IBM WebSphere MQ Integrator for z/OS V2.1 - Performance Report

Version 1.1

April 19, 2002

Andy Abbey

Les Churchard

MQSeries Performance and Test
IBM UK Laboratories
Hursley Park
Winchester
Hampshire
SO21 2JN

Take Note!

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

Second edition, April 2002

This edition applies to Version 1.1 of *IBM WebSphere MQ Integrator for z/OS - V2.1 Performance Report* and to all subsequent releases and modifications until otherwise indicated in new editions.

Notices

This report is intended to help the customer understand the performance characteristics and perform capacity planning for WebSphere MQ Integrator for z/OS - V2.1. The information is not intended as the specification of any programming interfaces that are provided by MQSeries or WebSphere MQ Integrator for z/OS - V2.1.

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
- z/OS
- MQSeries
- WebSphere MQ Integrator
- DB2

The following terms are trademarks of other companies:

- Windows NT, Windows 2000, Visual Studio Microsoft Corporation
- NEONFormatter, NEONRules, NEON

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

Summary of Amendments

Date	Changes
20 December 2001	Initial release
19 April 2002	Update with data from evaluation of V2.1 code at CSD2 level

Contents

1.0 CONCEPTS

WebSphere MQ Integrator (WMQI) is IBMs' message broker product, addressing the needs of business and application integration. Business integration is the coordination of all of a company's processes. Application integration is the coordination of its applications. This process of integration involves the bringing together of the data and processes within an organization to maximize the sharing of data and applications in order to cope with changing organization structure (merger, acquisition etc.) and increase the effectiveness of the organization.

A key requirement of such business and application integration is that applications are able to communicate with each other without having to make code changes. WMQI V2.1 makes the required integration easier through the services that it provides. These services are:

- Route a message to several destinations, using rules that act on the contents of one or more of the fields in the message or message header.
- Transform a message, so that applications using different formats can exchange messages in their own formats.
- Store and retrieve a message, or part of a message, in a database.
- Modify the contents of a message (for example, by adding data extracted from a database).
- Publish a message to make it available to other applications. Other applications can choose to receive publications that relate to specific topics, have specific content, or both.
- Extend the capabilities of rules and formats defined in MQSeries Integrator V2.

The above services are based on the messaging transport services provided by the MQSeries Messaging products.

For a full description of the concepts introduced in this chapter refer to manual *WebSphere MQ Integrator Introduction and Planning*.

1.1 Major Components

The major components of WMQI V2.1 are:

- The Configuration Manager
- The Brokers
- The Control Center.
- The User Name Server.

The Configuration Manager and the Control Center must be run on a Windows NT or Windows 2000 machine. Brokers and the User Name Server run on any of the supported operating systems, including z/OS.

1.1.1 The Configuration Manager

The Configuration Manager is the main component of the WMQI environment. The components and resources managed by the Configuration Manager constitute the broker domain. The Configuration Manager serves three main functions:

- It maintains configuration details in the configuration repository. This is a set of database tables that provide a central record of the broker domain components.

- It manages the initialization and deployment of brokers and message processing operations in response to actions initiated through the Control Center. It communicates with other components in the broker domain using MQSeries transport services.

- It checks the authority of defined user Ids to initiate those actions.

There is a single Configuration Manager to manage a broker domain. The Configuration Manager provides a service to other components in the broker domain providing them with configuration updates in response to actions taken by the user of the Control Center.

1.1.2 The Brokers

The broker is a named resource that hosts and controls the business processes that are defined as message flows. Applications send new messages to the message flow and receive processed messages from the message flow, using MQSeries queues and connections.

Any number of brokers can be created within a broker domain. It is possible to create more than one broker on any one physical system if desired, but there must be a unique queue manager for each broker. It is possible for a single broker to share a queue manager with the Configuration Manager.

Within each broker it is possible to define execution groups that are responsible for running the message flows. An execution group is implemented as an operating system process. Within an execution group it is possible to define additional threads that will also perform the processing of the message flows, these are known as additional instances. On z/OS brokers run in the Unix System Services (USS) environment.

When creating message flows that provide a publish/subscribe service it is possible to connect a number of brokers in a collective using the Control Center. A collective contains a number of brokers that are all physically interconnected. All the broker queue managers must be connected by pairs of MQSeries channels.

A collective optimizes the publish/subscribe of messages in the broker domain by reducing the number of clients per broker, without increasing the hops taken by any message by more than one. In this way collectives are more efficient than a hierarchy.

It is possible to connect collectives to other collectives and to other individual brokers. When collectives are connected to a standalone broker only one broker in each collective must provide the connection.

Messages published to any one broker are propagated to all connected brokers (whether or not they are in a collective) to which an application has subscribed to the messages topic or content.

1.1.3 The Control Center

The Control Center interfaces with the Configuration Manager to allow the user to configure and control the broker domain. The Control Center and Configuration Manager exchange messages (using MQSeries) to provide the information requested and to make updates to the broker domain configuration.

It is possible to install and invoke any number of Control Center instances. The Control Center can be installed one the same physical system as the Configuration Manager, or any other system that can connect to the Configuration Manager.

The Control Center is structured as a number of views on the configuration and message repositories. The message repository contains all message definitions that have been created or imported through

the Control Center. The configuration repository contains configuration information pertaining to all other resources within the broker domain; brokers, collectives, message processing nodes, message flows, topics and subscriptions.

The Control Center can be used to

- Develop, modify, assign and deploy message flows.
- Develop, modify, assign and deploy message sets.
- Define the broker domain topology and create collectives.
- Control topic security of messages by topic.
- View status information.

1.1.4 The User Name Server

The User Name Server monitors the underlying security subsystem provided by the operating system and provides information about the valid principals (users and groups of users) in the system. The User Name Server shares this information with the brokers and Configuration Manager and updates it at frequent intervals. The information can be used to control access to topic-based messages produced by the publish/subscribe service. Topic-based security gives the ability to control the authority of applications, identified by the user ID under which they are executing, to publish on topics, to subscribe to topics and to request persistent delivery of messages on topics.

In WMQI V2.1 it is recommended that the User Name Server is only used with a small number of userids.

1.2 Message Flows

A message flow is a sequence of operations on a message, performed by a series of message processing nodes. The actions are defined in terms of the message format, its content, and the results of individual actions along the message flow.

MQSeries Integrator supplies a number of predefined message processing node types, known as IBM primitives. These provide basic functions including input, output, filter (on message data content), and compute (manipulate message content: for example, add data from a database).

A message flow and the message processing nodes it contains describe the transformation and routing applied to an incoming message to transform it into outgoing messages. These actions form the rules by which the message is processed.

A message flow can also be made up of a sequence of other message flows, that are joined together. This function allows message flows to be defined and reused in other message flows when required.

When the message flow creation is complete, it can be assigned for execution to one or more brokers. The message flow must be operationally complete. That is it must contain at least an MQInput node. Most message flows will also contain at least one MQOutput or one Publication node, although this is not required.

A message flow can be defined as transactional: it is possible to define message flows to perform all processing within a single unit of work. Therefore the receipt of every message by the input node, and the database operations performed as a result of that message being received and processed by the message flow, are coordinated.

If an error occurs within a transactional message flow, the transaction is rolled back and the message will be handled according to normal error handling rules. It is possible to define a message flow to work outside of a unit of work if this transactional support is not required.

When a message flow is deployed to a broker, the broker automatically starts an instance of the message flow for each input node (one or more). This is the default behaviour. Each instance retrieves a message from the input node, and runs in parallel with other instances that retrieve a message from other input nodes.

The broker provides the run-time environment for a set of deployed message flows: this environment is called an execution group. An execution group provides an isolated environment, because each is started as a separate operating system process.

One execution group, the default execution group, is set up for use whenever a broker is created. By setting up additional execution groups, it is possible to isolate message flows that handle sensitive data such as payroll records or security information, from other non-senstive message flows.

In order to further increase the throughput of the message flow, it is possible to set a property of the assigned message flow that defines how many additional instances are to be started by the broker for that message flow. It is possible to set the *orderMode* property of the input node to exercise control over the order in which messages are processed.

It is possible to increase message flow throughput by assigning more than one copy of the message flow to the same broker, and by assigning a message flow to more than execution group. This is only appropriate if the message order is not important because the multiple copies of the message flow are handled independently by the broker with no correlation between them.

Within an execution group the assigned message flows run in different WMQI thread pools. The size of the thread pool that is assigned for each message flow is set by specifying the number of additional instances of each message flow.

1.3 Messages and Message Sets

In WMQI messages are always in one of two broad categories:

- Predefined. The content of a predefined message is described by the message template.
- Self-defining. The content of a self-defining message is described by the message itself.

The message definition process is managed by the Message Repository Manager (MRM) component of the Control Center.

Message definitions are created or modified using the Control Center, the MRM stores them in the message repository.

Predefined Messages

A predefined message has a logical structure and a physical structure.

The logical structure defines the contents of the message using a tree structure that identifies each field and its relation to other fields. The applications sending and receiving messages like this understand the format and type of each field. For example they might use a C structure that shows AccountNumber is an eight byte field, AccountName is a 20 byte character field and AccountBalance is an 8 byte character field.

The physical structure, also known as a wire format, is a string of bytes. Without the logical structure the physical structure has no intrinsic meaning.

The physical structure of each element in a message is further defined by its Custom Wire Format(CWF) characteristics. These give the physical format (for example COBOL packed decimal), the length, whether the field is signed and so on.

Message Templates

A message template is made up of four values contained within the <Msd> element of the <mcd> folder in the RFH2 header:

1. *Message Domain* which identifies the message parser that will interpret the bit-stream of the message. By default WMQI supports the values MRM, XML, BLOB and NEON. MRM is the MRM-enabled parser and is used for all messages whose definitions have been created in, or imported to, the message repository. XML is for self defining messages only. BLOB is used for messages whose format is not understood, in which case they are treated as a bit stream. NEON is for NEON messages only.

2. *Message set* which identifies the grouping of messages within the message domain. Typically a message set contains a number of related messages that provide the definitions required for a specific business task or application suite. The message set is similar in concept to the application group in NEON.

3. *Message type* which identifies the logical structure of the data in the message. For example the number and location of character strings and their relationships.

4. *Message format* which identifies the physical representation of the message (its wire format). The MRM-enabled parser supports the following wire formats:

   XML the message is identified as an XML document that complies with a Document Type Descriptor (DTD) that can be generated for a message by the MRM. This option does not apply to self-defining messages, which have the domain XML, rather than MRM.

   CWF denotes legacy data structures used in common programming languages (C or COBOL). Data structures for CWF messages are typically imported into the message repository.

The message format value is only valid for predefined messages and not self-defining messages.

Self-defining messages

Self-defining messages use the XML standard to structure their content. They can be used in any message flow, and are supported by all message flow nodes.

Self-defining messages do not have to be defined to the Control Center, nor do they have to be assigned to brokers to ensure that they can be interpreted.

Self defining messages are said to use generic XML.

When a message is processed in a message flow, its format must be determined first so that the correct parser is used. The message characteristics are identified by the input node of a message flow in one of two ways:

- For messages with an MQRFH or MQRFH2 architected header, the input node checks the value in the message header.

- For messages that do not have an MQRFH or MQRFH2 header, the input node uses the default message template, defined as a properly of the input node, to determine how the message must be parsed.

1.4 Message Parsers

WMQI can handle any message template for which a suitable parser is available. The parsers interact with the message templates stored in the message dictionaries. The range of messages supported can be extended by creating your own message parsers.

Message parsers are provided for :

- Predefined XML. Such messages have an MQRFH or MQRFH2 header.
- The standard MQSeries headers: MQCIH, MQDLH, MQIIH, MQMD, MQMDE, MQRFH, MQRFH2, MQRMH, MQSAPH ,MQCFH, and MQWIH.
- Record-orientated C and COBOL language structures.
- Self-defining (generic XML) messages.
- Messages whose formats are defined in the NEON dictionary (These messages are defined using the NEON interface not the Control Center).

If no parser can be identified for a message, WMQI treats it as a binary object that passes, of necessity, unaltered through any message flow. However such a message can be stored in a database, be routed according to topic, and have headers added or removed.

WMQI V2.1 provides a function that allows messages to be transformed from one format to another.

2.0 BROKER THROUGHPUT MEASUREMENTS

In order to understand the processing characteristics of WMQI V2.1 a number of performance measurements have been taken using multiple aspects of the product. The test cases used have been deliberately made trivial in order to be able to report the cost of using WMQI V2.1, rather than to report the cost of running a particular application. It is very difficult to accurately represent what might be considered a typical application since the business logic is always enterprise specific.

The effect of the queue manager has been minimized where possible. This has meant using predominantly non persistent messages.

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, what are the relative costs of the different node types in terms of CPU and memory usage.

In the throughput measurements the following node types have been measured:

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

Refer to manual *WebSphere MQ Integrator Using The Control Center* for further information on node types.

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. Some node types have been measured in more than one configuration in order to investigate the various configuration effects, such as running multiple execution groups. 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 within a single message flow. This should be borne in mind when performing capacity planning.

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

All measurements were conducted in the same measurement environment. This is described in Section 6.0 APPENDIX A - MEASUREMENT HARDWARE AND SOFTWARE.

All measurements were driven by an MQSeries program written in C which put messages to the MQInput node queue, then used get-with-wait for any message on the MQOutput node queue. The program ran on the same z/OS image as WMQI in 15 concurrent jobs, all sending the same message format and content. Experimentation showed that on our test system 15 driving jobs kept the broker sufficiently busy. The Message Queue Interface programming interface was used to write and read messages.

There was 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.

The message rates reported are the number of roundtrips between the MQSeries programs and the MQSeries queue manager to which the input data is written and from which the reply data is read.

For an MQInput and MQOutput node it is possible to define transaction support for a node using the *transactionMode* property. 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 messageflow 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 an MQSeries unit of work. This involves additional CPU and I/O processing by MQSeries 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 WMQI V2.1 all the throughput measurements in this document used a value of automatic for the transaction parameter unless otherwise specified,.

2.1 MQInput/MQOutput Throughput

A message flow consisting of a single MQInput and MQOutput node represents a very simple message flow. Measuring the throughput achievable with such a message flow shows the maximum message rate that can be achieved using WMQI V2.1 to move messages between MQSeries 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.

Figure 1 below shows the results that were obtained as a result of running the message flow with varying message sizes and persistence. There was a single instance and single execution group running the message flow.

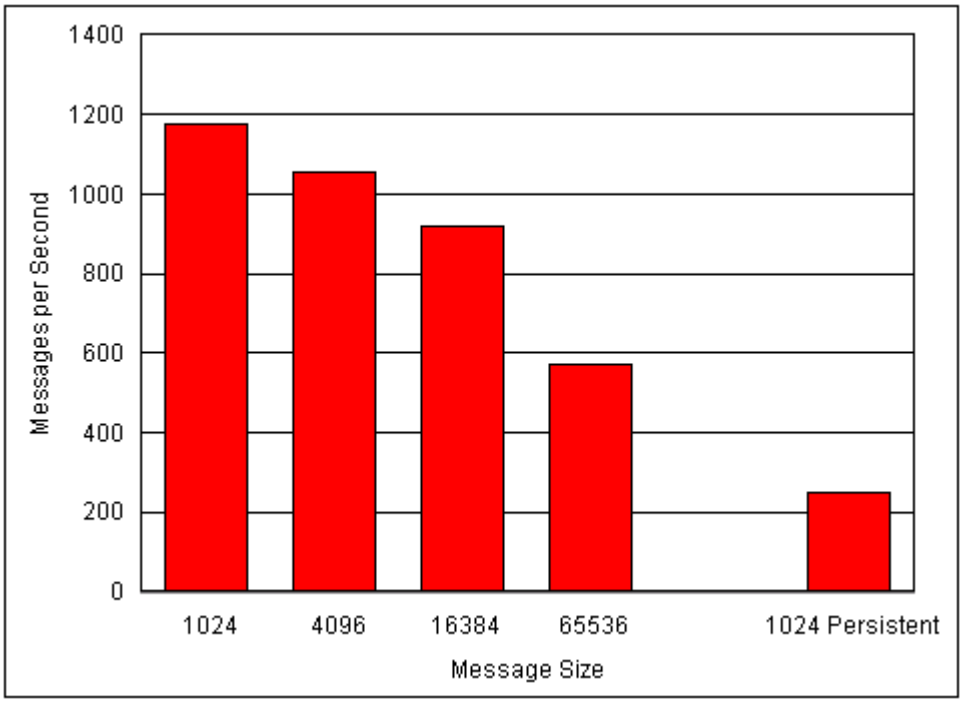

Figure 1: MQInput/MQOutput Throughput Results

With a 1K non persistent message it was possible to process 1176 msgs/second. Increasing the message size to 4K had an effect on the throughput rate achieved (1057 msgs/second), but the ratio of decrease was far less than the ratio of increase of extra data being handled. This is consistent with the behaviour that is observed when using other applications within MQSeries.

Increasing the message size to 16K and beyond had a significant effect on the maximum message throughput that could be achieved. This decrease in throughput is as a result of the additional volume of data that must be managed both within WMQI V2.1 and the associated MQSeries queue manager.

As the input message was non persistent there was no transactional control. We are, therefore observing the maximum rate at which WMQI V2.1 is able to transfer messages from the input queue to the output queue for a single execution group. Adding additional execution groups, flows or instances allows greater throughput to be achieved.

The use of persistent messages had a significant effect on the maximum message throughput rate that was achievable. For a 1K persistent message the message rate was 248 msgs/second.

When persistent messages are used there are two additional effects that dominate the maximum message throughput rate achievable:

1. Any messages read from or written to an MQSeries queue now take place under MQSeries transaction control

2. The MQSeries queue manager must make the message persistent, which involves a synchronous write to the MQSeries log. Sufficient queue manager buffers were defined such that there was no I/Os to the page sets.

As a result of the additional disk I/O required the message rate becomes dominated by I/O processing and is no longer CPU bound. The message rate that is achievable is totally dependent on the speed of the I/O device on which the MQSeries log is located.

The detailed measurement data for the MQInput/MQOutput throughput measurements is available in **Section 7.1 - MQInput/MQoutput Throughput Results**.

2.2 Compute Node Throughput

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 varying levels of user specified processing in order to illustrate these effects.

Each test case consisted of an MQInput and MQOutput node with varying numbers of compute nodes in between. The level of complexity in the compute nodes was also varied. 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 WMQI V2.1 optimises the process and simply repeats the input message which can give an over optimistic 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.

In these measurements the transaction mode on the MQInput and MQOutput nodes was set to automatic.

2.2.1 Simple Compute Node

Figure 2 below shows the results that were obtained as a result of running the simple compute node with varying message sizes and persistence. There was a single instance and single execution group

running the message flow.

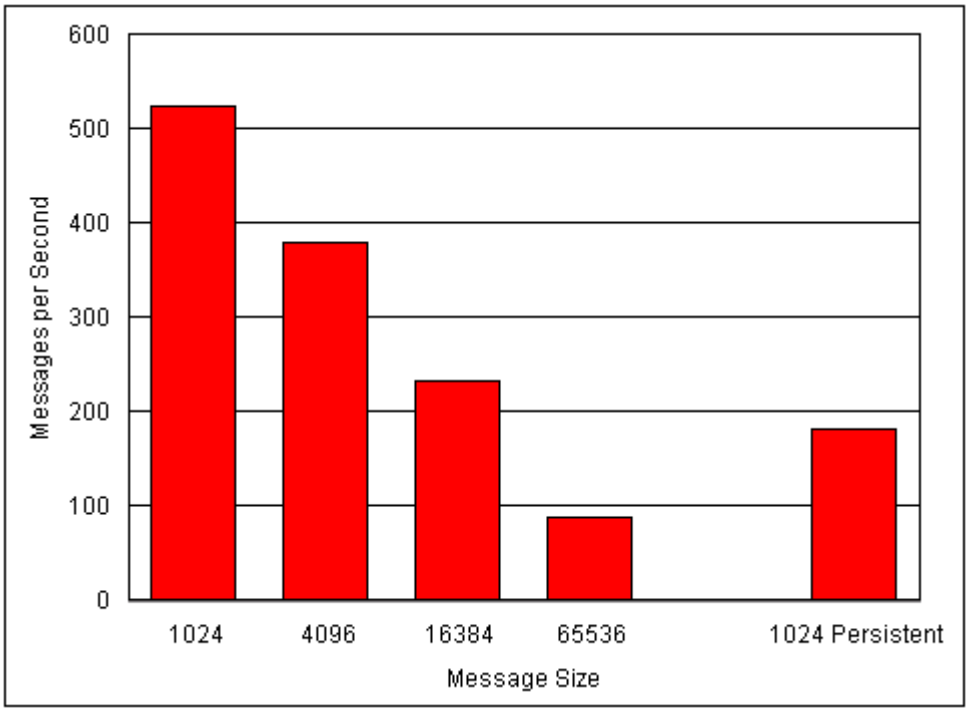

Figure 2: Simple Compute Node Throughput Results

With a 1K non persistent message it was possible to process approximately 525 msgs/second. The message throughput rate declined with size, reflecting the increased volume of data and additional processing required to deal with the messages.

With 1K persistent messages it was possible to process approximately 183 msgs/second. This reduced message rate, when compared with 1K non persistent messages is as a result of the additional logging within the MQSeries manager.

The detailed measurement data for the Simple Compute Node throughput measurements is available in **Section 7.2 - Compute Node Throughput Results**.

2.2.2 Complex Compute Node

Figure 3 below shows the results that were obtained as a result of running a complex message flow with varying message sizes and persistence. See Appendix C for a description of this complex flow. 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.

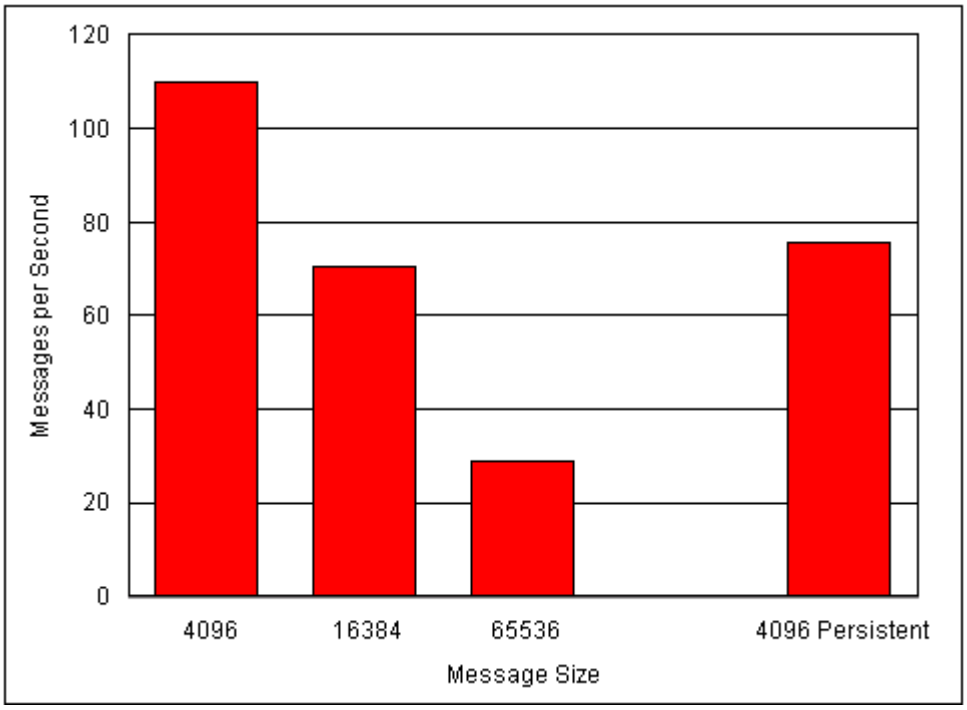

Figure 3: Complex Compute Node Throughput Results

With a 4K non persistent message it was possible to process approximately 110 msgs/second. The message throughput rate declined with size, reflecting the increased volume of data and additional processing required to deal with the messages. 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 4K persistent messages it was possible to process approximately 76 msgs/second. This reduced message rate, when compared with 4K non persistent messages is as a result of the additional logging within the MQSeries manager.

The detailed measurement data for the Compute Node throughput measurements is available in **Section 7.2 - Compute Node Throughput Results**.

2.2.3 Multiple Complex Compute Nodes

Figure 4 below shows the results that were obtained as a result of running five of the above complex nodes daisy chained together for varying message sizes and persistence. See Appendix C for a description of this complex flow. 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.

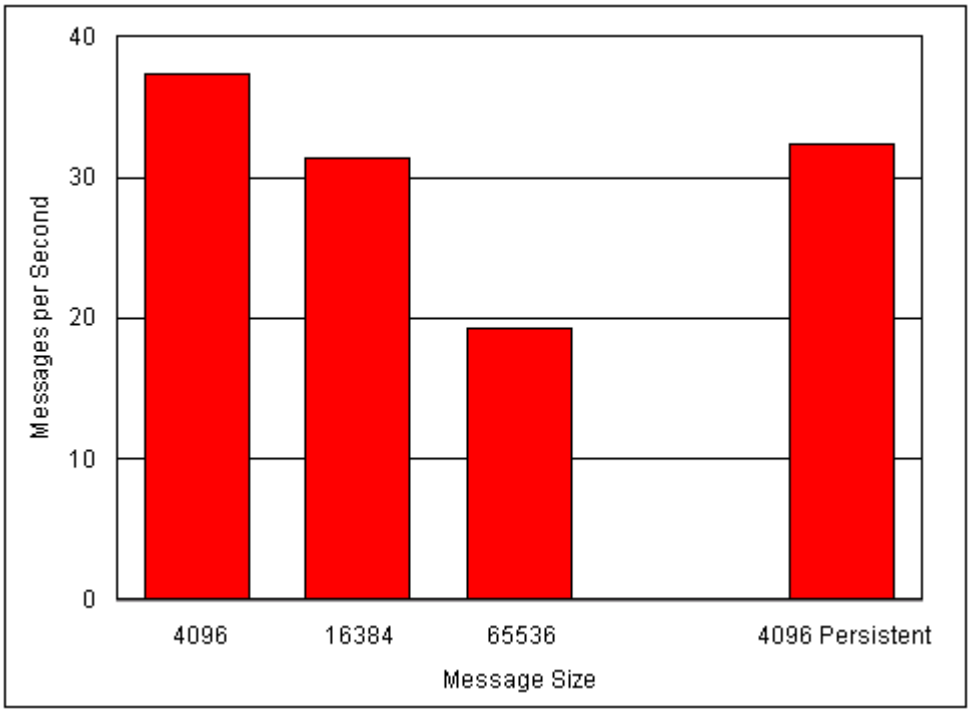

Figure 4: Multiple Complex Compute Node Throughput Results

With a 4K non persistent message it was possible to process approximately 37 msgs/second. The message throughput rate declined with size, reflecting the increased volume of data and additional processing required to deal with the messages.

With 4K persistent messages it was possible to process approximately 33 msgs/second.

The detailed measurement data for the Compute Node throughput measurements is available in **Section 7.2 - Compute Node Throughput Results**.

2.2.4 Very Complex Compute Node

Figure 5 below shows the results that were obtained as a result of running a very complex message flow with varying message sizes and persistence. Appendix C has a description of a very complex flow. Briefly, a very complex flow is defined as 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.

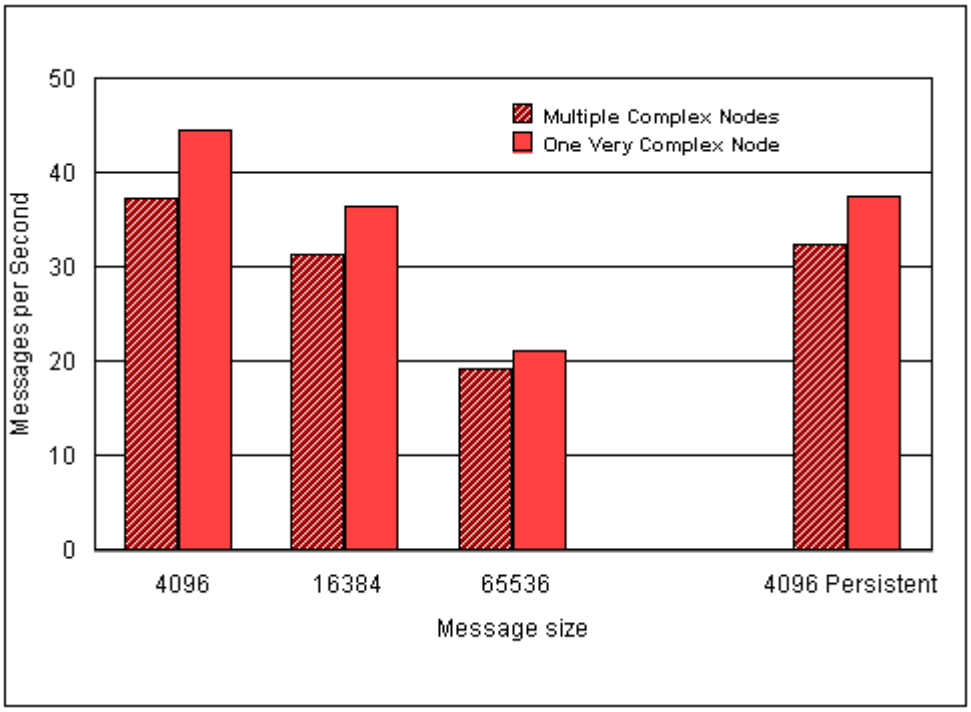

Figure 5: Very Complex Compute Node VS Multiple Complex Compute Node Throughput Results

With a 4K non persistent message it was possible to process approximately 45 msgs/second. The message throughput rate declined with size, reflecting the increased volume of data and additional processing required to deal with the messages.

With 4K persistent messages it was possible to process approximately 38 msgs/second.

For comparison purposes Figure 5 also shows the message throughput rates that were achieved for the multiple complex compute node case detailed in **Section 2.2.3 - Multiple Complex Compute Nodes**.

For 4K non persistent messages there was a 1.16 times improvement in message throughput as a result of 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 detailed measurement data for the Very Complex Compute Node throughput measurements is available in **Section 7.2 - Compute Node Throughput Results**.

2.3 Database Node Throughput

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 on the database node definition.

A message flow consisting of an MQInput node, a database node and an MQOutput node was defined.

The message flow 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 coordinatedTransaction value for the message flow was set to yes. The effect of doing this is to specify that the message flow should 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. **Figure 6** below shows the results that were obtained for varying message size and persistence.

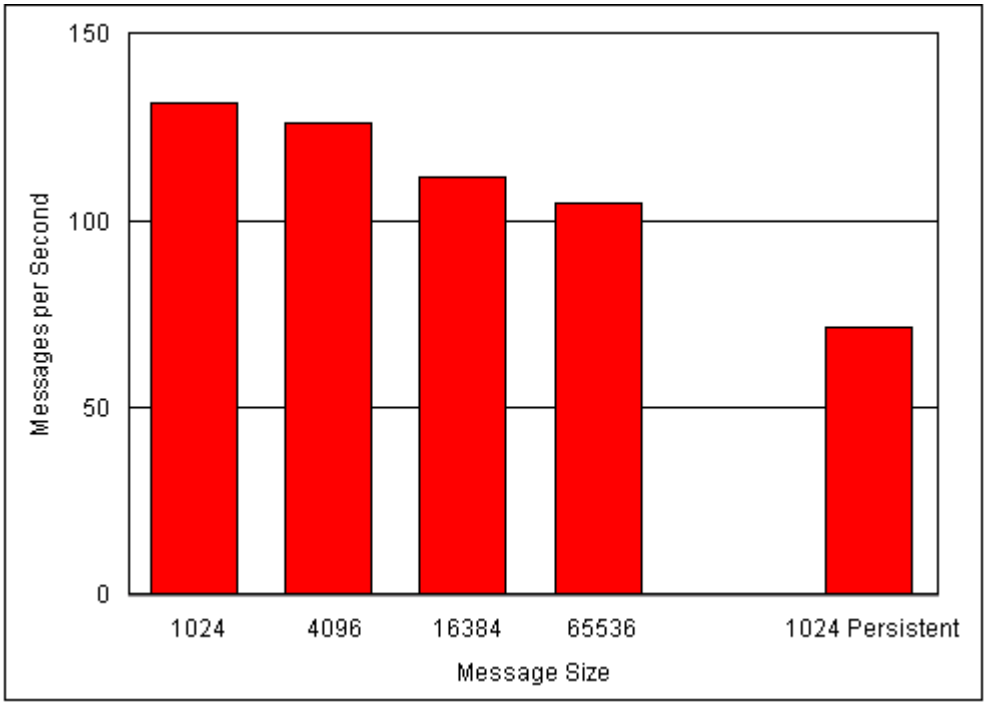

Figure 6: Database Insert/Delete Throughput Results

With 1K non persistent messages it was possible to achieve a message throughput rate of 131 msgs/second. This is 131 database insert and deletes per second. The rate of insert/delete activity reduced with message size.

With 1K persistent messages it was possible to achieve a message throughput rate of 71 msgs/second. This is 71 insert and deletes per second. This lower rate is due to the increased volume of I/O processing to both the MQSeries queue manager log and the DB2 log. It should be noted that persistent message throughput is highly dependent on the performance of the I/O subsystem and on the placement of the DB2 and MQSeries logs.

The detailed measurement data for the Database node measurements is available in **Section 7.3 - Database Node Throughput Results**.

2.4 Filter Node Throughput

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 node was defined. The Filter node processing involved selecting a message on the basis of the contents of a tag value.

The input message consisted of a message with an MQRFH2 header, with two tags specified following the header. The transaction mode on the MQInput and MQOutput nodes was set to automatic.

Figure 7 below shows the results that were obtained as a result of running the message flow with varying message sizes and persistence. There was a single instance and single execution group running the message flow.

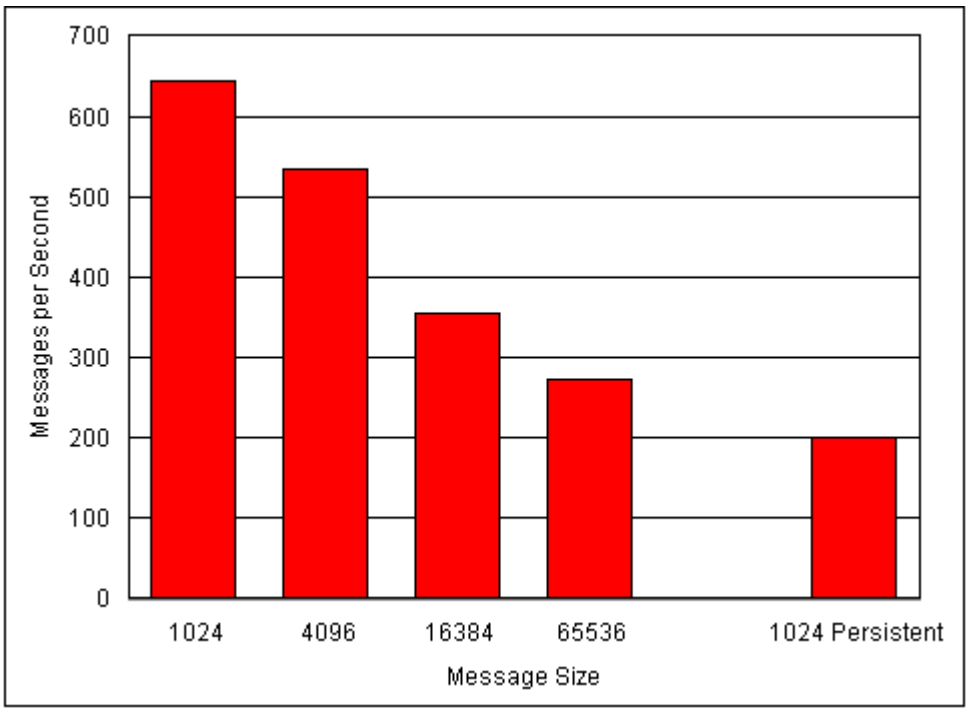

Figure 7: Filter Node Throughput Results

With 1K non-presistent messages it was possible to run 644 msgs/second. The difference between this rate and the 1127 achieved with an MQInput/MQOutput node pair for 1K non-persistent messages represents the overhead of using a Filter node. The cost of the Filter node will vary with the complexity of the filter expression and the number of fields in the input message.

With 1K persistent messages the throughput was 201 msgs/second. The reduction in throughput is as a result of using persistent messages that involves additional logging within the MQSeries manager as well as the fact that the message is processed under MQSeries transaction control.

The detailed measurement data for the Filter Node Throughput measurements is available in **Section 7.4 - Filter Node Throughput Results.**

2.5 RouteToLabel Node Throughput

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 may 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** (located at URL **http://www.ibm.com/software/ts/mqseries/txppacs/ip04.html**.)

The cost of this node is dependent on the size of the destination list. For example, we may have 10 or 100 potential target Label nodes. If the destination list has just one entry, the cost will always be the same. A destination list with 100 entries will cost more to process but a point to note is that this extra cost is not dependent on whether the Route to first or Route to last option is chosen.

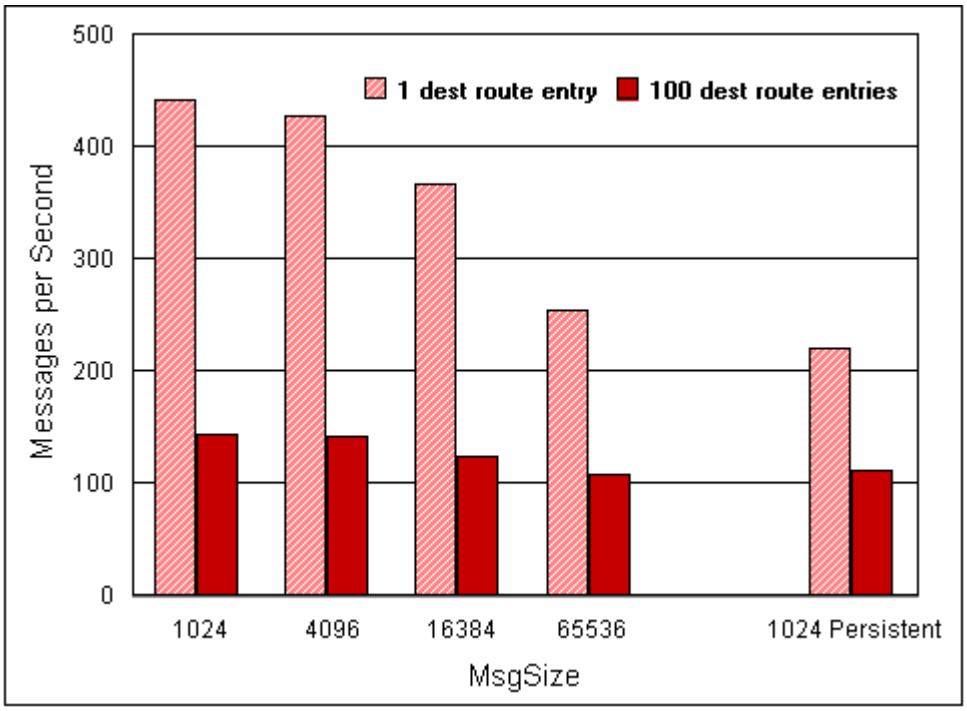

Figure 8: RouteToLabel Node Throughput Results

With 1K non-presistent messages it was possible to run 442 msgs/second when there were only one destination in the list and only 144 msgs/sec when there were 100 entries in the destination list.

With 1K persistent messages it was possible to run 221 msgs/second when there were only one destination in the list and only 112 msgs/sec when there were 100 entries in the destination list. The reduction in throughput is due to additional logging within the MQSeries manager as well as the fact that the message is processed under MQSeries transaction control.

The detailed measurement data for the RouteToLabel Node Throughput measurements is available in **Section 7.5 - RouteToLabel Node Throughput Results.**

2.6 Publication Node Throughput

A publication node may be used within a message flow to represent a point from which messages are "published" that 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 and MQOutput nodes was set to automatic. The measurement used topic routing with MQRFH2 format messages.

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.

With this measurement there was only one subscriber.

Figure 9 below shows the results that were obtained as a result of running the message flow with varying message sizes and persistence. 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.

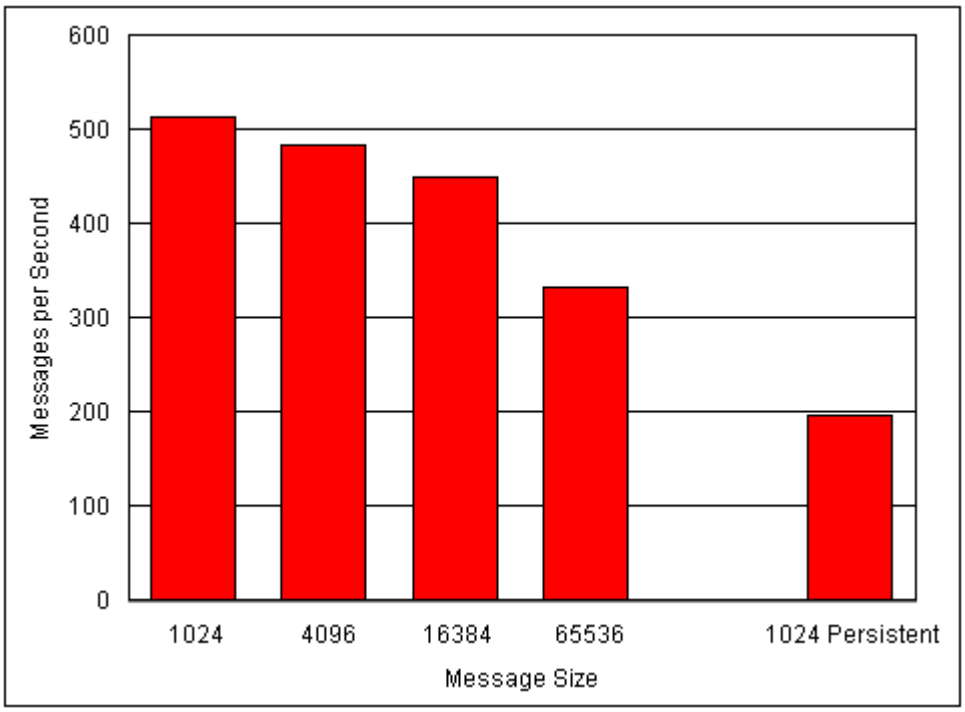

Figure 9: Publication Node Throughput Results

With 1K non-presistent messages it was possible to publish 514 msgs/second.

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

With 1K persistent messages the throughput was 197 msgs/second. The reduction in throughput is as a result of using persistent messages which involves additional logging within the MQSeries manager as well as the fact that the message is processed under MQSeries transaction control.

The detailed measurement data for the Publication Node Throughput measurements is available in **Section 7.6 - Publication Node Throughput Results**.

2.7 What Is The Cost Of Converting Messages To Different Formats ?

WMQI V2.1 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 WMQI V2.1 to convert messages between MRM XML, Generic XML and CWF formats, where MRM XML refers to the predefined XML used within the MRM, Generic XML refers to self-defining XML and CWF denotes a legacy data structure such as a C structure or COBOL copybook.

The same message type was used for each of the conversions. This was a 4096 byte non persistent message containing 31 input fields, including 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 and MRM XML and the message throughput achieved was measured. There was a single execution group running the message flow and no additional instances specified. The results are presented in **Table 1** below.

Conversion	TO	Generic XML	MRM CWF	MRM XML	MRM TAG
FROM					
Generic XML		194.2	121.5	84.6	72.4
MRM CWF		140.8	126.1	92.4	74.0
MRM XML		100.3	90.7	74.9	60.5
MRM TAG		37.8	36.3	33.3	30.4

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, deconstructed by WMQI and then reconstructed into the required output format. However the cost of converting between two formats occurs on a once per message flow basis and not in each node.

The detailed measurement data for cost of message conversion is available in **Section 7.7 - Converting Messages Between Formats Throughput Results.**

2.8 Parallel Processing options

If the message processing rate which can be achieved with a single copy of a message flow is not sufficient for the requirements it is likely that you will need to run multiple copies of the message flow concurrently. Within WMQI V2.1 there are several ways of doing this, they are:

1. Use additional instances of a message flow within an execution group. Each additional instance is a thread within the execution group process

2. Run multiple copies of a message flow within an execution group. Each additional message flow is a thread within the execution group process

3. Run multiple execution groups each processing one or more copies of a message flow. This option uses the most memory as each execution group is a process.

This section shows the effect of running each of these options for the very complex compute message flow

Note that all of the scaling measurements in this section were taken on a 2-way processor, hence the lack of throughput gains with more than 2 instances, flows or execution groups.

2.8.1 What Is The Effect Of Using additional Instances ?

Figure 10 below shows the results that were obtained as a result of running one, two and four instances of a message flow containing the very complex compute node within a single execution group. The very complex compute node is described in Appendix C Complex Compute Node. 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.

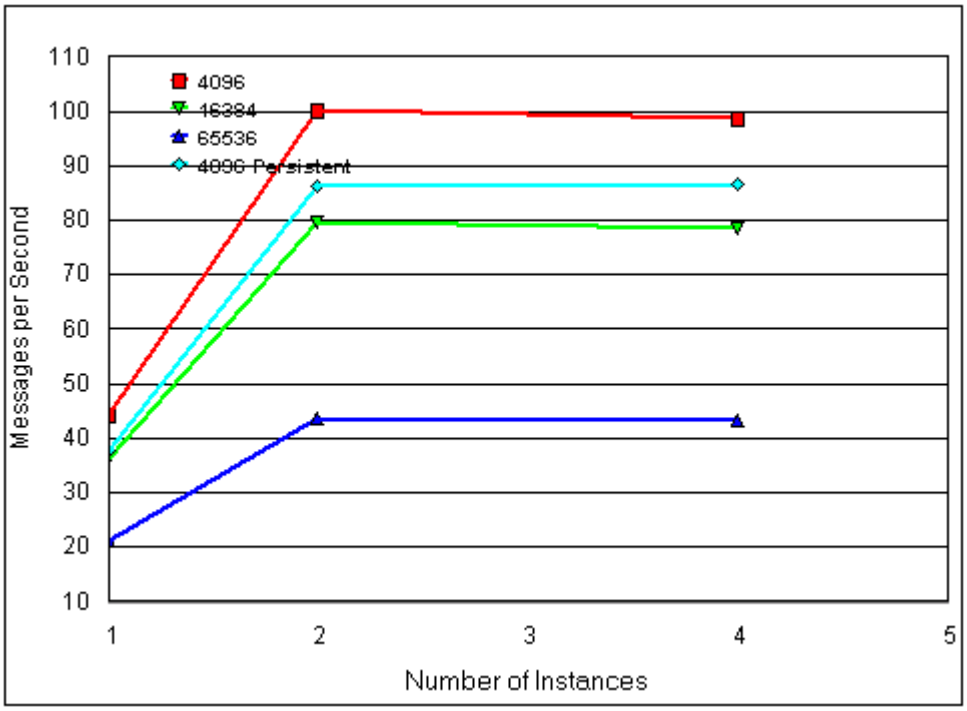

Figure 10: Additional Instances Throughput Results

Figure 10 shows that greater message throughput can be achieved by using additional instances within a single execution group. With non persistent 4K messages and one additional instance it was possible to achieve more than 2 times the throughput that was achieved for a single execution group with one instance. As the tests were performed on a dual-processor machine, using 3 additional instances in this case did not produce 4 times the throughput of a single instance, since both processors reached full capacity (100% CPU busy). 4k persistent messages also benefited from adding one additional instance.

The detailed measurement data showing the effect of using additional instances is available in **Section 7.8.1 - The Effect of Using Additional Instances**.

2.8.2 What Is The Effect of Using Multiple Copies Of a Message Flow Within an Execution Group ?

Figure 11 below shows the results that were obtained as a result of running one, two and four copies of a message flow containing the very complex compute node within a single execution group. The very complex compute node is described in Appendix C Complex Compute Node. 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.

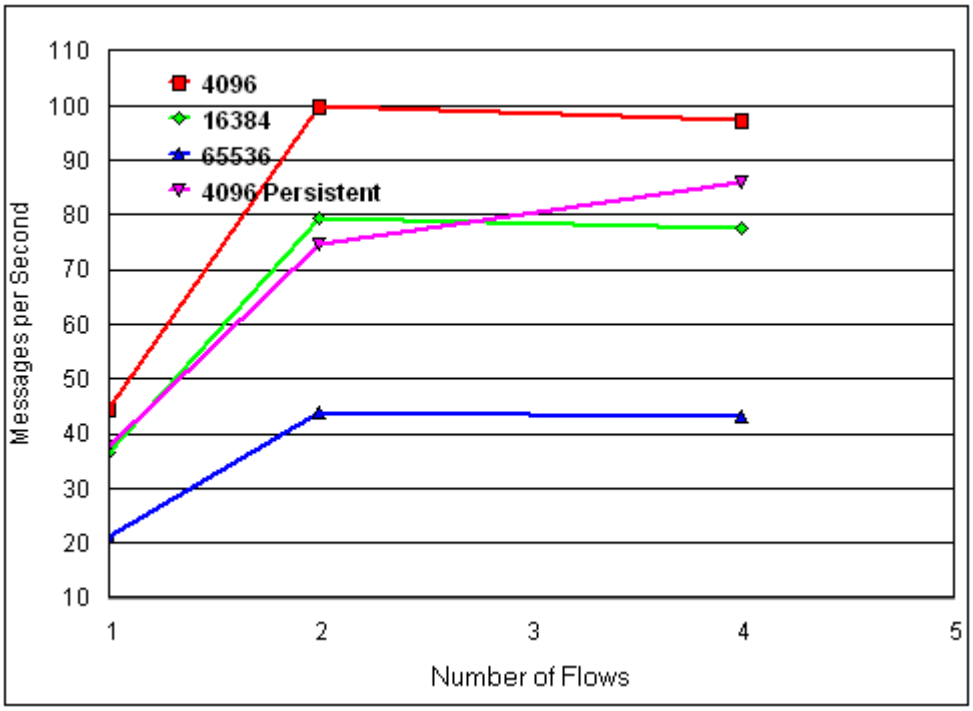

Figure 11: Multiple Copies Throughput Results

Figure 11 shows that greater message throughput can be achieved by using additional copies within a single execution group. With non persistent 4K messages and two copies of the message flow it was possible to achieve 2 times more throughput than was achieved for a single copy of the message flow. As the tests were performed on a dual-processor machine, using 4 copies in this case did not produce 4 times the throughput of a single copy, again due to the fact that both processors had reached full capacity (100% CPU busy) in both the two flow and four flow cases.

There was some further limited gain in throughput for 4k persistent messages however, between the two flow case where CPU was around 90% busy, and the four flow case, where CPU reached 100% busy. For persistent messages this represents the possible upper limit limit of throughput improvement to be acheived by increasing numbers of flows therefore, although it should be stressed that running at 100% CPU busy values is not recommended for either persistent or non-persistent messages, since this is likely to cause undesirably large queueing delays on some messages.

The detailed measurement data showing the effect of using multiple copies of message flow is available in **Section 7.8.2 - The Effect of Using Multiple Copies of a Message Flow.**

2.8.3 What Is The Effect of Increasing The Number of Execution Groups ?

Figure 12 below shows the results that were obtained as a result of running one, two, and four execution groups for a message flow containing the very complex compute node for varying message size and persistence. The message flow is described in Appendix C Complex Compute Node. 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.

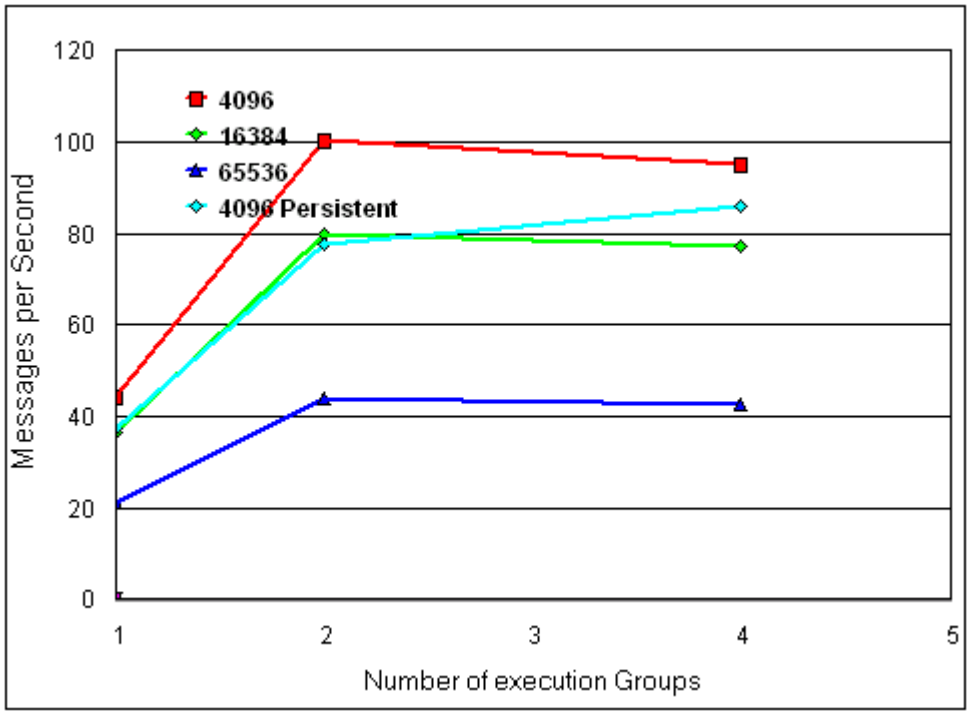

Figure 12: Additional Execution Group Throughput Results

Figure 12 shows that greater message throughput can be achieved by using additional execution groups. With non persistent 4K messages and two execution groups it was possible to achieve over twice the throughput that was achieved for a single execution group. As the tests were performed on a dual-processor machine, using 4 execution groups in this case did not produce 4 times the throughput of a single execution group, since both processors were at full capacity (100% CPU busy) for both the two and four execution groups cases.

It is reasonable expect that if measurements were to be taken, for example, on an 8 processor machine, that the number of execution groups could be increased to seven or eight before the CEC neared maximum capacity, and that a seven or eight times increase in message throughput over the single execution group case would result. In this way scalability in message throughput would be accomplished.

The gain in throughput for 4k persistent messages was also significant. When using two execution groups it was possible to achieve over 2 times the throughput that was achieved with a single execution group, again showing the potential for good scaling of message processing.

The benefit of using multiple execution groups in this case is clearly significant. This is principally because of the nature of the message flow that was used for the measurements. There was a significant amount of ESQL processing in the node. This meant that the level of queue access as a

proportion of all processing was low and so the potential for conflicts on queue access was low and consequently multiple execution groups were able to achieve greater throughput.

The detailed measurement data showing the effect of adding execution groups is available in **Section 7.8.3 - The Effect of Increasing The Number Of Execution Groups.**

2.9 What Is The Effect of Making a Message Flow Transactional?

Making a message flow transactional (as opposed to making an individual node transactional) means that the unit of work is recoverable, but it does result in an additional overhead as work must now take place under transactional control. This involves the locking of data and logging of data images.

The purpose of these measurements was to illustrate the overhead of making a message flow transactional. A simple message flow was created consisting of a single MQInput and MQOutput node. The maximum message throughput rate was measured when the message flow had a transaction mode value of automatic and then with a value of yes.

Figure 13 below shows the results that were obtained as a result of running the message flow with varying message sizes and persistence.

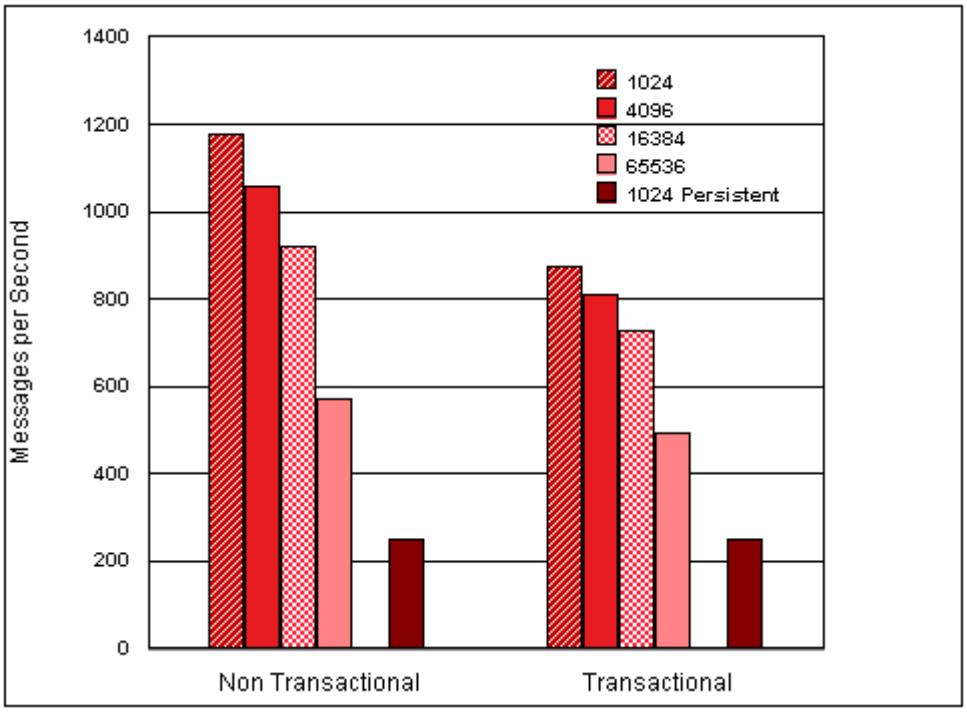

Figure 13: Making a Message Flow Transactional Throughput Results

Making non-persistent messages transactional had a significant effect on message throughput. The reduction in throughput is as a result of the additional CPU and I/O processing that must take place. The overhead of making the message flow transactional was most significant with the smaller message sizes.

For the persistent messages there is little difference in throughput as persistent messages would proceed under transaction control any way with a transaction mode of automatic.

The detailed measurement data for showing the effect of making a node transactional is available in **Section 7.1** and **Section 7.9 - The Effect of Making a Message Flow Transactional**.

2.10 What Is The Effect of Using Coordinated Transaction=yes on a Message Flow?

Specifying a value of yes for the Coordinated Transaction parameter on a message flow means that all updates performed within the message flow will take place as a global unit of work. Any database updates that are in the message flow will be committed atomically with the message processing. On z/OS however, unlike on other platforms, all WMQI transaction coordination is managed by RRS.

If data is to be updated in an MQSeries message and a relational database, and recovery is required, this configuration must be used.

Measurements were taken to illustrate the comparative costs associated with using the Coordinated Transaction parameter on an execution group definition. The message flow consisted of an MQInput, database and MQOutput node. The database node performed an update of a row of in a relational database. The purpose of this measurement was to illustrate the base cost of a defining the transaction as a coordinated transaction..

Figure 14 below shows two sets of results. The first is for the case when coordinatedTransaction was set to no on the message flow and transaction mode was set to automatic on the MQInput and MQOutput nodes. The second case shows the results that were obtained when a value of yes was specified for coordinatedTransaction on the message flow.

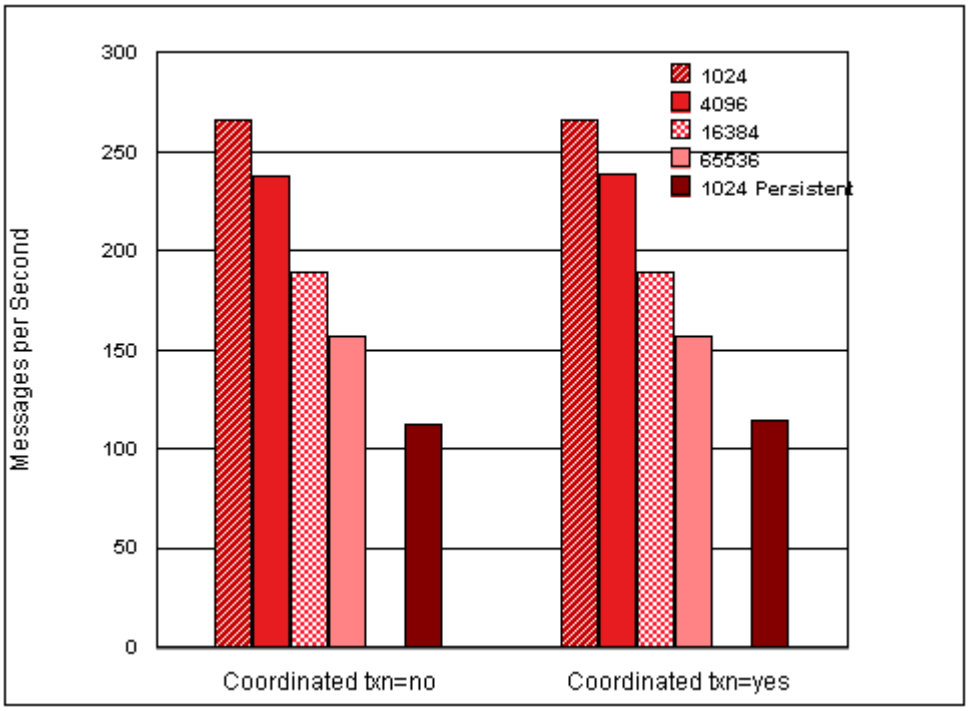

Figure 14: Database Update with coordinatedTransaction=yes.

Figure 14 shows that a message rate of approximately 266 msgs/second was achieved for 1K non-persistent messages and 113 msgs/second for persistent messages when coordinated Transaction was set to “no” on the message flow, transaction mode was set to automatic on the MQInput and MQOutput nodes.

With coordinatedTransaction set to “yes” on the message flow a rate of 266 msgs/second was achieved for non persistent messages and 114 msgs/second for persistent messages.

The results show that message throughput was very similar for both the coordinatedTransaction=no and coordinatedTransaction=yes cases. This is because RRS is included by default as the transaction coordinator when WebSphere MQ and database operations performed within the same message flow. In this particular measurement there was an insert and delete of a message into a database table. It is not possible to run without RRS where an external resource is involved and this is why we see no difference in message rate.

The detailed measurement data showing the effect of using coordinatedTransaction set to the value of yes is available in **Section 7.10 - The Effect of using coordinatedTransaction=yes.**

2.11 What Effect Does an Increasing Number of Subscribers Have on Publish/Subscribe Throughput?

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 currently 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 a series of measurements were taken with 1, 10, 30, 50, 70, 100, and 1000 subscribers. Messages of varying size and persistence were published to a single topic. The results obtained are presented in **Figure 15** 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.

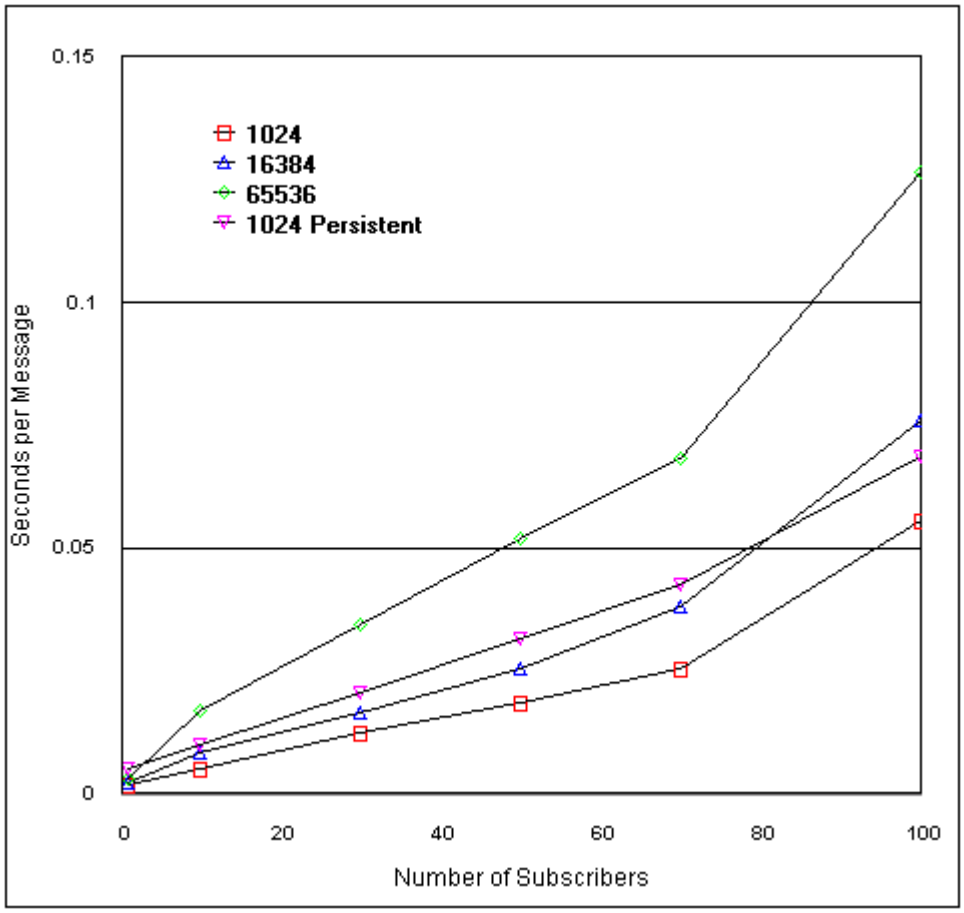

Figure 15: Varying Number of Subscribers

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 MQSeries 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. Before 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 that registered to receive publications the published message was written to a queue for that subscriber. With 30 subscribers for example, a single message was written to each of 30 queues. In the measurement environment there was a background program consuming all but one of the published messages. Taking the example of 10 subscribers, 9 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 WMQI V2.1 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.

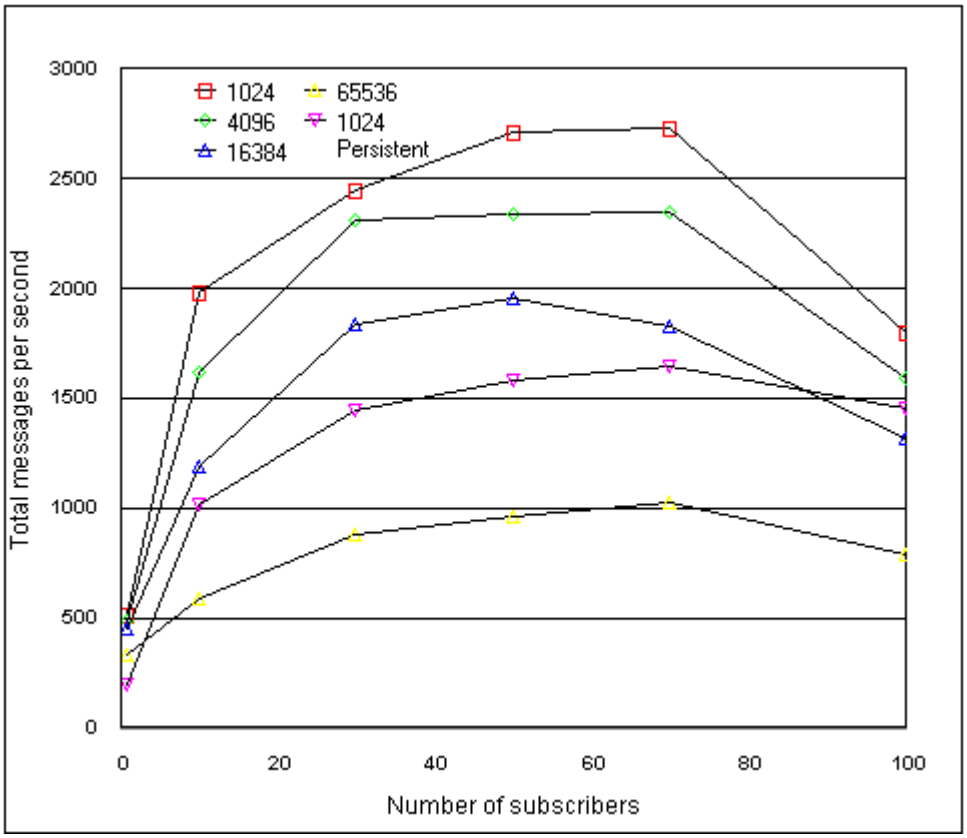

Figure 16: Total Message Rate with Varying Number of Subscribers

The above chart shows, for each message size, that as the number of subscribers was increased, the overall number of messages processed per second reached a peak, and then tailed off as the number of subscribers was further increased.

All measurements in this section with more than 20 subscribers used the mqsichangeproperties command to increase the size of the cache used to hold open queue descriptors. The default value is 30. If the number of open queue descriptors has to increase by 1 beyond the cache size, a queue must be closed before another can be opened and the published message delivered. This can have a significant effect on the rate at which messages can be published. This sequence of closing one queue and opening another will occur each time a message is published unless the cache size is

increased. It is recommended to increase the size of the cache in accordance with the number of registered subscribers, although care should be taken not to exceed 500 cache handles.

The mqsichangeproperties command used to increase the cache for the 100 subscriber measurements was issued using the following JCL:

```
//WMQICHGP  JOB MSGCLASS=A,NOTIFY=&SYSUID,REGION=0M
//*****************************************************************
//*  JCL to issue the mqsichangeproperties command to change the     *
//*  number of queue cache handles.                                  *
//*****************************************************************
//BIPJLMPS EXEC PGM=IKJEFT01,REGION=0M
//STEPLIB  DD DISP=SHR,DSN=PP.ADLE370.OS390R12.SCEERUN
//         DD DISP=SHR,DSN=SYS2.DB2.V610.SDSNEXIT
//         DD DISP=SHR,DSN=SYS2.DB2.V610.SDSNLOAD
//         DD DISP=SHR,DSN=MQM.V520.SCSQAUTH
//         DD DISP=SHR,DSN=MQM.V520.SCSQANLE
//STDENV   DD PATHOPTS=(ORDONLY),
// PATH='/u/wmqi/V52GBRK/ENVFILE'
//STDOUT   DD PATHOPTS=(OWRONLY,OCREAT),
// PATHMODE=(SIRWXU,SIRWXG),
// PATH='/u/wmqi/V52GBRK/output/mqsicpout'
//STDERR   DD PATHOPTS=(OWRONLY,OCREAT),
// PATHMODE=(SIRWXU,SIRWXG),
// PATH='/u/wmqi/V52GBRK/output/mqsicperr'
//SYSPRINT DD SYSOUT=*
//SYSTSPRT DD SYSOUT=*
//SYSTSIN  DD *
BPXBATCH PGM -
/usr/lpp/wmqi/bin/mqsichangeproperties -
V52GBRK -
-e pubsub -
-o ComIbmMQConnectionManager -
-n queueCacheMaxSize -
-v 100
/*
```

where V52GBRK was the name of the WMQI Broker and pubsub was the name of the execution group running the message flow. This command was issued on the machine running the broker.

The command needs to be issued for each execution group containing a publication node with more than 30 subscribers.

The detailed measurement data showing the effect of an increasing number of subscribers is available in **Section 7.5** and **Section 7.11 - The Effect of Increasing the Number of Subscribers.**

3.0 CAPACITY PLANNING

This section gives general guidelines on capacity planning for WMQI V2.1. For more detailed assistance in estimating message rates for particular message flows and the processing power required to support them see Supportpac IP03, MQSeries Integrator V2 - Capacity Planning Tool, available from URL **http://www.ibm.com/software/ts/mqseries/txppacs/ip03.html**. The remainder of this section provides general guidelines to assist with the implementation of WMQI V2.1.

When capacity planning for the introduction of a new software it is important to be able to establish two things:

- How much resource (CPU, disk space, memory) is required to support the required message rate.
- If it is possible to run the software at the expected message rate? Has the software been run at that rate before, and will the deployed system be working within known limits ?

The way in which WMQI V2.1 can be used varies enormously. For this reason it is not possible to provide detailed guidance on the resources that are required for all possible configurations of WMQI V2.1. It is possible however to provide a series of guidelines in order to get an initial estimate of the required capacity. Once a prototype implementation has been developed, future resource requirements can be based on measurement and observation of the prototype and its successor implementations, which is the only meaningful exercise for you. The problem is invariably one of getting started. This section helps with that process.

In gauging the capabilities of WMQI V2.1 to process messages at the required rate examine the various throughput measurements detailed in Section 2.0 - BROKER THROUGHPUT MEASUREMENTS. Remember that the message rates are typically for one instance of a message flow running in one execution group and so these numbers do not represent the highest total message rate which is achievable. They show what one copy of the message flow is capable of.

Projecting measurement results to other machine types is difficult because performance depends on many factors. The only method currently available to gauge relative machine performance is to use published performance figures such as the Large Scale Processor Ratio (LSPR.) LSPR figures can be found at URL **http://www.ibm.com/servers/eserver/zseries/lspr/zSeries.html**. For a WMQI or MQSeries type workload, it is recommended that the ‘Mixed’ LSPR figures are used.

3.1 Throughput

Key factors affecting the throughput rate that is achievable are:

- Use of non-persistent vs persistent messages.
- The types of WMQI V2.1 nodes being used.
- The amount of processing in the nodes, simple vs complex compute nodes for example.
- The number of WMQI V2.1 nodes being used.
- The number of times messages are written to an MQSeries queue from a node.
- The complexity of processing in the nodes, for example complex ESQL statements vs simple copying.

Of the above, those having most effect on message throughput are:

- The use of persistent messages. If persistent messages are used the maximum message rate will be governed by I/O subsystem performance rather than by processor speed.

- The amount of user supplied (ESQL) processing in compute nodes.

- The average number of subscribers to match a topic for a Publication node.

Processing of non persistent messages is CPU intensive. It is therefore important to ensure that there is sufficient CPU available to process the message in order to maximize throughput. Unless otherwise stated, the measurements contained in this report are based on a two-way LPAR of a 9672-XZ7 processor (approximately equivalent to a 9672-X27; LSPR = 1.28.) Using faster or slower processors will have a corresponding effect.

Processing of persistent messages is I/O intensive. It is therefore important to ensure that the adjacent MQSeries queue manager logs are located on separate disks, as well ensuring that the other standard MQSeries tuning tasks have been performed. When running persistent messages CPU utilization will be lower than for non-persistent messages, and so in order to fully utilise available processors it may be necessary to run more copies of the message flow.

3.2 Scaling Message Throughput

Running multiple copies of a message flow provides the opportunity to increase message throughput. This is normally done because the message rate that is achievable with a single execution group is not sufficient for the planned message rates. In general the ability to scale message throughput depends on a number of factors:

1. The availability of sufficient resources (CPU, disks, memory) to cope with the increased resource demands as a result of simply processing more messages.

2. The ability to schedule multiple pieces of work in parallel.

3. A minimum of contention between the parallel pieces of work.

If we look at each of these issues with respect to WMQI V2.1: resolving item 1 above depends on having sufficient hardware of sufficient speed (CPU and disk) available. This is principally a planning issue; For item 2, WMQI V2.1 provides the ability to schedule multiple pieces of work in parallel by the use of multiple execution groups for example; The contention between pieces of work (item 3) depends on the message type (persistent vs non persistent messages), the amount of access to MQSeries queues (high level of access vs low level) and the nature of additional database processing where it is used (Insert/Update/Delete vs read only).

The level of contention in any implementation is largely determined by the nature of the application being implemented and minimizing the effects of contention is an essential part of the application architecture, design and implementation. This is a large subject and is not covered in any level of detail in this report.

3.3 Memory

In estimating memory requirements for WMQI V2.1 there are a number of components that need to be considered. These are:

- The Control Center. There are likely to be multiple Control centers in use.

- The Configuration Manager. There is one Configuration Manager per WMQI V2.1 implementation.

- The Broker. There may be multiple brokers and within these multiple execution groups and so multiple operating system processes.

- MQSeries Queue Manager. There will be one queue manager per broker.

- Relational Database. DB2 systems are required to hold information on behalf of the Configuration Manager and the broker. Additional relational databases may be in use which hold business data.

For the Control Center an at initial recommendation is to allow 100MB memory per Control Center. This would be for development use.

The Configuration Manager and its associated DB2 database and queue manager should have a minimum of 512MB of memory available in a development environment, but the recommendation is to have more.

The amount of memory (real storage) required by a broker will depend on the way in which it is configured. A guideline is to allow 180MB for WMQI V2.1 and its dependent software (broker related components only, no User Name Server), with an additional 130 MB per running execution group. This recommendation is based on an MQSeries queue manager and channel initiator configuration consisting of 20 active channels processing 1 KB messages. If the number of MQSeries resources (channels, queues etc.) to be configured in a system is different or you have a large number of nodes in a flow or very large messages being processed you must make an allowance and amend the amount of memory required accordingly.

3.4 Recommended Minimum Configurations

This section contains recommendations on the type of hardware on which an WMQI V2.1 configuration should be based when running in production. These are only minimum recommendations and are not a substitute for a formal planning and sizing exercise in which requirements are accurately determined.

For production use it is recommended that the components of WMQI V2.1 are allocated over multiple machines with the following purposes:

- One or more Windows NT machines to support Control Center usage.
- One Windows NT machine to support the Configuration Manager. This may also include one Control Center.
- One or more machines to support brokers.

It is important that processing on the broker machine proceeds without competition for resources from other processes in order to ensure the smooth flow of messages through the enterprise.

A recommended Windows NT machine specification for the Control Center is a fast uni-processor with 512MB memory.

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

The specification of the broker machine is more difficult to determine since it requires knowledge of the expected message rate, the types of node that are to be used and the level of transaction control that is used. A recommended minimum specification would be a 2 way processor with 2048MB real storage. The processor speed and number of CPUs may need to be upgraded if the required message rates are high. 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 to be used the use of a fast disk subsystem is recommended for the devices on which the MQSeries queue manager logs are located, for example IBM ESS model 2105-F20. Adjacent MQSeries logs should be placed on separate disk volumes.

If business data is accessed from a relational database the database logs and data should each be located on dedicated disks. Consider using a fast device for the database manager log.

4.0 PERFORMANCE RECOMMENDATIONS

This section contains a number of recommendations to assist in the planning and implementation of an efficient WMQI V2.1 configuration.

4.1 Understand Recovery Requirements

In designing messageflows within WMQI V2.1 it is important that the subject of data recovery is approached from the top down rather than bottom up. If you do not consider the recovery needs as a whole it is possible that more logging than is actually required will be undertaken. This will lead to a drop in the throughput rate that is achievable with a message flow as the flow becomes I/O bound.

In designing the message flow it is important to establish whether the whole message flow is to be made a recoverable or whether only certain parts of it are. It is also important to establish whether external resource managers such as a database are required. Establish whether data updated in an external resource manager is to be committed within a global unit of work or not.

If a message flow is to involve data held in an external resource manager, i.e. not a queue manager, then consider using the coordinatedTransaction parameter in order to make all changes to external data within the scope of a single unit of work.

Think carefully about when deciding to use MQSeries transactional control on messageflows. Maybe this is something that is only required for persistent messages.

4.2 Optimize Queue Manager

The performance of the underlying queue manager for a broker plays a key role in the performance that can be obtained from using WMQI V2.1.

To improve overall performance with the queue manager consider minimizing message sizes and only use persistent messages where required.

With non persistent messages there is little that can be done to optimize queue manager performance other than ensuring that there is sufficient memory and CPU available.

With persistent messages the limiting factor is the speed at which the MQSeries queue manager log operates. To minimize the amount of logging taking place and improve the efficiency where possible consider the following points:

- The MQSeries queue manager adjacent logs should be placed on separate disks.
- Use the fastest disks available for the MQSeries log.
- Ensure that the amount of disk space made available to the MQSeries logs is sufficient for the volumes and rates at which message data will be logged, so that offloading/archiving is kept to a minimum.
- Run with parallel applications rather than a single application. There is some benefit to be obtained from coat tailing on log I/O that occurs when there is more than one application running with the queue manager. This can be achieved by using multiple copies of a message flow, additional instances or multiple execution groups.

4.3 Configuration Considerations

Consider the following points when building an WMQI V2.1 configuration:

- It is not recommended to use the database instances for the Configuration Manager or broker to hold business data.
- It is recommended to ensure that the database instance for the Configuration Manager is local to the machine on which the Configuration Manager is installed.
- It is recommended to ensure that the database instance for the broker is local to the machine on which the broker runs.
- It is recommended to use a local database for business data. Where such a database is remote from the broker machine, ensure that there is a fast, preferably dedicated, communications link between the broker machine and the database manager.
- Carefully examine default settings for nodes and messageflows, especially those related to recovery, to ensure that the values are those required. The transaction mode parameter for an MQInput node will default to yes, meaning that the message flow will proceed under transaction control. This may not be what was required.
- When creating and deploying large messageflows increase the heap allocation of the Configuration Manager database. In DB2 this is the APP_CTL_HEAP_SIZE parameter. You should increase the value empirically.
- It is recommended that all HFSs relating to WMQI are mounted locally in a sysplex environment.

4.4 Maximizing Throughput

In order to improve the message throughput for a message flow, consider the following points:

- Achieve as much parallelism as possible. This can be achieved in WMQI V2.1 by running multiple copies of a message flow. Doing this will result in the creation and use of another thread or process which provides the potential to increase CPU utilization by using another processor. These approaches are only effective on a machine that has multiple processors. With a single processor machine it will not be possible to improve throughput in this way. However, it may still be necessary to configure multiple execution groups for other reasons.
- Avoid small message flows which use MQSeries queues to communicate. Writing a message to a queue is a relatively expensive operation when compared with moving a message between nodes in the same message flow. It would be better to form one larger single message flow, do not move to the other extreme though and put all processing into one single flow.
- Compute nodes are expensive in processing costs because they build a representation of the input message. Aim to minimize the number of compute nodes therefore. Dependent on the circumstances, you may consider using a filter node instead of a compute node if message selection is required.
- When using publication nodes ensure that the open queue cache size is set appropriately. See section 2.12 *What Effect Does an Increasing Number of Subscribers Have on Publish/Subscriber Throughput?* for more details.
- When designing messages, make them as simple as possible. Large and more complex messages require more parsing. This consumes more CPU.
- Regularly monitor the performance of any database containing business data.
- Ensure that there are sufficient resources available to the database manager (CPU, memory, disk) so that it does not becoming a limiting factor in message throughput.
- Use fast disks for the queue manager log. See Section 4.2 - Optimize Queue Manager above.

- Use fast disks for the database log where insert/delete/update activity is taking place in message flows.

- Turn on Dynamic Statement Cacheing for your DB2 system (add CACHEDYN = YES to the DB2 Zparm member) if your flows contain DB2 database insert/delete/update activity, as this has been found to considerably improve throughput.

- Consider the recommendations described in Supportpac IP04, Designing Message Flows for Performance. This Supportpac makes a number of recommendations for message flow design and the use of ESQL.

4.5 Configuring Shared Libraries

Each WMQI execution group results in a new z/OS USS address space and for each, separate copies of the WMQI libraries are loaded. In a configuration which employs multiple execution groups, the multiple library copies can soon put a strain on real storage, often resulting in excessive swapping which in turn adversely affects performance.

One way of reducing the load on real storage is to make use of USS shared library support. When the extended attribute **st_sharelib** is set for an executable, upon first use it is loaded into the USS shared library region from where it can be shared by multiple USS address spaces. Full details of the use of st_sharelib can be found in manuals *UNIX System Services Programming: Assembler Callable Services Reference* and *UNIX System Services Planning*.

Below is a checklist of the tasks required to enable shared DLLs for WMQI:

- Ensure your userid has READ authority to RACF facility class BPX.FILEATTR.SHARELIB. This is needed in order to use the **extattr +l** (Note, lowercase L) command which sets the st_sharelib attribute .
- Run extattr against the *.a and *.lil WMQI executables, located in directories /usr/lpp/wmqi/lib and /usr/lpp/wmqi/lil respectively, thus

```
cd /usr/lpp/wmqi
extattr +l lib/*.a lil/*.lil
```

- Ensure that all shared executables have read permission set for 'other', i.e.

```
cd /usr/lpp/wmqi
chmod -R o+r lib lil
```

- Check the current setting of USS parameter SHRLIBRGNSIZE - it needs to be set to around 300MB to accommodate the WMQI shared libraries. z/OS command /D OMVS,L displays the current setting. SHRLIBRGNSIZE can be increased with the command /SETOMVS SHRLIBRGNSIZE=300000000.

- Start the WMQI brokers. The /D OMVS,L command can be used to check that the shared library region is being used - the current usage column of the display output should contain a value other than 0, e.g.

```
D OMVS,L
BPXO051I 12.13.01 DISPLAY OMVS 682
OMVS     000F ACTIVE          OMVS=(P3,12,25)
SYSTEM WIDE LIMITS:         LIMMSG=NONE
                   CURRENT  HIGHWATER     SYSTEM
                     USAGE      USAGE      LIMIT
MAXPROCSYS              41         46        512
MAXUIDS                  4          5        100
MAXPTYS                  1          4        256
MAXMMAPAREA              3          3      40960
MAXSHAREPAGES        12032      19203   32768000
IPCMSGNIDS              10         10        500
IPCSEMNIDS               9         10        500
IPCSHMNIDS               0          0        500
IPCSHMSPAGES             0          0     262144
IPCMSGQBYTES           ---         12 2147483647
IPCMSGQMNUM            ---          1      10000
IPCSHMMPAGES           ---          0      25600
SHRLIBRGNSIZE   219152384  219152384  300000000 *
SHRLIBMAXPAGES        2965       2965       4096
```

5.0 GLOSSARY

- **CPU %**

  Average % busy time of the entire machine. 100% means all processors in the machine are busy.

- **CPU mS per message**

  CPU milliseconds consumed per message. These figures are relative to a **single** processor of the 9672-XZ7 CEC on which the evaluation was performed and are derived from the average CPU % busy figure, i.e. they are calculated as.....

  (((CPU % / 100) x (No-of-processors used in LPAR (2)) / (Messages per second)) x 1000

- **Msg Size**

  Size of the user portion of a message. Does not include the MQSeries header size.

- **Persistent or Persist**

  Indicates whether the message type was persistent (yes) or non persistent (no).

- **Process**

  On z/OS a process typically runs in a separate address, but this depends on how the process was created.

- **MQRFH**

  Rules and Formatting Header(RFH) used for Publish/Subscribe applications. Publishing and Subscribing applications send their messages to the broker in MQRFH format

- **MQRFH2**

  MQSeries Rules and Formatting Header(MQRFH2) version 2. The principal use is to contain message format information, and, for Publish/Subscribe additional control information for publications and subscriptions.

- **Thread**

  On z/OS a thread runs under a separate TCB.

- **Msgs/sec**

  Messages throughput in messages per second processed by the WMQI broker. An individual message operation starts when the request message is put to the broker by the driving application and ends when the reply message (or messages) is received by the driving application.

- **USS**

  Unix System Services. The unix environment provided by z/OS.

6.0 APPENDIX A - MEASUREMENT HARDWARE AND SOFTWARE

All throughput measurements where taken on a single server machine driven by MQSeries programs running on the same processor as the WMQI broker and the MQSeries queue manager.

Broker Machine

The broker machine hardware consisted of

- A two-engined LPAR of a 9672-XZ7 processor. This is approximately equivalent to a 9672-X27 which has a mixed workload LSPR figure of = 1.28 compared to the LSPR reference zSeries 900 2064-1C1machine .
- An ESS 2105-E20 DASD subsystem with Feature 2121
- 2048 MB real storage.

The broker machine software consisted of:

- z/OS V1.2.0.
- MQSeries for z/OS V5.2 .
- WebSphere MQSeries Integrator for z/OS V2.1.
- DB2 for z/OS V6.1.

7.0 APPENDIX B - MEASUREMENT DATA

This section contains the detailed results of the WMQI V2.1 performance measurements described in **Section 2.0 - Measurement Results**. For an explanation of column headings see **Section 5.0 - Glossary**. The CPU utilization reported for the host machine, under the headings “Cpu %” and “Cpu mS per msg” in each table, is the total CPU utilization on the host machine. This includes **all** processes on the machine. The figure therefore reflects the cost of the MQSeries queue manager processes, the driving application, DB2 where appropriate etc. in addition to the CPU used by the WMQI V2.1 broker.

The default messages used in the performance measurements were not set to any particular type, i.e. MQRFH, MQRFH2 or XML. Where a particular type was used it is indicated.

7.1 MQInput/MQOutput Throughput Results

Persist	MsgSize	Msgs/sec	CPU%	CPU mS per msg
no	1024	1175.64	79.43	1.351
no	4096	1057.36	83.1	1.571
no	16384	918.6	84.66	1.843
no	65536	570.16	86.19	3.023
yes	1024	248.38	32.46	2.613

7.2 Compute Node Throughput Results

7.2.1 Simple Compute Node

This measurement used a message type of XML

Persist	MsgSize	Msgs/sec	CPU%	CPU mS per msg
no	1024	525.22	63.54	2.419
no	4096	379.17	63.64	3.356
no	16384	233.7	60.19	5.151
no	65536	88.76	57.37	12.926
yes	1024	182.56	36.63	4.012

7.2.2 Complex Compute Node

This measurement used a message type of XML.

Persist	MsgSize	Msgs/sec	CPU%	CPU mS per msg
no	4096	109.77	55.5	10.112
no	16384	70.54	54.78	15.531
no	65536	29.01	53.83	37.111
yes	4096	75.57	45.49	12.039

7.2.3 Multiple Complex Compute Nodes

This measurement used a message type of XML.

Persist	MsgSize	Msgs/sec	CPU%	CPU mS per msg
no	4096	37.3	53.15	28.498
no	16384	31.41	53.2	33.874
no	65536	19.28	53.15	55.134
yes	4096	32.36	49.3	30.469

7.2.4 Very Complex Compute Node

This measurement used a message type of XML.

Persist	MsgSize	Msgs/sec	CPU%	CPU mS per msg
no	4096	44.55	53.37	23.959
no	16384	36.43	53.41	29.321
no	65536	21.07	53.32	50.612
yes	4096	37.58	48.83	25.987

7.3 Database Node Throughput Results

Database Insert/Delete with transaction coordination, using XML messages

Persist	MsgSize	Msgs/sec	CPU%	CPU mS per msg
no	1024	131.44	38.35	5.835
no	4096	126.14	39.79	6.308
no	16384	111.92	41.97	7.5
no	65536	104.91	45.58	8.689
yes	1024	71.43	31.36	8.78

7.4 Filter Node Throughput Results

This measurement used xml messages.

Persist	MsgSize	Msgs/sec	CPU%	CPU mS per msg
no	1024	644.07	66.46	2.063
no	4096	533.35	66.6	2.497
no	16384	353.93	64.38	3.638
no	65536	271.63	70.72	5.207
yes	1024	201.41	33.97	3.373

7.5 RouteToLabel Node Throughput Results

7.5.1 RouteToLabel 1 destination entry

This measurement used xml messages.

Persist	MsgSize	Msgs/sec	CPU%	CPU mS per msg
no	1024	441.99	64.61	2.923
no	4096	427.61	66.98	3.132
no	16384	366.48	68.83	3.756
no	65536	254.14	68.51	5.391
yes	1024	220.63	35.9	3.254

7.5.2 RouteToLabel 100 destination entries

This measurement used xml messages.

Persist	MsgSize	Msgs/sec	CPU%	CPU mS per msg
no	1024	143.62	55.55	7.735
no	4096	141.25	56.84	8.048
no	16384	124.55	57.62	9.252
no	65536	107.66	59.57	11.066
yes	1024	112.45	49.02	8.718

7.6 Publication Node Throughput Results

This measurement used MQRFH2 messages.

Persist	MsgSize	Msgs/sec	CPU%	CPU mS per msg
no	1024	513.65	65.1	2.534
no	4096	484.26	67.53	2.788
no	16384	450.64	69.11	3.067
no	65536	333.88	76.12	4.559
yes	1024	197.08	38.33	3.889

7.7 Converting Messages Between Formats

This measurement is from generic xml to generic xml format.

Persist	MsgSize	Msgs/sec	CPU%	CPU mS per msg
no	4096	194.2	57.91	5.963
no	16384	100.49	56.03	11.151
no	65536	33.22	54.19	32.624
yes	4096	110.05	44.11	8.016

This measurement is from generic xml to cwf format

Persist	MsgSize	Msgs/sec	CPU%	CPU mS per msg
no	4096	121.5	55.55	9.144
no	16384	80.1	55.46	13.847
no	65536	40.06	54.53	27.224
yes	4096	87.28	47.86	10.967

This measurement is from generic xml to mrm xml format

Persist	MsgSize	Msgs/sec	CPU%	CPU mS per msg
no	4096	84.59	54.73	12.94
no	16384	48.35	53.92	22.304
no	65536	17.57	53.14	60.489
yes	4096	62.69	48.22	15.383

This measurement is from generic xml to tag (tds) format

Persist	MsgSize	Msgs/sec	CPU%	CPU mS per msg
no	4096	72.38	54.18	14.97
no	16384	56.31	54.18	19.243
no	65536	29.36	53.95	36.75
yes	4096	58.24	49.13	16.871

This measurement is from cwf format to generic xml format.

Persist	MsgSize	Msgs/sec	CPU%	CPU mS per msg
no	4096	140.75	56.75	8.063
no	16384	82.31	55.3	13.437
no	65536	31.53	54.07	34.297
yes	4096	87.04	45.32	10.413

This measurement is from cwf format to cwf format.

Persist	MsgSize	Msgs/sec	CPU%	CPU mS per msg
no	4096	126.11	56.12	8.9
no	16384	88.65	55.6	12.543
no	65536	35.85	54.75	30.543
yes	4096	87.77	47.11	10.734

This measurement is from cwf format to mrm xml format.

Persist	MsgSize	Msgs/sec	CPU%	CPU mS per msg
no	4096	92.43	54.98	11.896
no	16384	50.92	54.04	21.225
no	65536	18.37	53.15	57.866
yes	4096	67.41	48.2	14.3

This measurement is from cwf format to tag (tds) format

Persist	MsgSize	Msgs/sec	CPU%	CPU mS per msg
no	4096	74.03	54.41	14.699
no	16384	57.69	54.3	18.824
no	65536	31.16	54.12	34.736
yes	4096	56.22	48.72	17.331

This measurement is from mrm xml format to generic xml format.

Persist	MsgSize	Msgs/sec	CPU%	CPU mS per msg
no	4096	100.26	55.24	11.019
no	16384	66.15	54.63	16.517
no	65536	27.27	53.86	39.501
yes	4096	72.48	47.89	13.214

This measurement is from mrm xml to cwf format

Persist	MsgSize	Msgs/sec	CPU%	CPU mS per msg
no	4096	90.67	54.64	12.052
no	16384	68.94	54.75	15.883
no	65536	34.13	54.23	31.778
yes	4096	70.53	49.03	13.903

This measurement is from mrm xml format to mrm xml format.

Persist	MsgSize	Msgs/sec	CPU%	CPU mS per msg
no	4096	74.93	54.47	14.538
no	16384	44.83	53.82	24.01
no	65536	16.96	53.06	62.57
yes	4096	59.36	48.86	16.462

This measurement is from mrm xml format to tag (tds) format

Persist	MsgSize	Msgs/sec	CPU%	CPU mS per msg
no	4096	60.5	53.82	17.791
no	16384	48.77	53.85	22.083
no	65536	26.89	53.77	39.992
yes	4096	48.15	49.45	20.539

This measurement is from tag (tds) format to generic xml format.

Persist	MsgSize	Msgs/sec	CPU%	CPU mS per msg
no	4096	37.79	53.18	28.145
no	16384	18.04	52.56	58.27
no	65536	5.85	52.26	178.666
yes	4096	33.17	49.86	30.063

This measurement is from tag (tds) to cwf format

Persist	MsgSize	Msgs/sec	CPU%	CPU mS per msg
no	4096	36.33	53.18	29.276
no	16384	18.21	52.64	57.814
no	65536	6.06	52.26	172.475
yes	4096	32.35	50.46	31.196

This measurement is from tag (tds) format to mrm xml format.

Persist	MsgSize	Msgs/sec	CPU%	CPU mS per msg
no	4096	33.27	52.99	31.854
no	16384	15.91	52.54	66.046
no	65536	5.14	52.11	202.762
yes	4096	29.71	50.33	33.88

This measurement is from tag (tds) format to tag (tds) format.

Persist	MsgSize	Msgs/sec	CPU%	CPU mS per msg
no	4096	30.35	52.91	34.866
no	16384	16.41	52.54	64.034
no	65536	5.85	52.28	178.735
yes	4096	27.5	50.55	36.763

7.8 Parallel Processing

This section contains the results of running with additional instances, multiple copies of a message flow and multiple execution groups for a message flow with a complex compute node.

7.8.1 The effect of Using Additional Instances

One Instance of a Very Complex Compute Message Flow Running in One Execution Group

Persist	MsgSize	Msgs/sec	CPU%	CPU mS per msg
no	4096	44.55	53.37	23.959
no	16384	36.43	53.41	29.321
no	65536	21.07	53.32	50.612
yes	4096	37.58	48.83	25.987

Two Instances of a Very Complex Compute Message Flow Running in One Execution Group

Persist	MsgSize	Msgs/sec	CPU%	CPU mS per msg
no	4096	100.1	100	19.98
no	16384	79.58	100	25.131
no	65536	43.71	100	45.756
yes	4096	86.15	92.24	21.413

Four Instances of a Very Complex Compute Message Flow Running in One Execution Group

Persist	MsgSize	Msgs/sec	CPU%	CPU mS per msg
no	4096	98.66	100	20.271
no	16384	78.48	100	25.484
no	65536	43.44	100	46.04
yes	4096	86.48	99.95	23.115

7.8.2 The Effect of Using Multiple Copies of a Message Flow

One Copy of a Very Complex Compute Message Flow Running in One Execution Group

Persist	MsgSize	Msgs/sec	CPU%	CPU mS per msg
no	4096	44.55	53.37	23.959
no	16384	36.43	53.41	29.321
no	65536	21.07	53.32	50.612
yes	4096	37.58	48.83	25.987

Two Copies of a Very Complex Compute Message Flow Running in One Execution Group

Persist	MsgSize	Msgs/sec	CPU%	CPU mS per msg
no	4096	99.91	100	20.018
no	16384	79.58	100	25.131
no	65536	43.7	100	45.766
yes	4096	74.51	91.32	24.759

Four Copies of a Very Complex Compute Message Flow Running in One Execution Group

Persist	MsgSize	Msgs/sec	CPU%	CPU mS per msg
no	4096	97.38	100	20.538
no	16384	77.55	100	25.789
no	65536	43.08	100	46.425
yes	4096	86.11	99.98	23.221

7.8.3 The Effect of Increasing The Number Of Execution Groups

One Execution Group Running a very complex Compute Message Flow

Persist	MsgSize	Msgs/sec	CPU%	CPU mS per msg
no	4096	44.55	53.37	23.959
no	16384	36.43	53.41	29.321
no	65536	21.07	53.32	50.612
yes	4096	37.58	48.83	25.987

Two Execution Groups Running a very complex Compute Message Flow

Persist	MsgSize	Msgs/sec	CPU%	CPU mS per msg
no	4096	100.61	100	19.878
no	16384	79.95	100	25.015
no	65536	43.86	100	45.599
yes	4096	77.64	90.84	23.4

Four Execution Groups Running a Very Complex Compute Message Flow

Persist	MsgSize	Msgs/sec	CPU%	CPU mS per msg
no	4096	95.14	99.99	21.019
no	16384	77.48	100	25.813
no	65536	42.71	100	46.827
yes	4096	85.94	99.97	23.265

7.9 The Effect of Making a Message Flow Transactional

Transactional MQInput/MQOutput Message Flow (Transaction Mode set to yes)

Persist	MsgSize	Msgs/sec	CPU%	CPU mS per msg
no	1024	875.13	72.4	1.654
no	4096	808.84	75.93	1.877
no	16384	728.28	78.6	2.158
no	65536	492.97	82.58	3.35
yes	1024	247.96	32.48	2.619

7.10 The Effect of using coordinatedTransaction=yes

coordinatedTransaction=no, Transaction Mode=automatic, Using XML Messages

Persist	MsgSize	Msgs/sec	CPU%	CPU mS per msg
no	1024	265.81	55.06	4.142
no	4096	238.02	55.06	4.626
no	16384	189.86	55.44	5.84
no	65536	157.07	55.62	7.082
yes	1024	113.06	36.17	6.398

coordinatedTransaction=yes, Using XML Messages

Persist	MsgSize	Msgs/sec	CPU%	CPU mS per msg
no	1024	266.31	55	4.13
no	4096	238.69	55.02	4.61
no	16384	189.56	55.51	5.856
no	65536	156.89	55.69	7.099
yes	1024	114.44	36.61	6.398

7.11 The Effect of Increasing the Number of Subscribers

10 Subscribers Receiving MQRFH2 Type Published Messages

Persist	MsgSize	Msgs/sec	CPU%	CPU mS per msg
no	1024	197.9	67.01	6.772
no	4096	161.76	68.51	8.47
no	16384	119.37	66.02	11.061
no	65536	58.66	61.56	20.988
yes	1024	101.25	44.79	8.847

30 Subscribers Receiving MQRFH2 Type Published Messages

Persist	MsgSize	Msgs/sec	CPU%	CPU mS per msg
no	1024	81.52	63.8	15.652
no	4096	77.13	70.84	18.368
no	16384	61.12	69.09	22.607
no	65536	29.19	63.49	43.501
yes	1024	48.31	45.84	18.977

50 Subscribers Receiving MQRFH2 Type Published Messages

Persist	MsgSize	Msgs/sec	CPU%	CPU mS per msg
no	1024	54.33	62.7	23.081
no	4096	46.83	69.68	29.758
no	16384	39.12	69.59	35.577
no	65536	19.25	64.43	66.94
yes	1024	31.61	46.84	29.636

70 Subscribers Receiving MQRFH2 Type Published Messages

Persist	MsgSize	Msgs/sec	CPU%	CPU mS per msg
no	1024	39.06	62.17	31.833
no	4096	33.59	69.67	41.482
no	16384	26.16	69.18	52.889
no	65536	14.65	64.62	88.218
yes	1024	23.56	50.97	43.268

100 Subscribers Receiving MQRFH2 Type Published Messages

Persist	MsgSize	Msgs/sec	CPU%	CPU mS per msg
no	1024	18.03	63.96	70.948
no	4096	15.91	64.29	80.817
no	16384	13.18	63.94	97.025
no	65536	7.89	61.59	156.121
yes	1024	14.53	54.29	74.728

1000 Subscribers Receiving MQRFH2 Type Published Messages

Persist	MsgSize	Msgs/sec	CPU%	CPU mS per msg
no	1024	1.95	64.02	656.615
no	4096	1.36	63.82	938.529
no	16384	1.06	61.87	1167.358
no	65536	0.72	59.21	1644.722
yes	1024	1.58	57.58	728.86

8.0 APPENDIX C - COMPLEX COMPUTE NODE

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

8.1 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 &lt;= 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;
```

8.2 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 8.1 - Complex Compute Node**.

8.3 Very Complex Compute Node

The very complex compute node consisted of five repetitions of the logic for complex compute node (see **Section 8.1 - Complex Compute Node**) all contained within one compute node.

End of Document