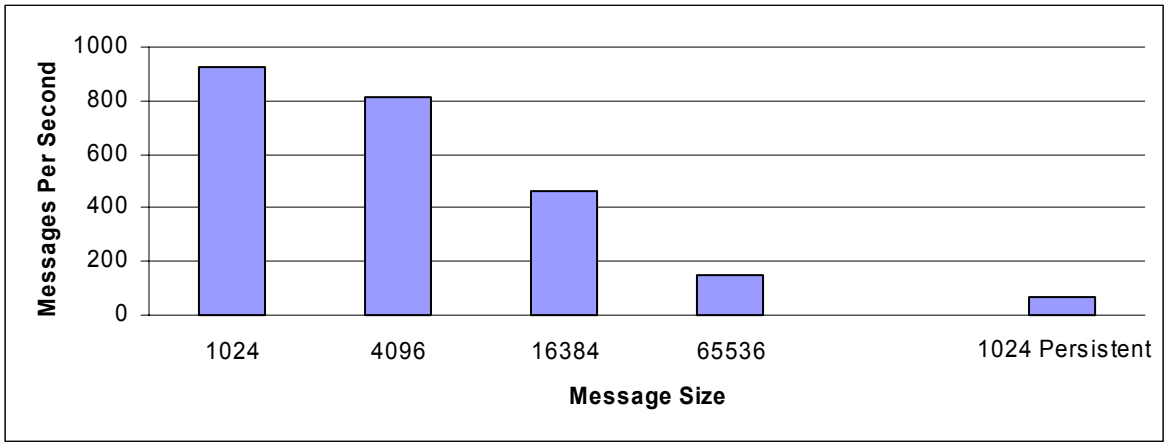

Figure 4: Simple Compute Node Throughput Results

With a 1KB non persistent message it was possible to process approximately 922 msgs/second. The message throughput rate declined with size, reflecting the increased volume of data.

With 1KB persistent messages it was possible to process approximately 67 msgs/second. This lower rate in comparison to the non persistent message processing is due to the fact that messages are processed within a WebSphere MQ unit of work with the consequent commit processing which involves I/O processing to the WebSphere MQ queue manager log.

The measurement data for this test is shown in Chapter 10, Measurement Data, Section 10.3.1.

3.3.2 Complex Compute Test

The figure below shows the results that were obtained when running a complex message flow with varying message sizes and persistence. There was a single instance and single execution group running the message flow. Due to the message complexity the minimum size message that could be used was 4k.

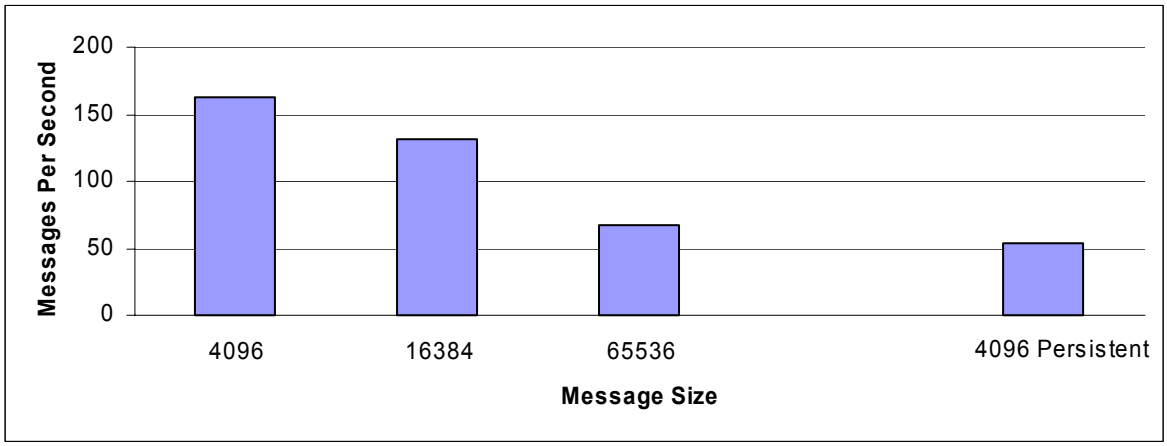

Figure 5: Complex Compute Node Throughput Results

With a 4KB non persistent message it was possible to process approximately 163 msgs/second. The message throughput rate declined with size, reflecting the increased volume of data. The lower message rate achieved with this compute node compared with the simple compute node case above reflects the increased processing that was added to the compute node.

With 4KB persistent messages it was possible to process approximately 53 msgs/second. This lower rate in comparison to the non persistent message processing is due to the fact that messages are processed within a WebSphere MQ unit of work with the consequent commit processing which involves I/O processing to the WebSphere MQ queue manager log.

The measurement data for this test is shown in Chapter 10, Measurement Data, Section 10.3.2.

3.3.3 Multiple Complex Compute Test

The figure below shows the results that were obtained when running five of the above complex nodes daisy chained together for varying message sizes and persistence. There was a single instance and single execution group running the message flow. Due to the message complexity, the minimum size message that could be used was 4k.

Figure 6: Multiple Complex Compute Node Throughput Results

With a 4KB non persistent message it was possible to process approximately 58 msgs/second. The message throughput rate declined with size, reflecting the increased volume of data.

With 4KB persistent messages it was possible to process approximately 41 msgs/second. This lower rate in comparison to the non persistent message processing is due to the fact that messages are processed within a WebSphere MQ unit of work with the consequent commit processing which involves I/O processing to the WebSphere MQ queue manager log.

The measurement data for this test is shown in Chapter 10, Measurement Data, Section 10.3.3.

3.3.4 Very Complex Compute Test

The figure below shows the results that were obtained when running a very complex message flow with varying message sizes and persistence. Briefly the very complex flow consists of the ESQL in the complex flow repeated 5 times in the same node. There was a single instance and single execution group running the message flow. Due to the message complexity, the minimum size message that could be used was 4k.

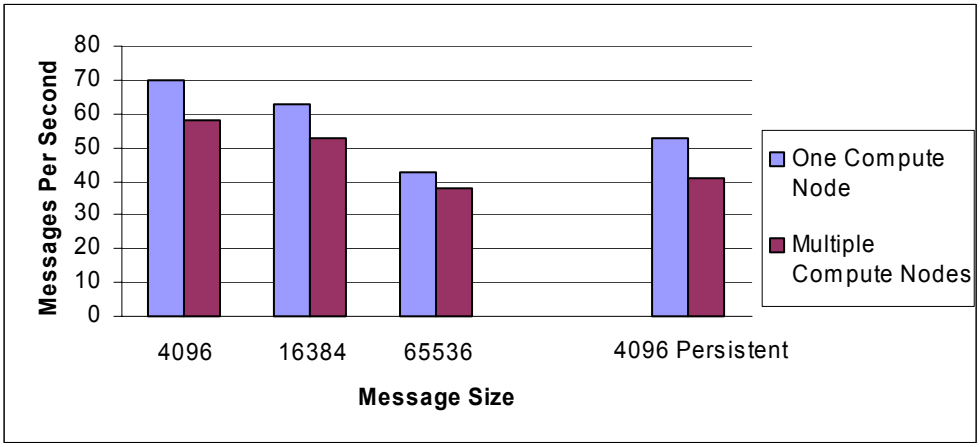

Figure 7: Very Complex Compute Node VS Multiple Complex Compute Node Throughput Results

With a 4KB non persistent message it was possible to process approximately 70 msgs/second. The message throughput rate declined with size, reflecting the increased volume of data.

With 4KB persistent messages it was possible to process approximately 53 msgs/second. This lower rate in comparison to the non persistent message processing is due to the fact that messages are processed within a WebSphere MQ unit of work with the consequent commit processing which involves I/O processing to the WebSphere MQ queue manager log.

For comparison purposes the graph also shows the message throughput rates that were achieved for the multiple complex compute node case detailed in Section 3.3.3, Multiple Complex Compute Test.

For 4KB non persistent messages there was a 1.2 times improvement in message throughput when using a single compute node for the processing, rather than using 5 nodes. For performance reasons it is clearly better to have one node that does the work of several less complex nodes. This performance improvement has to be offset against the management and support of more complex nodes.

The measurement data for this test is shown in Chapter 10, Measurement Data, Section 10.3.4.

3.3.5 Nested SELECT FROM Compute Test

The figure below shows the results that were obtained when running a nested SELECT statement within a compute node with varying message sizes and persistence. There was a single instance and single execution group running the message flow.

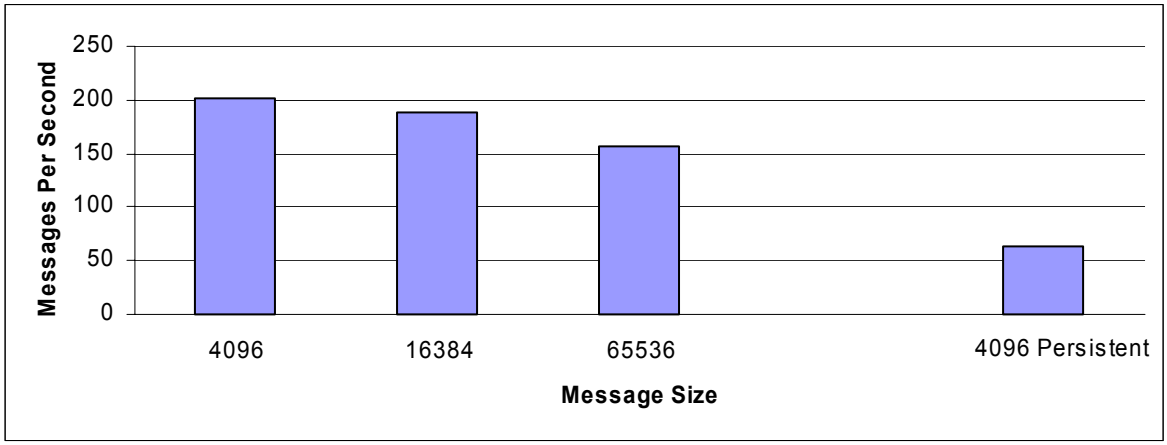

Figure 8: Nested SELECT FROM Compute Node Throughput Results

With a 1KB non persistent message it was possible to process approximately 202 msgs/second. The message throughput rate declined with size, reflecting the increased volume of data.

With 1KB persistent messages it was possible to process approximately 64 msgs/second. This lower rate in comparison to the non persistent message processing is due to the fact that messages are processed within a WebSphere MQ unit of work with the consequent commit processing which involves I/O processing to the WebSphere MQ queue manager log.

The measurement data for this test is shown in Chapter 10, Measurement Data, Section 10.3.5.

3.4 Database Node

A database node allows a database transaction in the form of an ESQL expression to be applied to a specified ODBC data source. The statement to be applied and the data source are specified in the database node definition.

In the following test the database transaction was not part of a global unit of work as the coordinated Transaction attribute of the message flow was not enabled. This should be borne in mind when performing capacity planning.

Three tests were run to illustrate different uses of a database node. In the first a row was inserted and deleted in a table of a database. In the second a randomly selected row within a table was updated. In the third a row within a table was read by a message flow.

3.4.1 Database Insert/Delete

A message flow consisting of an MQInput node, a database node and an MQOutput node was run. The processing in the database node consisted of an insert/delete for a row in a table of a database. The transaction mode value on the MQInput node was set to a value of automatic. The coordinated Transaction value for the message flow was set to no. The effect of doing this is to specify that the message flow should not be a globally coordinated unit of work.

The maximum possible message throughput rates were determined for a single instance and single execution group running the message flow. The figure below shows the results that were obtained for varying message size and persistence.

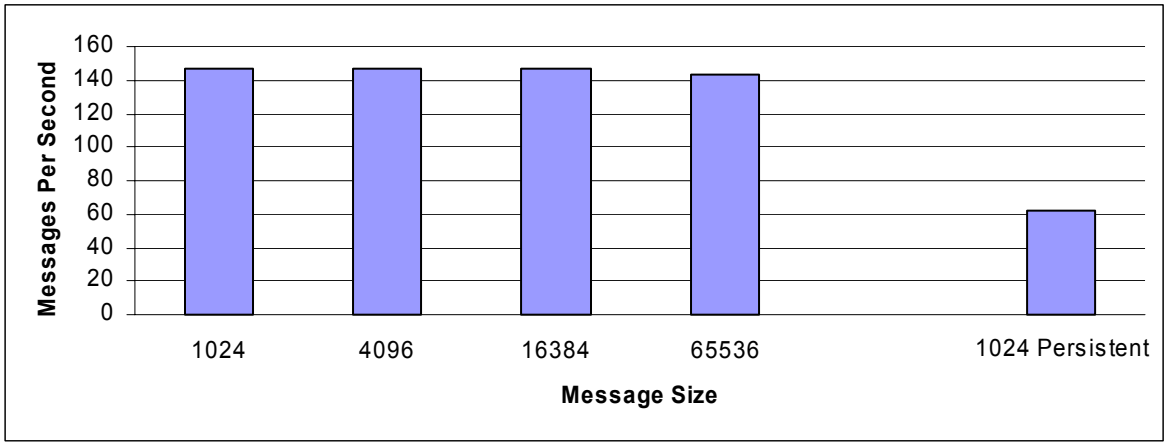

Figure 9: Database Insert/Delete Throughput Results

With 1KB non persistent messages it was possible to achieve a message throughput rate of approximately 147 msgs/second. This is 147 database insert and deletes per second. The rate of insert/delete activity did not change with message size. This was due to the message being heavily I/O bound, only utilizing 7% CPU.

With 1KB persistent messages it was possible to process approximately 62 msgs/second. This lower rate in comparison to the non persistent message processing is due to the fact that messages were processed within a WebSphere MQ unit of work.

The measurement data for this test is shown in Chapter 10, Measurement Data, Section 10.4.1.

3.4.2 Database Update

A message flow consisting of an MQInput node, a database node and an MQOutput node was run. The processing in the database node consisted of an update to a randomly selected row in a table of a database. The transaction mode value on the MQInput node was set to a value of automatic. The coordinated Transaction value for the message flow was set to no. The effect of doing this is to specify that the message flow should not be a globally coordinated unit of work.

The maximum possible message throughput rates were determined for a single instance and single execution group running the message flow. The figure below shows the results that were obtained for varying message size and persistence.

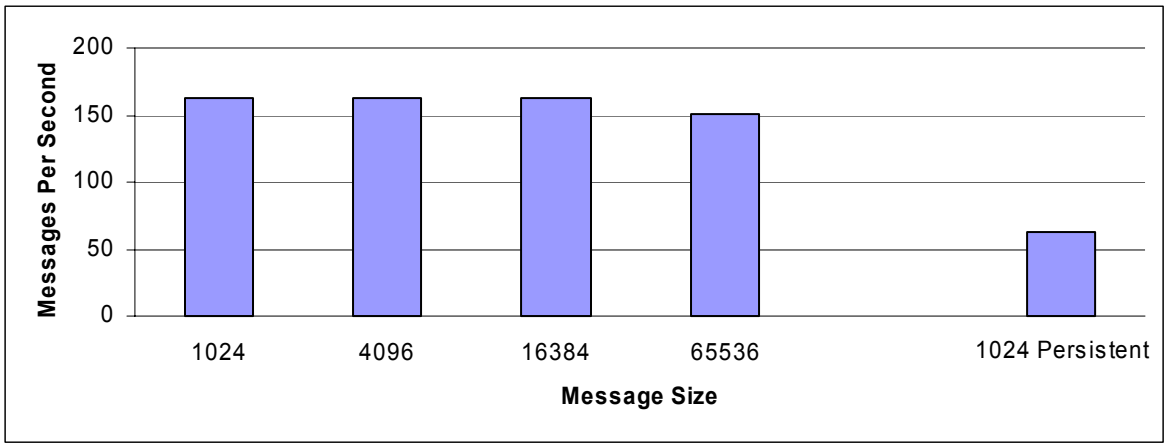

Figure 10: Database Update Throughput Results

With 1KB non persistent messages it was possible to achieve a message throughput rate of approximately 163 msgs/second. This is 163 database updates per second. This was due to the message being heavily I/O bound, only utilizing 7% CPU.

With 1KB persistent messages it was possible to process approximately 63 msgs/second. This lower rate in comparison to the non persistent message processing is due to the fact that messages were processed within a WebSphere MQ unit of work.

The measurement data for this test is shown in Chapter 10, Measurement Data, Section 10.4.2.

3.4.3 Database Read

In this test a message flow consisting of an MQInput node, a database node and an MQOutput node was run. The processing in the database node consisted of a single read of one row selected from a table within an application database.

The transaction mode value on the MQInput node was set to a value of automatic. The coordinated Transaction value for the message flow was set to no.

In this test the application database was small and all database reads were satisfied from database buffers. This was done deliberately in order to illustrate the cost of the call.

The maximum possible message throughput rates were determined for a single instance and single execution group running the message flow. The figure below shows the results that were obtained for varying message size and persistence.

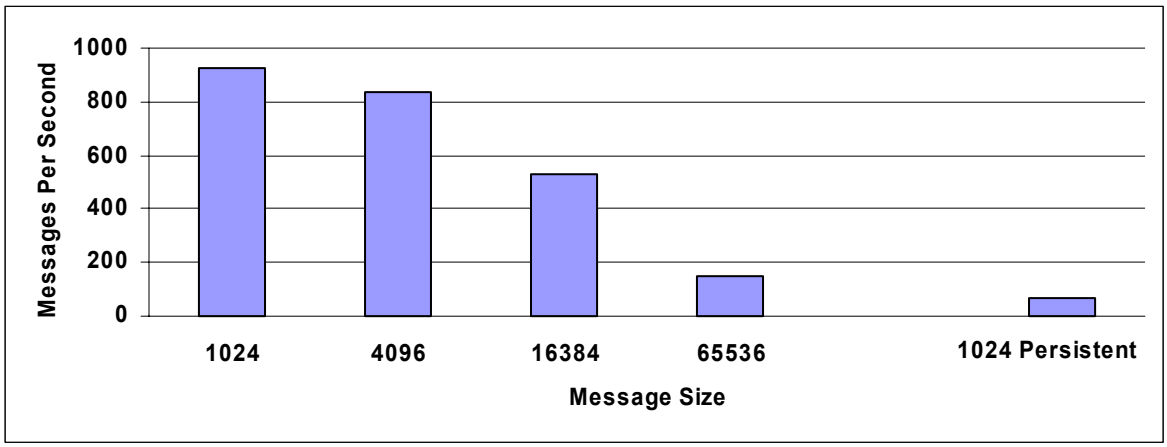

Figure 11: Database Read Throughput Results

With 1KB non persistent messages it was possible to achieve a message throughput rate of approximately 923 msgs/second. The rate of read activity reduced with message size as expected.

With 1KB persistent messages it was possible to process approximately 67 msgs/second. This lower rate in comparison to the non persistent message processing is due to the fact that messages are processed within a WebSphere MQ unit of work with the consequent commit processing which involves I/O processing to the WebSphere MQ queue manager log.

The measurement data for this test is shown in Chapter 10, Measurement Data, Section 10.4.3.

3.5 Filter Node

A Filter node evaluates an ESQL expression against the content of the input message. Based on the result of the expression evaluation the message is propagated to the true terminal if the expression evaluates to true. It is propagated to the false terminal if the expression evaluates to false.

A message flow consisting of an MQInput node, a Filter node and an MQOutput**Error! Bookmark not defined.** node was defined. The Filter node processing involved selecting a message on the basis of the contents of a tag value. The input message contained an MQRFH2 folder with two tags specified following the header. The transaction mode on the MQInput and MQOutput nodes was set to automatic.

The figure below shows the results that were obtained when running the message flow with varying message sizes and persistence. There was a single instance and single execution group running the message flow.

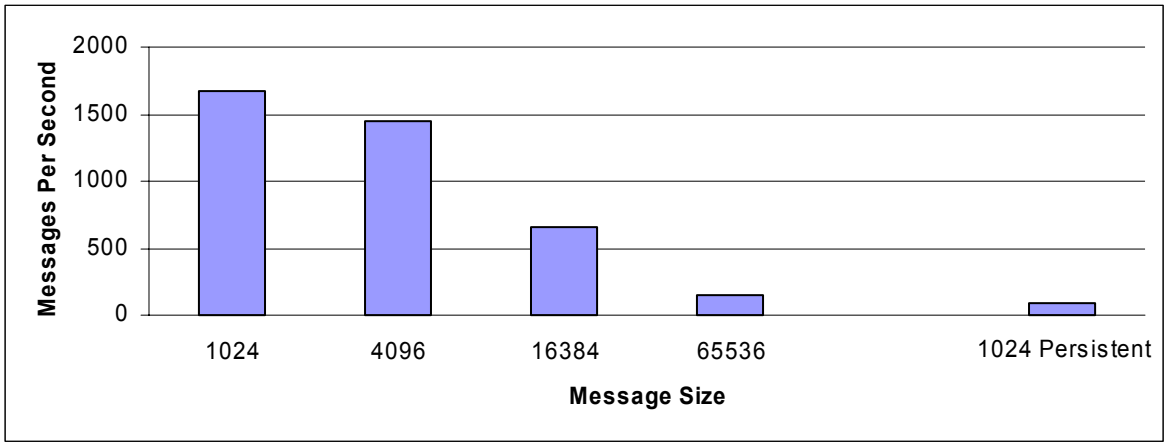

Figure 12: Filter Node Throughput Results

With 1KB non-persistent messages it was possible to run approximately 1670 msgs/second. The cost of the Filter node will vary with the complexity of the filter expression and the number of fields which need to be accessed in the input message and the position of the field being tested in the message. The nearer the beginning of a message that the field to be tested is the lower the parsing cost of accessing it.

With 1KB persistent messages it was possible to process approximately 89 msgs/second. This lower rate in comparison to the non persistent message processing is due to the fact that messages are processed within a WebSphere MQ unit of work with the consequent commit processing which involves I/O processing to the WebSphere MQ queue manager log.

The measurement data for this test is shown in Chapter 10, Measurement Data, Section 10.5.

3.6 FlowOrder Node

The FlowOrder node allows the order of execution within a message flow to be controlled. The node is used to control the order in which the message is propagated to each of two output terminals. The message is propagated to the second output terminal only if propagation to the first terminal is successful.

A measurement was conducted to determine the effect of using the FlowOrder node. The test consisted of two measurements. The first was to establish a base cost, the second to determine the overhead of the FlowOrder node.

The first test case consisted of an MQInput node, two compute nodes and an MQOutput node. The first Compute node contained only a `SET OutputRoot=InputRoot;` statement. The second Compute node contained the processing described in Section 3.3.1, Simple Compute Test. The flow is shown in the figure below.

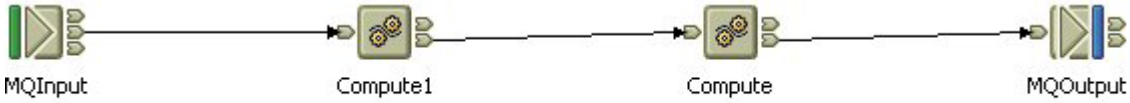

Figure 13 - Comparison Message Flow for the FlowOrder Node.

The second test case consisted of an MQInput node, a FlowOrder node with a Compute node connected to the first output terminal (The only processing in this compute node was a `SET OutputRoot=InputRoot;` statement) and a second Compute node connected to the second output terminal of the FlowOrder node. The processing in the second compute node consisted of the processing described in Section 3.3.1, Simple Compute Test. This message flow is shown in the figure below.

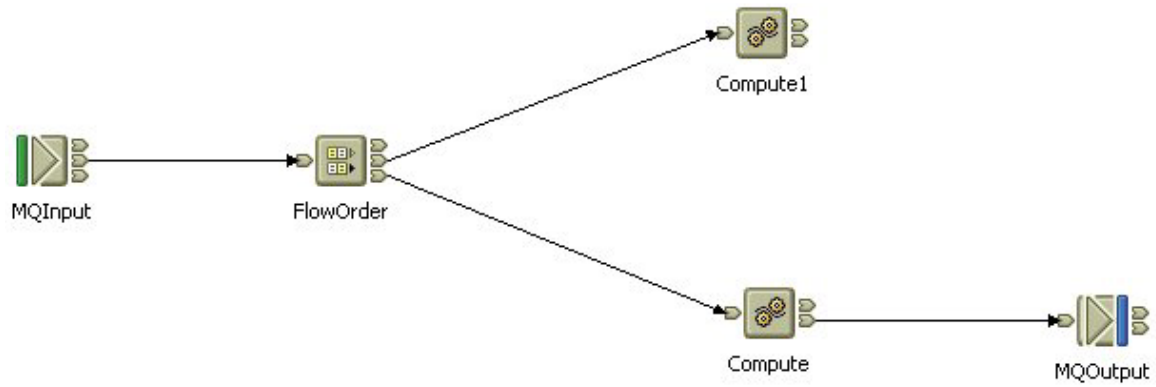

Figure 14 - Message Flow with the FlowOrder Node.

The figure below shows the results that were obtained when running the message flow with varying message sizes and persistence. There was a single instance and single execution group running the message flow.

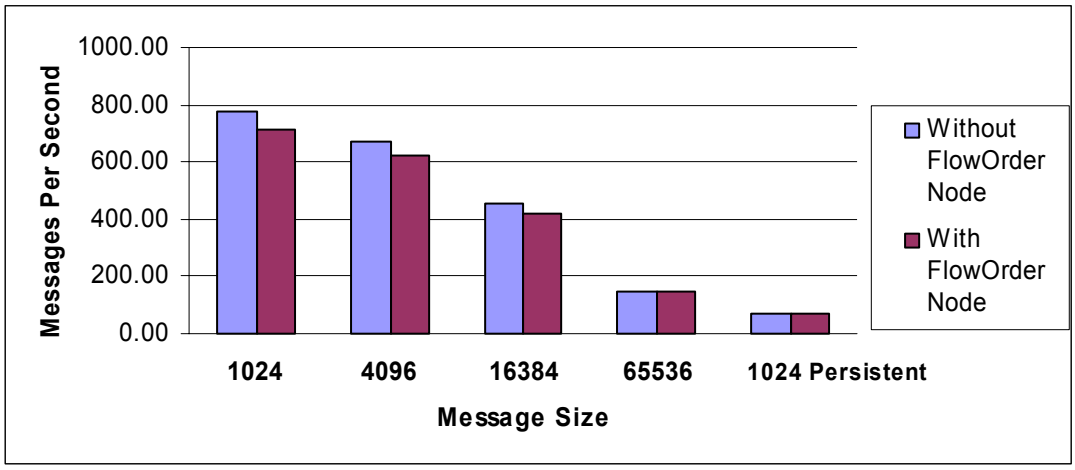

Figure 15: FlowOrder Node Throughput Results

With 1KB non persistent messages it was possible to run approximately 775 msgs/second without a FlowOrder node present and a rate of 716 msgs/second with the FlowOrder node in the message flow. This represents an overhead of 8%. In practice a message flow would typically consist of greater complexity and so the overhead would be lower as a percentage.

With 1KB persistent messages it was possible to run approximately 68 msgs/second without a FlowOrder node present and a rate of 67 msgs/second with the FlowOrder node in the message flow. This represents an overhead of 1.0%.

The measurement data for this test is shown in Chapter 10, Measurement Data, Section 10.6.

3.7 Message Conversion

WBIMB provides the capability to process messages of different formats as well as the ability to convert messages between formats. Throughput measurements were taken to show the effect of using WBIMB to convert messages between each of Generic XML, CWF, MRM XML and MRM TDS formats where Generic XML refers to self-defining XML, CWF denotes a legacy data structure such as a C structure or COBOL copybook, MRM XML refers to the predefined XML used within the MRM and MRM TDS refers to a predefined structure of fields of fixed length or separated by tags within the MRM.

3.7.1 Generic XML, MRM CWF, MRM XML and MRM TDS

The same logical message type was used for each of the conversions. This was a 4KB non persistent message containing 30 input fields, with 10 fields consisting of a short string (12 characters), 10 fields consisting of a floating pointer number, and 10 integer fields.

The format conversion was achieved using a Compute node with suitable ESQL statements. The input messages contained an MQRFH2 header in which the message type was set. The output format was specified in the Compute node processing. Each message format was converted to Generic XML, CWF, MRM XML and MRM TDS. The message throughput achieved with each of the conversions was measured. There was a single execution group running the message flow and no additional instances specified. The results are presented in the Table below.

	Generic XML	**MRM CWF**	**MRM XML**	**MRM TDS**
Generic XML	295.67	173.33	118.33	111.00
MRM CWF	198.00	170.33	119.67	106.33
MRM XML	145.33	124.00	91.33	85.67
MRM TDS	74.33	67.67	58.00	54.67

Table 1: Message Rates in messages per second, When Converting between Different Formats

Even when the output message is set to have the same format as the input message there are still significant costs in processing messages because the messages must be parsed and then reconstructed into the required output format and written as an output message. The cost of converting messages between the two formats is once per message flow and not in each node.

The measurement data for this test is shown in Chapter 10, Measurement Data, Sections 10.7.1 to 10.7.4.

3.8 Publication Node

A publication node may be used within a message flow to represent a point from which messages are "published", which is a point from which messages are transmitted to a set of subscribers who have registered interest in a particular set of messages.

A message flow consisting of an MQInput node and a Publication node was defined. The transaction mode on the MQInput node was set to automatic.

Two sets of measurements were run. The first examined the effect of differing numbers of subscribers when using topic based routing. The second examined the difference in performance between topic and content based routing. These measurements used WebSphere MQ messages for both the publisher and subscriber.

In the throughput measurements each client thread performed the role of publisher and subscriber queue reader. Firstly, an MQPUT was issued to publish a message on the given topic. Secondly, the client thread issued an MQGET to receive the published message.

3.8.1 Topic Based Routing

This processing was performed with different numbers of subscribers, ranging from one to one thousand.

3.8.1.1 One Subscriber

The figure below shows the results that were obtained when running the message flow with varying message sizes and persistence for one subscriber. There was a single instance and single execution group running the message flow. The rates shown are the rate at which messages are being published.

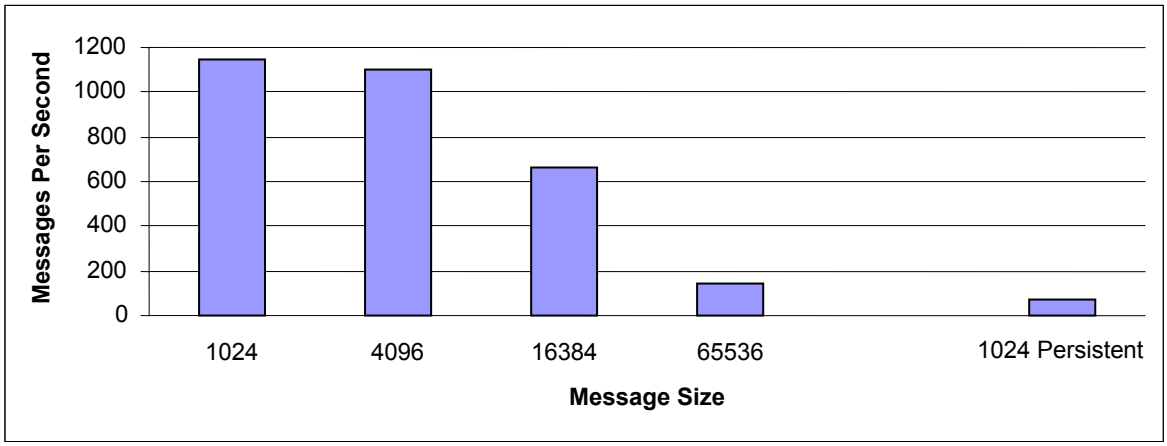

Figure 16: Topic Based Routing with 1 Publisher, 1 Subscriber

With 1KB non-persistent messages it was possible to publish approximately 1150 msgs/second. This can equally be viewed as a subscription rate of 1150 messages per second per subscriber.

As the message size increased, the rate at which messages were published decreased. This is as expected.

With 1KB persistent messages it was possible to process approximately 70 msgs/second. This lower rate in comparison to the non persistent message processing is due to the fact that messages are processed within a WebSphere MQ unit of work with the consequent commit processing which involves I/O processing to the WebSphere MQ queue manager log.

The measurement data for this test is shown in Chapter 10, Measurement Data, Section 10.8.1.

3.8.1.2 Ten, One Hundred and One Thousand SubscribersError! Bookmark not defined.

As an increasing number of subscribers register an interest in receiving published messages on a given topic, so the broker must undertake additional processing to maintain a list of current subscriptions and write a message to each subscribers queue when a message is published.

In order to illustrate the effect of coping with an additional number of subscribers for a given topic additional measurements were taken with 10, 100 and 1000 subscribers. Messages of varying size and persistence were published to a single topic. The results obtained are presented in a graph in the figure below. The X axis shows the number of subscribers. The Y axis shows the number of seconds taken to process a message. It is derived from the reciprocal of the message rate.

Each subscriber requested that published messages be placed on a unique queue. There was a single instance and single execution group running the message flow.

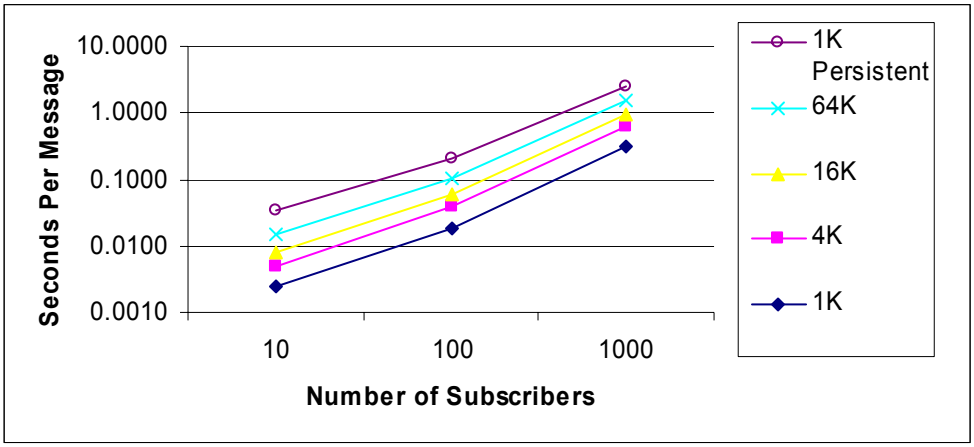

Figure 17: Topic Based Routing with 1 Publisher, 10, 100 and 1000 Subscribers

From the graph it is possible to see that the processing required to deliver messages to the subscribers rises with the increasing number of subscribers. This makes sense since with each additional subscriber there is an additional WebSphere MQ queue to write a message to.

The cost of publishing persistent messages is significantly higher as the processing is dominated by the necessary I/O processing. This is reflected in the steeper gradient of the 1KB persistent message measurements.

When examining the measurement data for varying number of subscribers it is important to understand the way in which the measurement was taken.

For each subscriber which registered to receive publications the published message was written to a queue for that subscriber. With 100 subscribers for example, a single message was written to each of 100 queues. In the measurement environment there was a background program consuming all but one of the published messages. Taking the example of 100 subscribers, 99 of the published messages were consumed by this program. The remaining message was read by the client program emulating the subscriber. In this situation a message count of 1 was registered for the purposes of reporting message rates, although the WBIMB broker had written multiple messages. It is because of this that the reported message rate declines with an increasing number of subscribers, although the level of work performed by the broker is obviously much greater with an increasing number of subscribers.

The measurement data for this test is shown in Chapter 10, Measurement Data, Section 10.8.1.

3.8.2 Content Based Routing

With WBIMB it is possible for a subscriber to register to receive publications based on the contents of a message. This is known as content based routing. This is more complex than topic based routing (in which a subscriber receives all messages on the topic to which they have subscribed) because the message contents must be examined.

In order to determine the cost of using content based routing, a measurement was taken to examine the message rates that could be achieved for both content and topic based routing when delivering messages of the same structure and size.

The figure below shows the message throughput rates that were achieved for content and topic based routing with varying message size and persistence. The topic and content based routing used MQRFH2 messages.

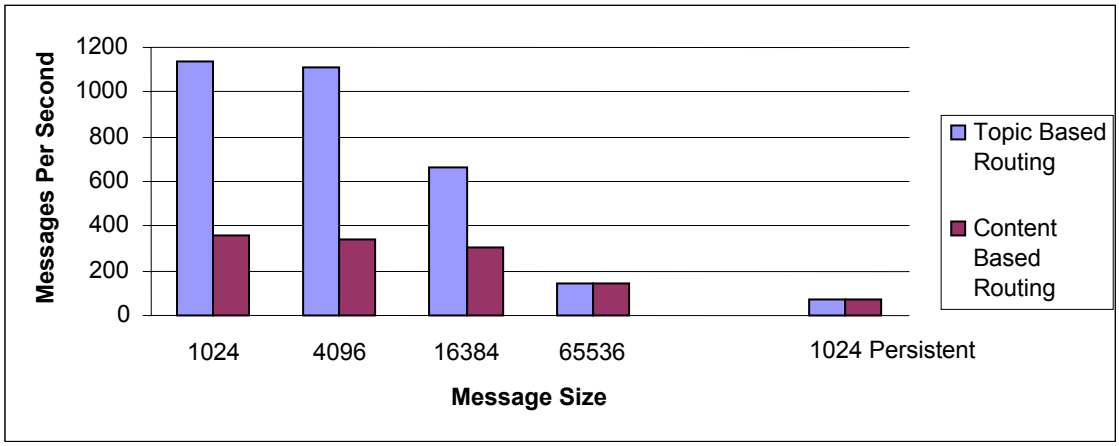

Figure 18: Content Based Routing with 1 Publisher and 1 Subscriber

The graph shows that it is possible to achieve greater message throughput using topic based routing when compared with content based routing. This is as expected since topic based routing is able to publish a message without regard to the message contents. Content based must parse the message contents, and then determine which of the subscribers is to receive the message before finally publishing to those subscribers.

The measurement data for this test is shown in Chapter 10, Measurement Data, Section 10.9.2.

This measurement shows the cost of using content based routing but does not illustrate the benefit which is that the filtering of messages is performed at the broker rather than at the subscriber. An effective filter for the content based routing can potentially save the transmission of many unwanted messages which would occur were topic based routing to be used. This reduction in message traffic will reduce network utilization and processing for the subscribing application.

3.9 RouteToLabel Node

A RouteToLabel node provides a dynamic routing facility based on the contents of the destination list contained within the message. The destination list contains the identity of one or more target Label nodes identified by their Label Name property (not the node name). The RouteToLabel node can be used instead of multiple Filter nodes.

The destination list which is used to control the routing must have been created and included in a previous compute node. Consequently a RouteToLabel node is more expensive to process than a single Filter node, but can be cheaper than many Filter nodes. For a better understanding of how to choose, please read the recommendations in **SupportPac IP04, Designing Message Flows for Performance** which is available at http://www.ibm.com/software/integration/support/supportpacs/individual/supportpacs/ip04.pdf.

The cost of this node is dependent on the size of the destination list. A destination list with 100 entries will require more CPU to process it then a destination list with one entry. The processing cost of the node is not dependent on whether the Route to first or Route to last option is chosen.

The figure below shows the message throughput rates that were achieved for the RouteToLabel node with different sizes of destination list and varying message size and persistence.

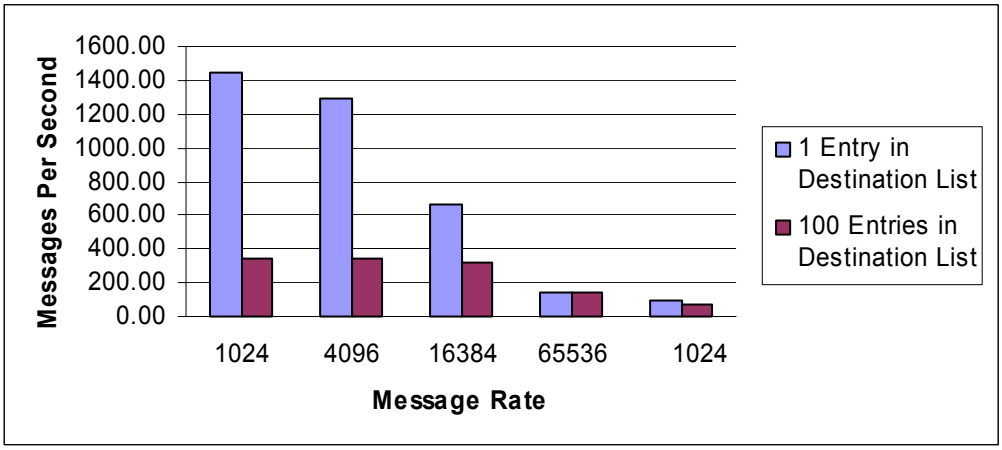

Figure 19: RouteToLabel Throughput with Different Size Destination Lists

With 1KB non-persistent messages it was possible to run approximately 1442 msgs/second when there was only one destination in the list and 340 msgs/sec there was 100 entries in the destination list.

With 1KB persistent messages it was possible to run approximately 90 msgs/second when there was only one destination in the list and 67 msgs/sec when there were 100 entries in the destination list. This lower rate in comparison to the non persistent message processing is due to the fact that messages are processed within a WebSphere MQ unit of work with the consequent commit processing which involves I/O processing to the WebSphere MQ queue manager log.

The measurement data for the RouteToLabel Node Throughput measurements is available in Chapter 10, Measurement Data, Section 10.9.

4 Parallel Processing Options

If the message processing rate which can be achieved with a single copy of a message flow is not sufficient to achieve the target message rate it is likely that you will need to run multiple copies of the message flow concurrently. Within WBIMB on Linux the recommended way of doing this is to run multiple execution groups each processing one or more copies of a message flow. N.B. Running multiple execution groups has a memory overhead associated with it, as each execution group is a separate process with its own allocated memory.

This section shows the effect of running each of these options for the very complex compute message flow which is described in Section 3.3.4, Very Complex Compute Test.

4.1 What Is The Effect Of Using Multiple Execution Groups?

The figure below shows the results that were obtained when running one, two, four and eight copies of one message flow per execution group. The message flow used was the very complex compute test. The transaction mode values on the MQInput and MQOutput node were set to the value of automatic. The same input and output queues were used for all measurements.

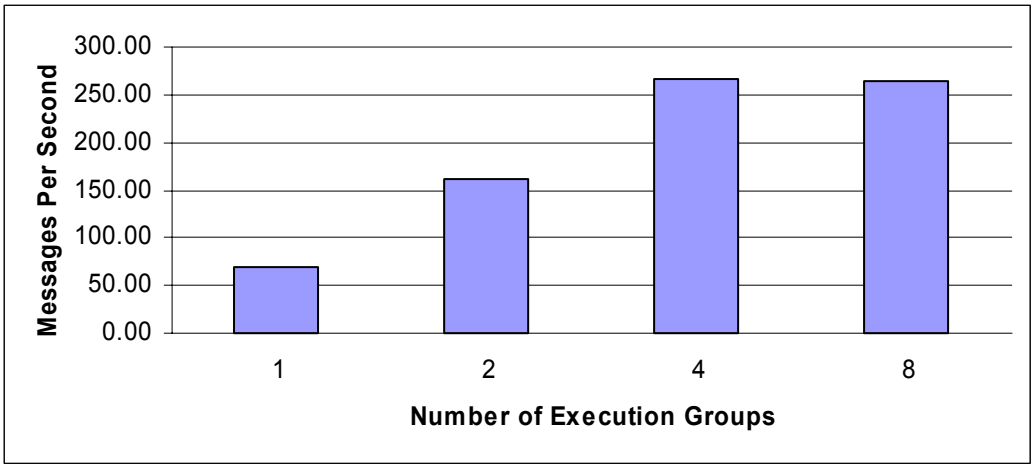

Figure 20: Message Throughput with a Message Flow in Multiple Execution Groups

The graph shows that greater message throughput can be achieved by using a message flow running in each of multiple execution groups. With non persistent 4KB messages and one additional message flow running in a separate execution group (two copies of the message flow in total) it was possible to achieve over 2 times the throughput that was achieved for a single execution group. When using 3 additional message flows, each running in a separate execution group (four copies of the message flow in total) it was possible to achieve 3.5 times the throughput that was achieved with a single execution group. This was not 4 times because the machine was 100% busy at this point. In additional to the message flows there was also the WebSphere MQ threaded channel running which required CPU so meaning that it was not possible to dedicate 100% CPU solely to the running of the message flows.

The measurement with 8 execution groups is a more extreme case of the 4 execution group test. There is not sufficient CPU to support the demand. This results in inefficiencies, hence the lower message rate. This is not a problem with WBIMB performance. It is as a result of an overcommitted machine.

The measurement data showing the effect of using multiple copies of message flow is available in Chapter 10, Measurement Data,10.10.1.

5 Recommended Minimum Specification

This section contains recommendations on the type of hardware on which a WBIMB configuration should be based when running in production. These are only recommendations and are not a substitute for a formal planning and sizing exercise in which requirements are accurately determined. The recommendations are given from a performance perspective and are greater than the minimum specifications needed in order to install and run the product.

For production use it is recommended that the components of WBIMB are allocated over multiple machines with the following purposes:

- One or more machines to support instances of the WebSphere Message Broker Toolkit.
- One machine to support the Configuration Manager. This may also include a WebSphere Message Broker Toolkit.
- One or more machines to support brokers.

Message Broker Toolkit

A recommended minimum machine specification for WebSphere Message Broker Toolkit is a fast uni-processor with 512MB memory.

Configuration Manager

A recommended minimum machine specification for the Configuration Manager is a fast uni-processor with 512MB or more of memory.

Message Broker

The specification of the broker machine is more difficult to determine since it requires knowledge of the expected message rate, the types of nodes that are to be used and the level of transaction control that is used. A recommended minimum specification would be a 2 way processor machine with the fastest possible processors and 512MB memory. The specification may need to be upgraded if message rates are high or there are many execution groups. In such cases more detailed planning would be required. Prototyping and benchmarking should be considered in order to accurately determine resource requirements. The results produced will then be specific and tailored to the individual configuration being built.

If persistent messages are required the use of a disk device with a non volatile fast write cache, through the use of a SAN Volume Controller, is recommended for the device on which the WebSphere MQ queue manager log is located. Where the message rate is less than 25 msgs/second per second fast I/O will improve message response time only. Where the rate is greater than 25 msgs/second then there will be an improvement in message throughput.

A separate disk is also recommended for the WebSphere MQ queue manager queue data. This disk need not have a fast write capability.

If Aggregation nodes or retained publications are used within a message flow it is recommended that the broker database log and data are located on disks with a non volatile fast write cache, through the use of a SAN Volume Controller, in order to minimize I/O times and so maximize message throughput. If the Aggregation nodes or retained publications are not used there is no need to optimize the speed of the broker database I/O.

If business data in a relational database is processed locate the database log and data on dedicated disks. Consider using a fast device such as a disk with non-volatile fast write cache, through the use of a SAN Volume Controller, for the database manager log when there is insert/delete/update activity on the database.

6 Summary of Tuning Information

This section summarizes the performance recommendations which are recorded throughout the report. The recommendations are split into a number of categories, those covering specific products and a final one providing a link to additional information. This information is not a complete guide to product tuning and for further guidance you should consult the relevant product specific documentation.

6.1 WebSphere Queue Manager Tuning

Higher message rates can be obtained with non persistent messages compared with persistent messages. Where possible use non persistent messages.

When processing persistent messages you are recommended to:

- Locate the log of any WebSphere MQ queue manager through which the messages pass on a dedicated disk.
- Locate the WebSphere MQ queue manager log on a very fast disk such as a device with a non volatile cache through the use of a SAN Volume Controller. When using a disk with a fast write cache it is essential that it has a non-volatile capability as the log data is critical to the integrity of your queue manager.

Note that there is no need to locate the WebSphere queue manager queue file on a fast disk. It is advisable to locate it on a dedicated disk in order to improve the efficiency of queue manager checkpoint processing.

When receiving messages over a WebSphere MQ channel you are recommended to use a trusted channel and a trusted listener. In order to do this ensure that the environment variable MQ_CONNECT_TYPE=FASTPATH is present when the channel and listener are started.

6.2 WebSphere Business Integration Message Broker Tuning

Consider the use of a trusted Message Broker. A trusted Message Broker is a broker connecting to the Broker queue manager as a WebSphere MQ trusted application. The effect of doing this is to improve the efficiency with which MQGET and MQPUT operations are performed. However there is a risk of queue manager corruption in the event of broker failure as there is no longer a WebSphere MQ agent process between the MQ application (the broker) and the queue manager. This is a characteristic which any trusted WebSphere MQ application carries and is not unique to the Message Broker. The extent to which a trusted Message Broker will benefit from being a trusted application will depend on the ratio of WebSphere MQ MQGET/MQPUT processing to other processing in the message flow.

When using the aggregation node follow the advice provided in **SupportPac IP05, WebSphere MQ Integrator V2.1 - Optimizing Use of Aggregation Nodes** which is available at http://www.ibm.com/software/integration/support/supportpacs/individual/supportpacs/ip05.pdf

Within a message flow minimize the number of nodes which are used. It is more efficient to use fewer nodes.

When using the RouteToLabel node minimize the size of the destination list which is established.

Use of Topic based routing with Publish/Subscribe is more efficient than Content based routing from a message throughput perspective but may not be most efficient overall. This is because there is no test as to whether the messages being sent are appropriate or wanted. The subscriber may not be interested in the message if the share price is over a given value for example, but with Topic based routing the message would always be sent. Although Content based routing uses more processing to examine the contents of a message and apply a supplied filter it may result in many fewer messages being sent and so result in lower network utilization and client costs.

6.3 Database Manager Tuning

If message flows use Retained publications, have continual subscribing/unsubscribing associated with the use of Publish/Subscribe or use message aggregation you are recommended to

- Locate the log of the Message Broker database on a dedicated disk.
- Locate the log of the Message Broker database on a very fast disk such as a device with a non volatile cache, through the use of a SAN Volume Controller.

6.4 Additional Tuning Information

In order to obtain the maximum message rate for your implementation it is important that you understand the current best practices for WebSphere Business Integration Message Broker. These practices cover the architecture of message flow processing, the coding of message flows as well as the configuration and tuning of the message broker and associated components.

Such information can be found in the Business Integration Zone of WebSphere Developer Domain. A suggested starting place is the article http://www.ibm.com/developerworks/websphere/library/techarticles/0311_dunn2/dunn.html which highlights what information is available and where it may be found.

7 Measurement Hardware and Software

All throughput measurements where taken on a single server machine driven by WebSphere MQ clients running on a separate client machine. WebSphere MQ Clients communicated with the WebSphere MQ queue manager on the server machine using MQI channels.

7.1 Server Machine

The server machine hardware consisted of

- An IBM Netfinity 8500R with 4 * 700Mhz Pentium III Xeon processors.
- Five 33.0 GB SCSI hard drives formatted to use NTFS.
- 4 GB RAM.
- 1Gb Ethernet Card.

The server machine software consisted of:

- Red Hat Linux Advanced Server SMP V 2.1
- WebSphere MQ V5.3.
- WebSphere Business Integration Message Broker for Linux V5 CSD2
- DB2 for Linux V8.1 with FixPack 2.

7.2 Client Machine

The client machine hardware consisted of

- An IBM Netfinity 8500R with 4 * 500Mhz Pentium III Xeon processors.
- Thirteen 2.0 GB SCSI hard drives formatted to use NTFS.
- 1 GB RAM.
- 1Gb Ethernet Card.

The client machine software consisted of:

- Microsoft Windows 2000 with Service Pack 3.
- WebSphere MQ V5.3.

7.3 Network Configuration

The client and server machines were connected using a full duplex 1 Gigabit Ethernet LAN with a single hub.

8 Evaluation Method

This section outlines the configuration of the software components used in the testing and the technique used to obtain the measurement results.

8.1 Input Message Generation

Input messages for the message flows running in the Message Broker where generated using a multi threaded WebSphere MQ Client program written in C. The client program used the Message Queue Interface (MQI). Both persistent and non persistent messages were used in the testing.

Sufficient threads were run in the multi threaded client to ensure that there were always messages on the input queue waiting to be processed. This is important when measuring message throughput.

8.2 Message Content

The composition of the messages which are processed in a message flow can noticeably affect the message throughput which is achieved. Messages which have many small fields will be more costly in CPU terms to process compared with a message of the same size which has fewer larger fields.

The messages used for the tests in this report were simple in nature with a small number of fields. The message conversion tests used messages with 31 fields. The other tests used messages with a smaller number of fields, typically less than 10. In all cases the only difference between the small and larger message sizes in a test was additional padding in the last field of the message.

8.3 Machine Configuration

The program used to generate and consume messages for the message flows was run on a dedicated machine, the Client Machine. The Message Broker, its dedicated WebSphere MQ queue manager and broker database were all located on a dedicated machine, the Server Machine. The figure below shows the configuration of software components and machines.

Both the client and server machine were configured with sufficient memory to ensure that no paging took place during the tests.

Messages were transmitted from the client machine to the server machine over WebSphere MQ SVRCONN channels. The messages were received on the server machine through use of a WebSphere MQ queue manager listener process. This was run as an authorized MQ application in order to improve message throughput. This was achieved by ensuring that the environment variable MQ_CONNECT_TYPE=FASTPATH was present in the environment in which the listener was started.

8.4 Message Broker Configuration

To improve message throughput the Message Broker ran as an authorized WebSphere MQ application. This was achieved by use of the '-t' flag on broker creation (the mqsicreatebroker command) and by ensuring that the environment variable MQ_CONNECT_TYPE=FASTPATH was present in the environment in which the broker was started.

The Message Broker Queue manager log was located on a SCSI disk. Circular logging was used by the Message Broker queue manager. This was for convenience.

Transactional support was used where appropriate. When processing persistent messages it was used, with non persistent messages it was not. The use of transactional was specified on the MQInput and MQOutput nodes for each test. Possible values are yes, no and automatic.

- A value of 'yes' means that the message flow will take place under transaction control. Any derived messages subsequently sent by an MQOutput node in the same instance of the message flow will be sent transactionally unless the MQOutput node has explicitly overridden the use of transaction control.

- A value of 'no' means that the message flow is not under transaction control. Any derived messages subsequently sent by an MQOutput node in the flow will be sent non-transactionally, unless the MQOutput node has specified that the message should be put as part of a transaction.

- A value of 'automatic' means that the message flow will be under transaction control if the incoming message is marked as persistent, otherwise it will not. Any derived messages subsequently sent by an MQOutput node will be sent under transaction control or not, as determined by the persistence on the incoming message, unless the MQOutput node has specifically overridden the use of transaction control.

The use of transaction control means that message processing takes place within a WebSphere MQ unit of work. This involves additional CPU and I/O processing by WebSphere MQ because the unit of work is recoverable. The result is inevitably a reduction in message throughput for both persistent and non persistent messages.

In order to show optimal performance of WBIMB all the throughput measurements in this document used a value of automatic for the transaction parameter unless otherwise specified. This is the recommended value to use for transaction mode unless there is a specific requirement to use a particular value.

There were no error processing or error conditions in the measurements. All messages were successfully passed from one node to another through the out or true terminal. No messages were passed through the failure terminal of a node.

8.5 Database Configuration

The DB2 instance used with the message broker was a default configuration.

8.6 Message Rate

The message rates reported are the number of round trips between the WebSphere MQ multithreaded client and the Message Broker WebSphere MQ queue manager. Another way of viewing it is as the message arrival rate on the input queue for the MQInput node.

9 Compute Node ESQL

This section contains details of the complex, multiple complex and very complex compute nodes.

9.1 Simple Compute Node

The ESQL statements used in the simple compute node are given below.

```
SET OutputRoot=InputRoot;
SET OutputRoot.XML.CSIM.A1='x';
```

9.2 Complex Compute Node

The ESQL statements used in the complex compute node are given below. The variable i has a maximum value of 20.

```
Set OutputRoot=InputRoot;
DECLARE i INTEGER;
DECLARE C INTEGER;
SET C=CARDINALITY(OutputRoot.XML.CSIM.TestCase.Stack.ProcessingPath.Element[]);
SET i = 1;
WHILE i  <= C DO
   SET OutputRoot.XML.CSIM.TestCase.ProcessingPath.Component[i].Name =
       OutputRoot.XML.CSIM.TestCase.Stack.ProcessingPath.Element[i].COMPONENT;
   SET OutputRoot.XML.CSIM.TestCase.ProcessingPath.Component[i].Transport.(XML.attr)Type='A';
   SET OutputRoot.XML.CSIM.TestCase.ProcessingPath.Component[i].Transport.Queue =
OutputRoot.XML.CSIM.TestCase.Stack.ProcessingPath.Element[i].QUEUE;
   SET i = i + 1;
END WHILE;
```

9.3 Multiple Complex Compute Node

The multiple complex compute nodes consisted of five identical complex compute nodes that were daisy chained. The logic within each of the complex compute nodes was the same as that for the complex compute node given in Section 9.2, Complex Compute Node.

9.4 Very Complex Compute Node

The very complex compute node consisted of five repetitions of the logic for complex compute node (see Section 9.2, Complex Compute Node) all contained within one compute node.

9.5 Nested Select From

The ESQL statements used in the nested SELECT test are given below.

```
SET OutputRoot.MQMD = InputRoot.MQMD;
DECLARE style CHARACTER;
SET style = 'Style';
SET OutputRoot.MQMD = InputRoot.MQMD;
DECLARE style CHARACTER;
SET style = 'Style';
SET OutputRoot.XML.CSIM.Data.Statement[] =
(SELECT
        'Monthly' AS (XML.Attribute)Type,
        'Full' AS (0x03000000){style}[1],
        I.Initial || COALESCE(I.Initial[2], '') As Customer.Initials,
        I.Surname AS Customer.Name,
        I.Balance As Customer.Balance,
        (SELECT
                II.Description AS Desc,
                CAST(II.Price AS FLOAT) * 1.6 AS Cost,
                II.Quantity AS Qty
                FROM I.Item[] AS II WHERE II.Price > 0.0
        ) AS Purchases.Article[],
        (SELECT
        SUM( CAST(II.Price AS FLOAT) * CAST(II.Quantity AS FLOAT) * 1.6)
        FROM I.Item[] AS II
        ) AS Amount,'Dollars'  As Amount.(XML.Attribute)Currency
FROM InputRoot.XML.CSIM.Data.Invoice[] AS I WHERE I.Surname <> 'Shop');
```

10 Measurement Data

This appendix contains the measurement data from each of the tests discussed in Section 3, Message Node Processing Profiles. The results of each measurement are presented in a table. The meaning of the column headings is as follows:

Persist: Indicates whether the messages used in the test were persistent or not

Msg Size: The tested message size in bytes (excluding WebSphere MQ header size).

Msgs/sec: The number of round trips or message flow invocations per second

% CPU Busy: System busy CPU percentage. This includes the CPU used by all processes(message broker, WebSphere MQ queue manager, database manager etc) on the system under test.

CPU ms/msg: Overall CPU cost per message , expressed as CPU milliseconds per message. This cost includes WebSphere Business Integration Message Broker, WebSphere MQ, DB2, operating system costs etc.

10.1 A Trivial MQInput/MQOutput Message Flow Results

Persist	Msg Size	Msgs/sec	% CPU Busy	CPU ms/msg
no	1024	3752.67	50.00	0.53
no	4096	2165.00	39.33	0.73
no	16384	661.00	25.00	1.51
no	65536	145.00	20.00	5.52
yes	1024	90.67	4.00	1.76

10.2 Aggregation Node Results

Persist	Msg Size	Msgs/sec	% CPU Busy	CPU ms/msg
no	1024	10.93	5.67	20.73
no	4096	10.73	5.00	18.63
no	16384	10.83	5.67	20.92
no	65536	10.77	6.00	22.29
yes	1024	10.17	6.00	23.61

10.3 Compute Node Results

This section contains the results for the Compute node tests.

10.3.1 Simple Compute Results

Persist	Msg Size	Msgs/sec	% CPU Busy	CPU ms/msg
no	1024	922.00	31.67	1.37
no	4096	813.67	35.00	1.72
no	16384	464.33	36.00	3.10
no	65536	148.00	37.00	10.00
yes	1024	68.67	5.00	2.91

10.3.2 Complex Compute Results

Persist	Msg Size	Msgs/sec	% CPU Busy	CPU ms/msg
no	4096	163.00	28.00	6.87
no	16384	131.33	29.00	8.83
no	65536	67.67	32.67	19.31
yes	4096	53.00	11.00	8.30

10.3.3 Multiple Complex Compute Results

Persist	Msg Size	Msgs/sec	% CPU Busy	CPU ms/msg
no	4096	58.00	27.00	18.62
no	16384	53.00	28.00	21.13
no	65536	37.93	30.33	31.99
yes	4096	40.63	21.00	20.67

10.3.4 Very Complex Compute Results

Persist	Msg Size	Msgs/sec	% CPU Busy	CPU ms/msg
no	4096	70.00	27.00	15.43
no	16384	63.00	28.33	17.99
no	65536	42.63	30.00	28.15
yes	4096	53.00	22.00	16.60

10.3.5 Nested SELECT FROM Compute Results

Persist	Msg Size	Msgs/sec	% CPU Busy	CPU ms/msg
no	4096	201.67	27.00	5.36
no	16384	188.00	29.00	6.17
no	65536	156.00	34.67	8.89
yes	4096	64.00	11.00	6.88

10.4 Database Node Results

This section contains the results for the Database node tests.

10.4.1 Database Insert/Delete Results

Persist	Msg Size	Msgs/sec	% CPU Busy	CPU ms/msg
no	1024	147.00	7.00	1.90
no	4096	146.67	9.00	2.45
no	16384	146.33	12.67	3.46
no	65536	143.00	26.00	7.27
yes	1024	61.67	5.00	3.24

10.4.2 Database Update Results

Persist	Msg Size	Msgs/sec	% CPU Busy	CPU ms/msg
no	1024	163.33	7.00	1.71
no	4096	163.00	8.00	1.96
no	16384	162.33	12.00	2.96
no	65536	150.33	27.00	7.18
yes	1024	63.00	5.00	3.17

10.4.3 Database Read Results

Persist	Msg Size	Msgs/sec	% CPU Busy	CPU ms/msg
no	1024	923.33	31.00	1.34
no	4096	833.67	34.00	1.63
no	16384	531.67	37.00	2.78
no	65536	146.00	26.00	7.12
yes	1024	67.33	5.00	2.97

10.5 Filter Node Results

Persist	Msg Size	Msgs/sec	% CPU Busy	CPU ms/msg
no	1024	1669.67	35.67	0.85
no	4096	1454.67	41.00	1.13
no	16384	660.00	37.67	2.28
no	65536	144.67	24.00	6.64
yes	1024	88.67	5.00	2.26

10.6 FlowOrder Node Results

This section contains the results for the FlowOrder Node test.

10.6.1 FlowOrder Node First Test Results

This test was to establish the base cost of the message flow before using a FlowOrder node.

Persist	Msg Size	Msgs/sec	% CPU Busy	CPU ms/msg
no	1024	775.00	30.00	1.55
no	4096	669.67	32.00	1.91
no	16384	453.00	36.00	3.18
no	65536	148.33	35.33	9.53
yes	1024	68.00	4.00	2.35

10.6.2 FlowOrder Node Second Test Results

This test was to measure message throughput with the FlowOrder node in use.

Persist	Msg Size	Msgs/sec	% CPU Busy	CPU ms/msg
no	1024	716.33	30.00	1.68
no	4096	619.00	31.33	2.02
no	16384	418.00	35.00	3.35
no	65536	148.00	36.00	9.73
yes	1024	67.33	4.00	2.38

10.7 Message Format Conversion Results

This section contains the results for the message conversion tests.

10.7.1 Generic XML

This section contains the results for the conversion of Generic XML messages to different formats.

10.7.1.1 Generic XML to Generic XML

Persist	Msg Size	Msgs/sec	% CPU Busy	CPU ms/msg
no	4096	295.67	28.00	3.79
no	16384	203.67	29.67	5.83
no	65536	88.33	34.33	15.55
yes	4096	54.00	7.00	5.19

10.7.1.2 Generic XML to MRM CWF

Persist	Msg Size	Msgs/sec	% CPU Busy	CPU ms/msg
no	4096	173.33	27.00	6.23
no	16384	141.00	29.00	8.23
no	65536	68.33	31.67	18.54
yes	4096	54.00	10.00	7.41

10.7.1.3 Generic XML to MRM XML

Persist	Msg Size	Msgs/sec	% CPU Busy	CPU ms/msg
no	4096	118.33	27.00	9.13
no	16384	93.00	28.00	12.04
no	65536	47.40	30.33	25.60
yes	4096	53.00	13.00	9.81

10.7.1.4 Generic XML to MRM TDS

Persist	Msg Size	Msgs/sec	% CPU Busy	CPU ms/msg
no	4096	111.00	26.00	9.37
no	16384	91.33	28.00	12.26
no	65536	50.27	30.00	23.87
yes	4096	53.33	14.67	11.00

10.7.2 MRM CWF

This section contains the results for the conversion of CWF messages to different formats.

10.7.2.1 MRM CWF to Generic XML

Persist	Msg Size	Msgs/sec	% CPU Busy	CPU ms/msg
no	4096	198.00	27.00	5.45
no	16384	152.33	29.33	7.70
no	65536	73.33	32.67	17.82
yes	4096	53.33	9.00	6.75

10.7.2.2 MRM CWF to MRM CWF

Persist	Msg Size	Msgs/sec	% CPU Busy	CPU ms/msg
no	4096	170.33	27.00	6.34
no	16384	138.67	29.00	8.37
no	65536	71.00	32.00	18.03
yes	4096	53.33	10.33	7.75

10.7.2.3 MRM CWF to MRM XML

Persist	Msg Size	Msgs/sec	% CPU Busy	CPU ms/msg
no	4096	119.67	26.00	8.69
no	16384	97.00	28.00	11.55
no	65536	51.00	31.00	24.31
yes	4096	53.00	13.00	9.81

10.7.2.4 MRM CWF to MRM TDS

Persist	Msg Size	Msgs/sec	% CPU Busy	CPU ms/msg
no	4096	106.33	26.00	9.78
no	16384	91.00	28.00	12.31
no	65536	53.00	30.33	22.89
yes	4096	53.00	15.33	11.57

10.7.3 MRM XML

This section contains the results for the conversion of MRM XML messages to different formats.

10.7.3.1 MRM XML to Generic XML

Persist	Msg Size	Msgs/sec	% CPU Busy	CPU ms/msg
no	4096	145.33	27.00	7.43
no	16384	116.33	28.00	9.63
no	65536	58.33	30.67	21.03
yes	4096	53.33	11.00	8.25

10.7.3.2 MRM XML to MRM CWFError! Bookmark not defined.

Persist	Msg Size	Msgs/sec	% CPU Busy	CPU ms/msg
no	4096	124.00	27.00	8.71
no	16384	103.33	27.33	10.58
no	65536	56.67	31.00	21.88
yes	4096	53.67	13.67	10.19

10.7.3.3 MRM XML to MRM XML

Persist	Msg Size	Msgs/sec	% CPU Busy	CPU ms/msg
no	4096	91.33	27.00	11.82
no	16384	76.00	28.00	14.74
no	65536	40.53	29.33	28.95
yes	4096	53.00	16.33	12.33

10.7.3.4 MRM XML to MRM TDS

Persist	Msg Size	Msgs/sec	% CPU Busy	CPU ms/msg
no	4096	85.67	26.00	12.14
no	16384	73.00	28.00	15.34
no	65536	43.70	29.00	26.54
yes	4096	53.33	18.33	13.75

10.7.4 MRM TDS

This section contains the results for the conversion of TDS messages to different formats.

10.7.4.1 MRM TDS to Generic XML

Persist	Msg Size	Msgs/sec	% CPU Busy	CPU ms/msg
no	4096	74.33	26.00	13.99
no	16384	38.40	26.00	27.08
no	65536	12.70	27.00	85.04
yes	4096	53.00	20.00	15.09

10.7.4.2 MRM TDS to MRM CWF

Persist	Msg Size	Msgs/sec	% CPU Busy	CPU ms/msg
no	4096	67.67	26.00	15.37
no	16384	36.70	26.00	28.34
no	65536	12.70	27.00	85.04
yes	4096	53.00	22.00	16.60

10.7.4.3 MRM TDS to MRM XML

Persist	Msg Size	Msgs/sec	% CPU Busy	CPU ms/msg
no	4096	58.00	26.00	17.93
no	16384	33.50	26.00	31.04
no	65536	11.73	26.67	90.91
yes	4096	41.37	20.00	19.34

10.7.4.4 MRM TDS to MRM TDS

Persist	Msg Size	Msgs/sec	% CPU Busy	CPU ms/msg
no	4096	54.67	26.00	19.02
no	16384	32.47	26.00	32.03
no	65536	11.67	26.33	90.29
yes	4096	40.73	20.00	19.64

10.8 Publication Node Results

This section contains the results for the topic and content based routing publication node tests.

10.8.1 Topic Based Routing

This section contains the results for the topic based publish/subscribe tests.

10.8.1.1 One Subscriber Results

Persist	Msg Size	Msgs/sec	% CPU Busy	CPU ms/msg
no	1024	1150.00	29.67	1.13
no	4096	1106.33	36.67	1.33
no	16384	659.00	36.00	2.19
no	65536	145.33	23.00	6.33
yes	1024	69.67	5.00	2.87

10.8.1.2 Ten Subscribers Results

Persist	Msg Size	Msgs/sec	% CPU Busy	CPU ms/msg
no	1024	417.67	36.67	3.51
no	4096	402.00	39.00	3.88
no	16384	329.00	44.00	5.35
no	65536	139.67	57.00	16.32
yes	1024	53.00	10.67	8.05

10.8.1.3 One Hundred Subscribers Results

Persist	Msg Size	Msgs/sec	% CPU Busy	CPU ms/msg
no	1024	54.00	36.00	26.67
no	4096	51.00	36.33	28.50
no	16384	44.30	40.00	36.12
no	65536	23.07	47.33	82.08
yes	1024	10.00	15.33	61.33

10.8.1.4 One Thousand Subscribers Results

Persist	Msg Size	Msgs/sec	% CPU Busy	CPU ms/msg
no	1024	3.24	32.00	395.06
no	4096	3.13	32.00	408.95
no	16384	2.87	34.00	473.32
no	65536	1.93	42.67	882.76
yes	1024	1.01	19.00	752.48

10.8.2 Content Based Routing

Persist	Msg Size	Msgs/sec	% CPU Busy	CPU ms/msg
no	1024	360.67	28.33	3.14
no	4096	344.67	29.67	3.44
no	16384	301.67	34.00	4.51
no	65536	146.00	33.33	9.13
yes	1024	68.00	8.00	4.71

10.9 RouteToLabel Node Results

This section contains the results for the RouteToLabel node tests.

10.9.1 RouteToLabel with One Entry in the Destination List

Persist	Msg Size	Msgs/sec	% CPU Busy	CPU ms/msg
no	1024	1442.33	35.00	0.97
no	4096	1290.33	38.00	1.18
no	16384	660.33	33.33	2.02
no	65536	145.33	22.00	6.06
yes	1024	90.00	5.00	2.22

10.9.2 RouteToLabel with One Hundred Entries in the Destination List

Persist	Msg Size	Msgs/sec	% CPU Busy	CPU ms/msg
no	1024	339.67	27.67	3.26
no	4096	341.00	29.00	3.40
no	16384	317.00	33.67	4.25
no	65536	146.33	31.00	8.47
yes	1024	67.33	7.00	4.16

10.10 Parallel Processing Options

This section contains the results for the parallel processing options. This is using multiple execution groups to increase message throughput.

10.10.1 Using Multiple Execution Groups

10.10.1.1 Message Throughput with One Execution Group

Persist	Msg Size	Msgs/sec	% CPU Busy	CPU ms/msg
no	4096	70.00	27.00	15.43
no	16384	63.00	28.33	17.99
no	65536	42.63	30.00	28.15
yes	4096	53.00	22.00	16.60

10.10.1.2 Message Throughput with Two Execution Groups

Persist	Msg Size	Msgs/sec	% CPU Busy	CPU ms/msg
no	4096	161.33	52.00	12.89
no	16384	141.00	53.33	15.13
no	65536	89.67	57.00	25.43
yes	4096	77.33	27.00	13.97

10.10.1.3 Message Throughput with Four Execution Groups

Persist	Msg Size	Msgs/sec	% CPU Busy	CPU ms/msg
no	4096	266.00	100.00	15.04
no	16384	227.67	100.00	17.57
no	65536	133.00	99.33	29.87
yes	4096	126.00	51.00	16.19

10.10.1.4 Message Throughput with Eight Execution Groups

Persist	Msg Size	Msgs/sec	% CPU Busy	CPU ms/msg
no	4096	265.33	100.00	15.08
no	16384	224.33	100.00	17.83
no	65536	129.67	99.33	30.64
yes	4096	162.00	66.00	16.30

Index

End of Document