MQSeries Integrator for Sun Solaris V2 Performance Report

Version 1

August 25, 2000

Tim Dunn

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".

First Edition, August 2000

This edition applies to Version 1 of MQSeries Integrator for Sun Solaris - V2 Performance Report and to all subsequent releases and modifications until otherwise indicated in new editions.

Notices

This report is intended to help the reader understand the performance characteristics and perform capacity planning for MQSeries Integrator for Sun Solaris - V2.0.1. The information is not intended as the specification of any programming interfaces that are provided by MQSeries or MQSeries Integrator for Sun Solaris - V2.0.1.

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
- Netfinity
- MQSeries
- MQSeries Integrator
- DB2

The following terms are trademarks of other companies:

- Solaris, Sun Microsystems, Windows NT, Microsoft Corporation
- NEONFormatter, NEONRules, NEON

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

Summary of Amendments

Date	Changes
25 August 2000	Initial release

Contents

1.0 CONCEPTS

MQSeries Integrator for Sun Solaris - V2.0.1 (MQSI V2) addresses 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. MQSI V2 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 V1.

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

1.1 Major Components

The major components of MQSI V2 are:

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

The Configuration Manager and Control Center must run on a machine running the Windows NT operating system. The Broker and User Name Server run on Solaris.

1.1.1 The Configuration Manager

The Configuration Manager is the main component of the MQSI environment. The components and resources managed by the Configuration Manager constitute the broker domain. The Configuration Manager servers 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 initialisation 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.

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 flow; these are known as additional instances.

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.

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 MQInput node. Most message flows will also contain at least one MQOutput or one Publication node, although this is not required.

A message flow is 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.

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 properties 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. 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.

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.

Within an execution group the assigned message flows run in different 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.

The broker guarantees operating isolation of each execution group, thus guaranteeing data integrity between execution groups and improving robustness of message flows.

1.3 Messages and Message Sets

In MQSI 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 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 header information:

1. *Message Domain* which identifies the message parser that will interpret the bit-stream of the message. By default MQSI 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, as it has been defined. 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 which do not have an MQRFH or MQRFH2 header, the input node uses the default message template, defined as a property of the input node, to determine how the message must be parsed.

1.4 Message Parsers

MQSI 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, MQCFH, MQDLH, MQIIH, MQMD, MQMDE, MQRFH, MQRFH2, MQRMH, MQSAPH 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, MQSI 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.

MQSI V2 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 MQSI V2 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 MQSI V2, 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 as well as having a compensating program to ensure that the queue manager queue cache did not overflow to disk. Where these two actions not taken the throughput possible with MQSI V2 would have been constrained by the necessary I/O processing of the queue manager with which the MQSI V2 broker was associated.

The performance measurements have focused on the throughput capabilities of the broker using different processing node types. The aim of the measurements was to be able to answer questions such how many messages a second can be processed with each of the node types, how much CPU does it cost to do, that and finally, how much memory is required.

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

- MQInput and MQOutput
- Compute
- Database
- Filter
- NEONFormatter
- NEONRules
- Publication

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. This should be borne in mind when performing capacity planning.

Recommended minimum specification machines are recommended for each of the MQSI V2 components.

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

All measurements used a broker that ran as an authorized MQSeries application in order to improve message throughput. This was achieved by use of the ‘-t’ flag on broker creation (the mqsicreatebroker command) and by ensuring that the environment variable MQ_CONNECT_TYPE=FASTPATH was present in the environment in which the broker was started.

The MQSeries queue manager listener was started by the INETD demon. This was the non threaded listener.

All measurements were driven by a multithreaded MQSeries Client program written in C. There were 30 threads used for each measurement, with all threads sending the same message format and content. 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 DB2 instance used with the broker was a default configuration with no tuning.

The message rates reported are the number of messages passing between the MQSeries multithreaded client and the MQSeries queue manager to which the input data is written and from which the reply data is read. Thus a client thread will issue an MQPUT to write a input message for a message flow and issue an MQGET to read the output message of the message flow. This results in a count of two messages. This method of reporting makes this report consistent with MQSeries performance reports. Those who wish to only deal with an input message rate should halve the message rates reported.

For an MQInput and MQOutput node it is possible to define transaction support for a node. Possible values are yes, no and automatic with the default value being yes.

- 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 MQSI V2 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 MQSI V2 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 which 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.

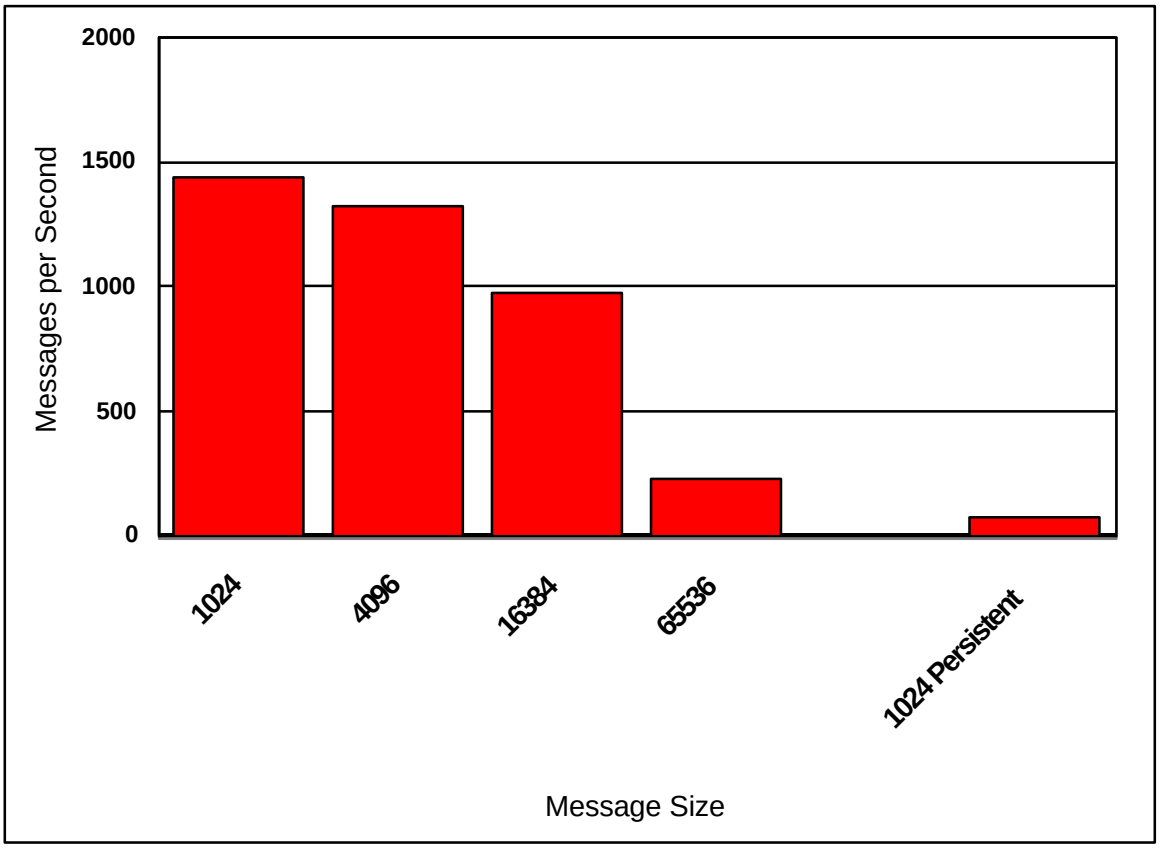

Figure 1: MQInput/MQOutput Throughput Results

With a 1K non persistent message it was possible to process approximately 1440 msgs/second. Increasing the message size to 4K had an effect on the throughput rate achieved (1322 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 behavior which is observed when using other applications within MQSeries.

Increasing that message size to 16K and beyond had a significant effect on the maximum message throughput which could be achieved. This decrease in throughput is as a result of the additional volume of data that must be managed both within MQSI V2 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 MQSI V2 is able to transfer messages from the input queue to the output queue for a single execution group. Adding additional execution groups 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 74 msgs/second.

When using 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 and a write to the file system for the message. The write to the file system may involve an I/O or not. This is dependent on the file system and is not forced by MQSeries.

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 to 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. 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 input message consisted of a single field. 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 message flow with varying message sizes and persistence. There was a single instance and single execution group running the message flow.

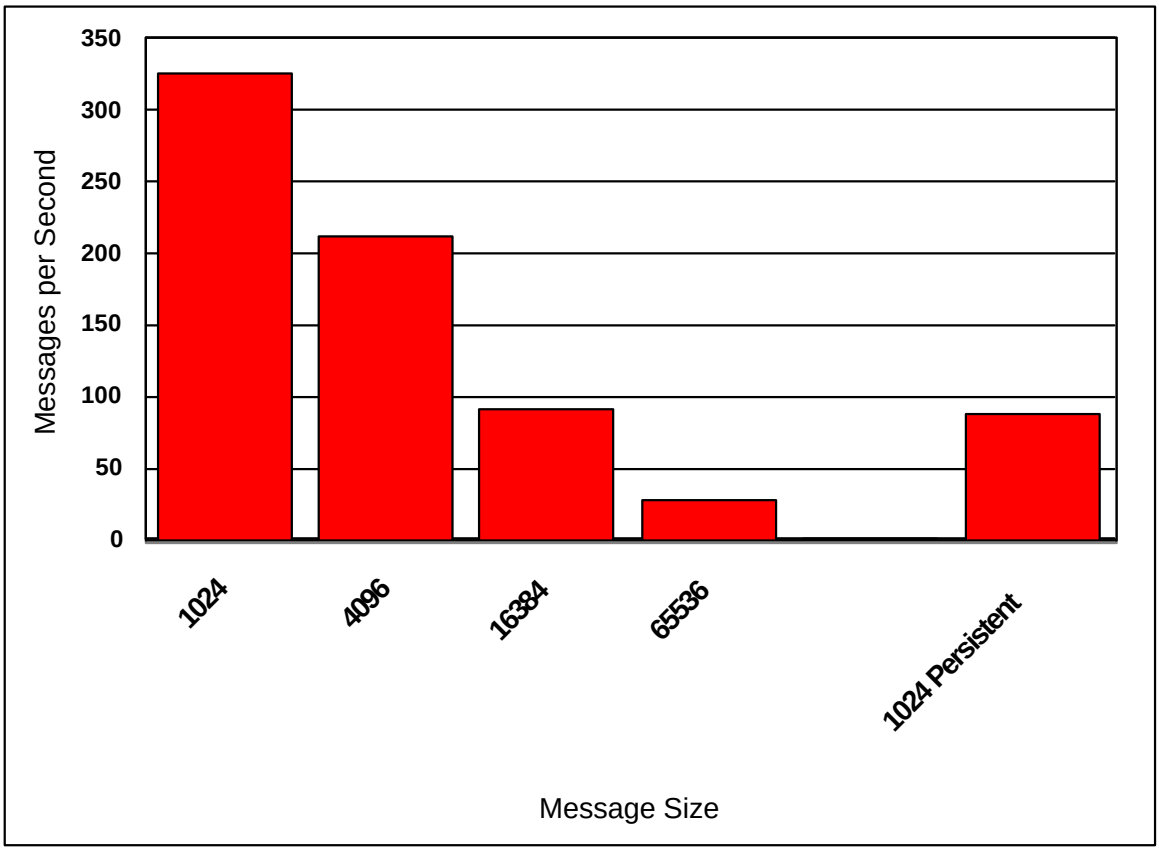

Figure 2: Simple Compute Node Throughput Results

With a 1K non persistent message it was possible to process 325 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 is was possible to process approximately 87 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. Because of the size of the message required to run this test it was not possible to measure with a message size of 1024 bytes.

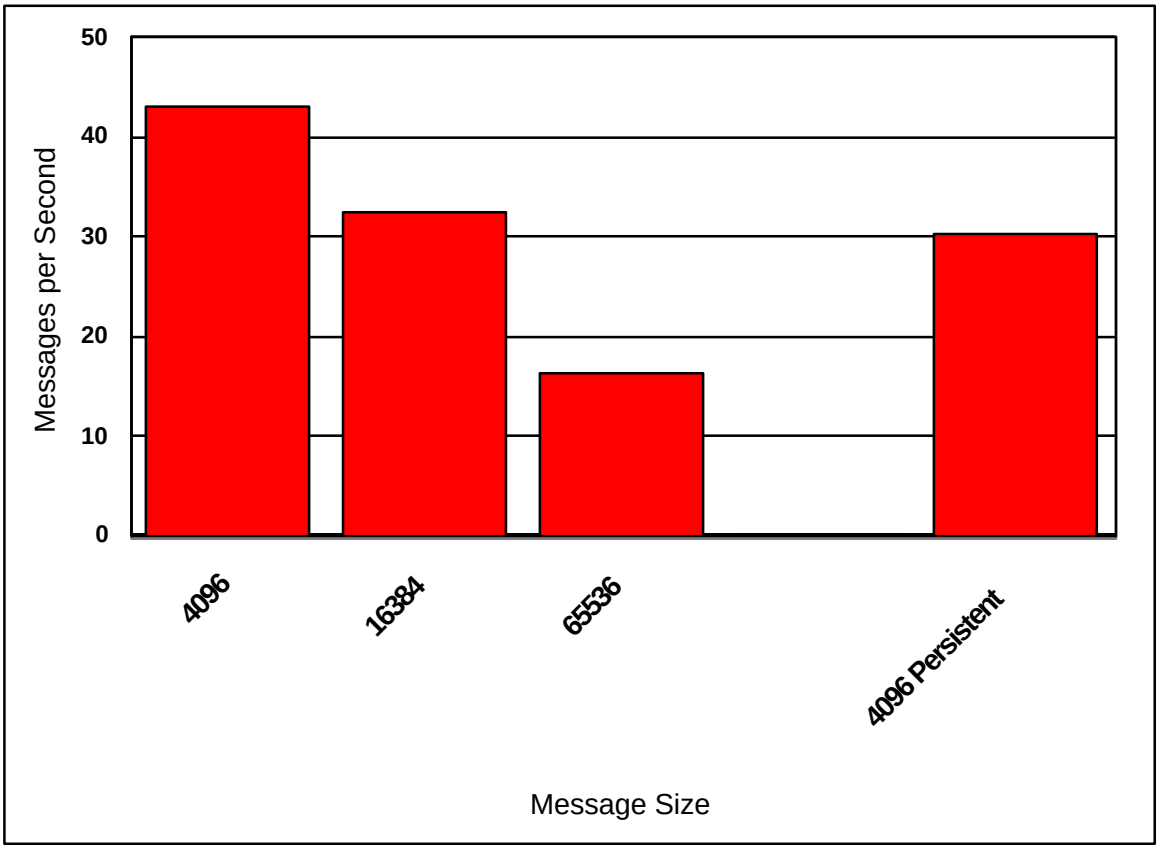

Figure 3: Complex Compute Node Throughput Results

With a 4K non persistent message it was possible to process 43 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 is was possible to process 30 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. Because of the size of the message required to run this test it was not possible to measure with a message size of 1024 bytes.

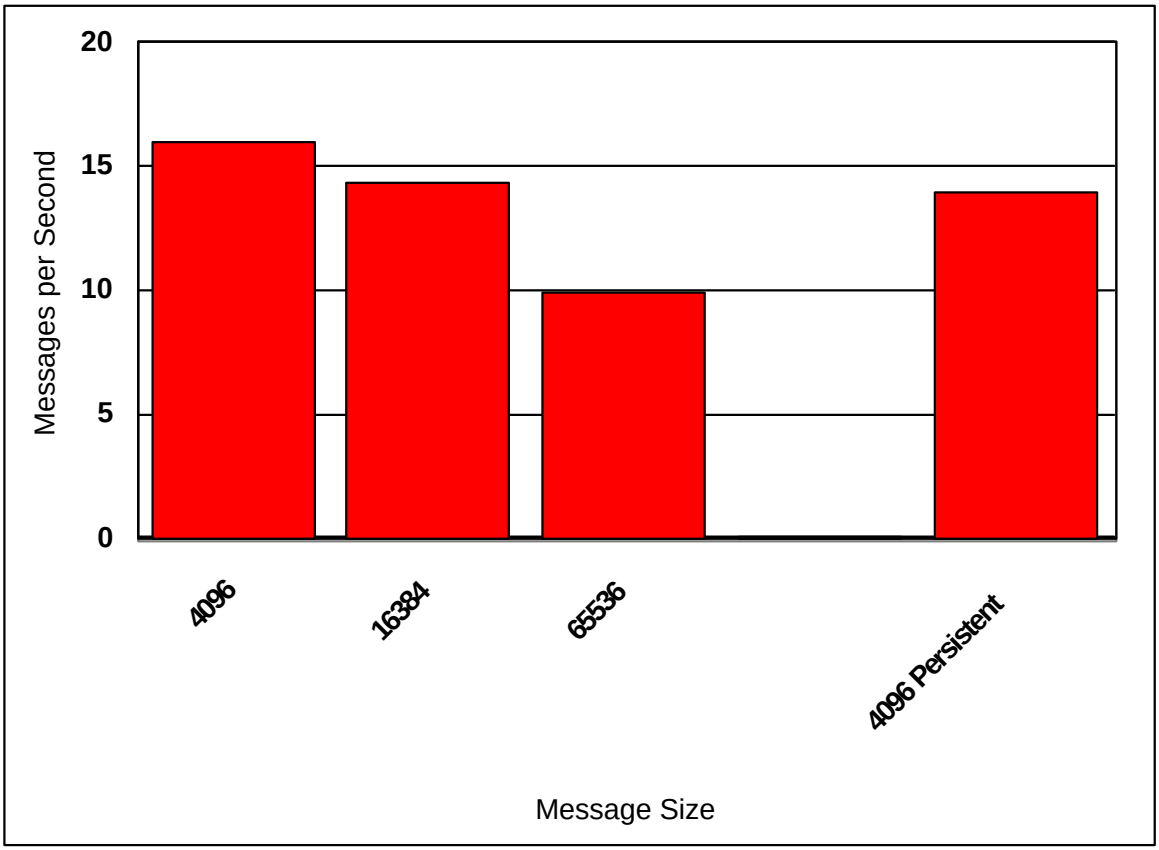

Figure 4: Multiple Complex Compute Node Throughput Results

With a 4K non persistent message it was possible to process 16 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 is was possible to process 14 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**.

2.2.4 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. Because of the size of the message required to run this test it was not possible to measure with a message size of 1024 bytes.

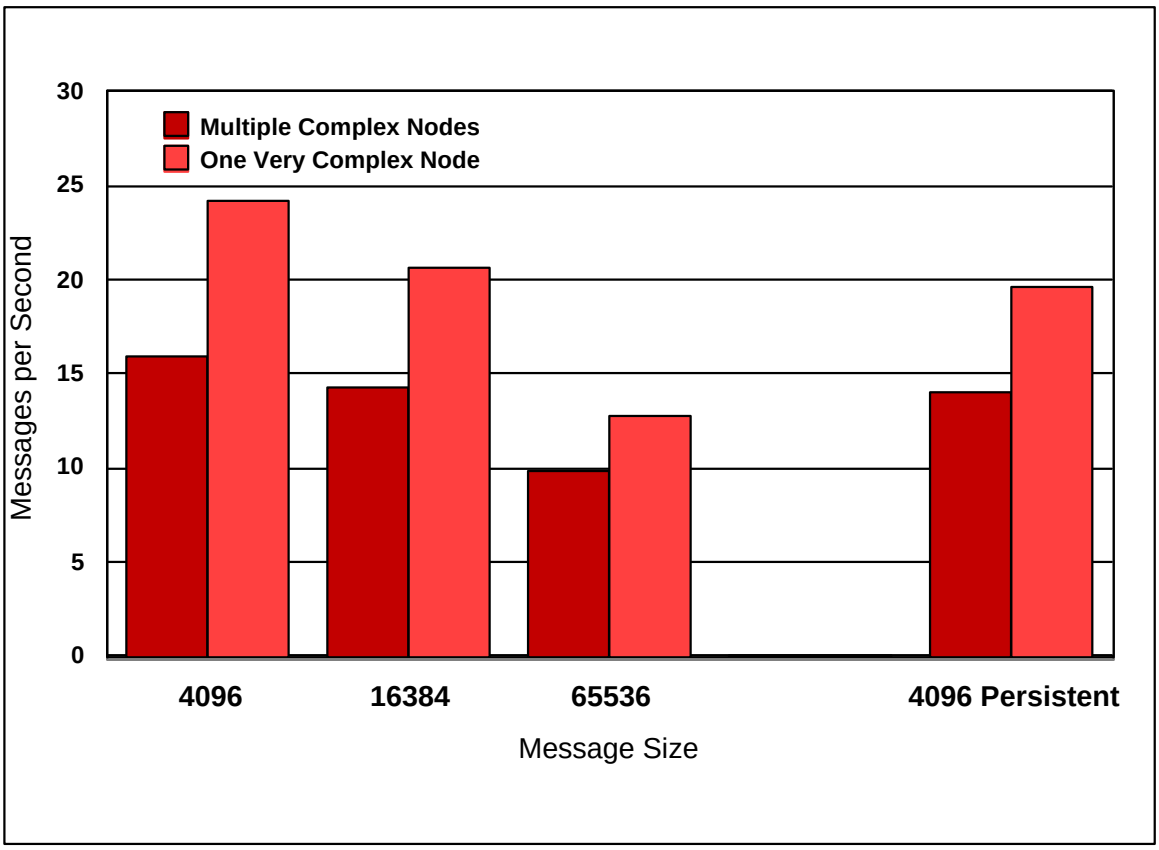

Figure 5: Very Complex Compute Node VS Multiple Complex Compute Node Throughput Results

With a 4K non persistent message it was possible to process approximately 24 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 is was possible to process approximately 20 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.

For comparison purposes Figure 5 also shows the message throughput rates which 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 50% 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.

In order for the database transaction to be part of a global unit of work that incorporates the processing of the message within the same transaction, the MQSeries queue manager associated with the MQSI V2 broker must have a suitable X/Open XA interface connection with the database. The MQSeries queue manager associated with the MQSI V2 broker acts as the transaction coordinator. The database is a resource manager. In order for the global unit of work to function correctly the database used must be DB2.

Without establishing such an XA connection it would not be possible for the database manager to commit (or backout) the database transaction at the same time as the message processing updates in the same unit work. This would mean data was an inconsistent state.

In this simple test to illustrate the effect of using a database node an XA connection was configured between the MQSeries queue manager and the database, DB2 in this case. 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 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 which where obtained for varying message size and persistence.

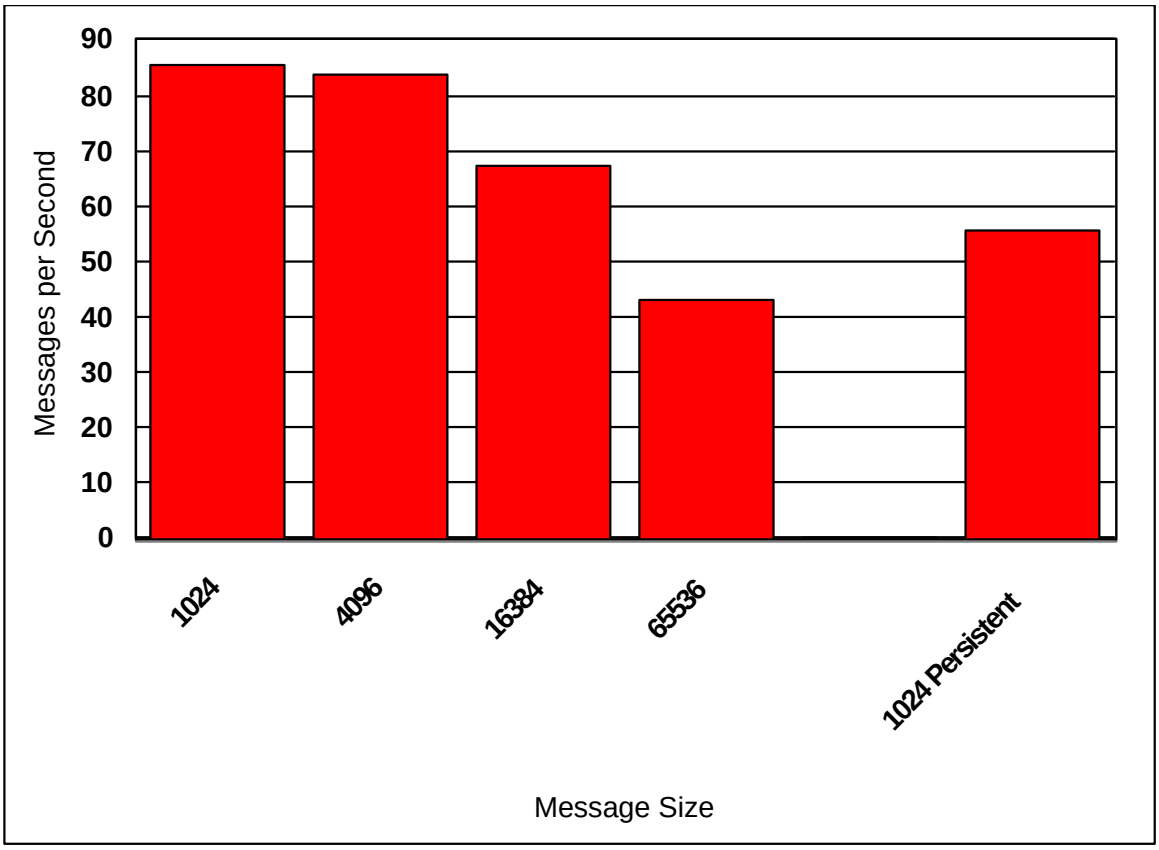

Figure 6: Database Insert/Delete Throughput Results

With 1K non persistent messages it was possible to achieve a message throughput rate of approximately 86 msgs/second. This is 43 database insert and deletes per second. The rate of insert/delete activity reduced with message size as expected.

With 1K persistent messages it was possible to achieve a message throughput rate of approximately 56 msgs/second. This is 28 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.

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 an XML message. The transaction mode on the MQInput and MQOutput nodes was set to automatic.

Figure 7 below shows the results which 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.

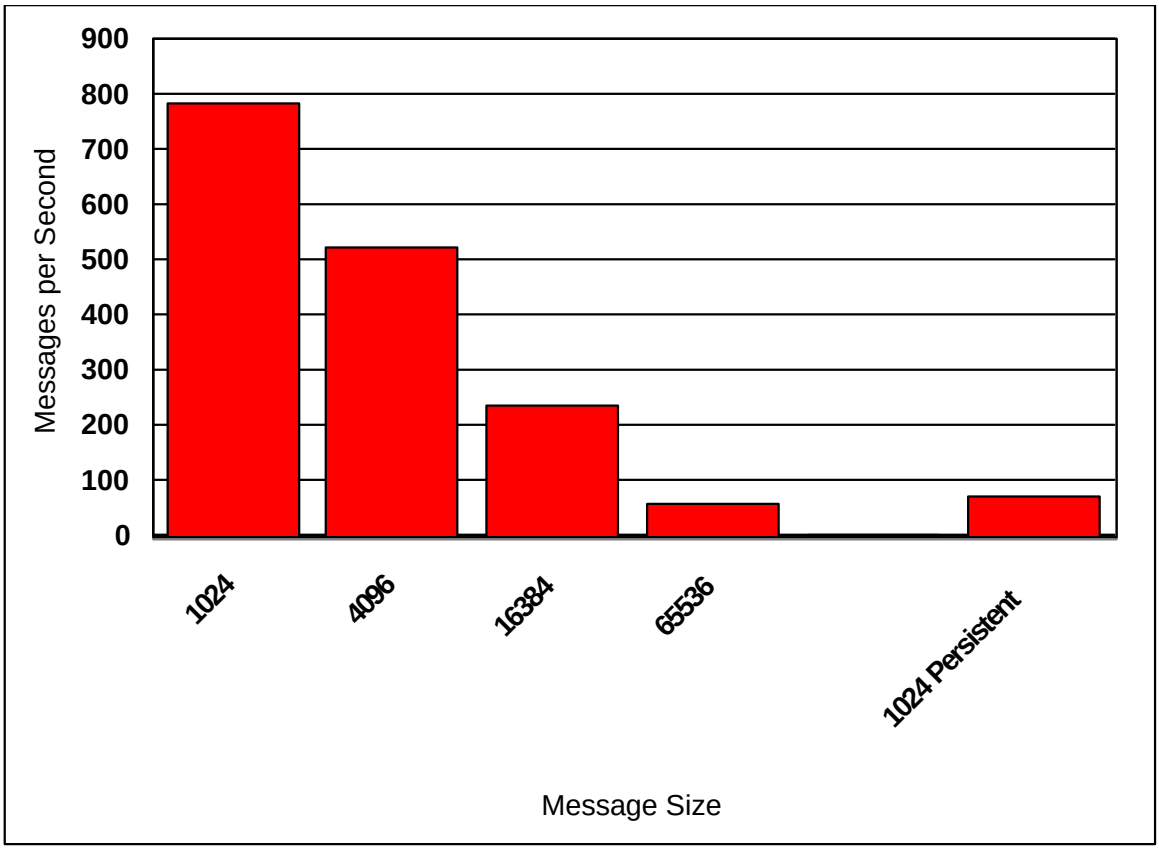

Figure 7: Filter Node Throughput Results

With 1K non-presistent messages it was possible to run 786 msgs/second. The difference between this rate and the 1443 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 73 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 Filter Node Throughput measurements is available in **Section 7.4 - Filter Node Throughput Results.**

2.5 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.

In the throughput measurements each client thread performed the role of publisher and subscriber. 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. At this point the client thread was a subscriber.

With this mode of operation there were:

- Two messages between the MQ client and queue manager
- One published message
- One subscriber

Figure 8 below shows the results which were obtained as a result of running the message flow with MQRFH2 messages of varying 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 received by the subscriber.

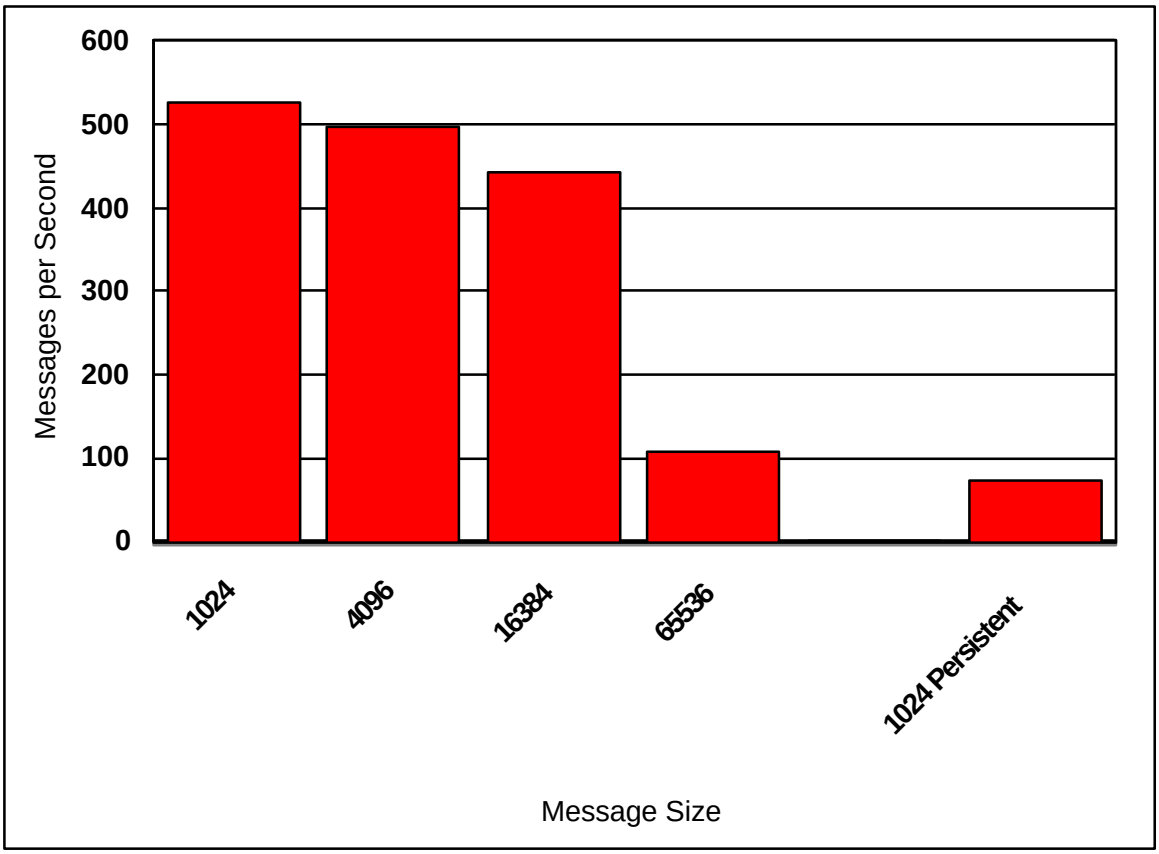

Figure 8: Publication Node Throughput Results

With 1K non-presistent messages it was possible to run 526 msgs/second. This represents a publication and subscription rate of 263 messages per second.

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

With 1K persistent messages the throughput was approximately 74 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 7.5 - Publication Node Throughput Results**.

In a separate measurement the overhead of using Access Control for topic publications was measured. The measurement consisted of a 1K non persistent message size and 50 subscribers. The overhead was an 18% reduction in message throughput.

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

MQSI V2 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 MQSI V2 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 30 input fields, with 10 fields consisting of a short string (12 characters), 10 fields consisting of a floating pointer number and 10 integer fields.

The format conversion was achieved using a Compute node with suitable ESQL statements. The input messages contained an MQRFH2 header in which the message type was set. The output format was specified in the Compute node processing. Each message format was converted to Generic XML, CWF and MRM XML and the message throughput achieved was measured. There was single execution group running the message flow and no additional instances specified. The results in msgs/second obtained are presented in **Table 1** below.

Conversion	**TO**	Generic XML	CWF XML	MRM XML
FROM				
Generic XML		96.90	59.40	59.60
CWF XML		46.80	31.00	31.00
MRM XML		54.60	40.80	41.20

Table 1: Message Rates 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 MQSI and 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.

Because of the cost of conversion between different message formats it is recommended that is used only where necessary.

The detailed measurement data for cost of message conversion is available in **Section 7.6 - Converting Messages Between Formats Throughput Results**.

2.7 How Well Do NEONRules and NEONFormatter Run Within MQSI V2?

MQSeries Integrator V1 consisted of NEONRules and NEONFormatter components. It is possible to continue to use those components, unchanged, in MQSI V2.

Throughput measurements were taken in order to show the relative performance of using NEONRules and NEONFormatter within MQSI V1and MQSI V2 node. Two test cases were used.

In the first case an input message with 50 fields was copied to an output message with 1 NEON rule present. The purpose of this measurement was to show the cost of reformatting a message using NEONFormatter. In the second test case the 50 field message was again copied but there were 100 rules defined, only 1 of which was executed - all are evaluated though. The purpose of this measurement was to show the effect of having an increasing number of rules to evaluate.

In both test cases three copies of the NEONRules engine were run. A NEONRules node was used when running within MQSI V2. The same MQSeries input and output queues were used for all 3 copies of the NEONRules engine. When NEONRules was run within MQSI V2 it was run with the transaction mode for the node set to yes so as to be consistent with the transactional support provided by NEONRules in MQSI V1.

Figure 9 below shows the results which were obtained as a result of running with 1 rule defined when using the NEONRules engine within MQSI V1 and within MQSI V2.

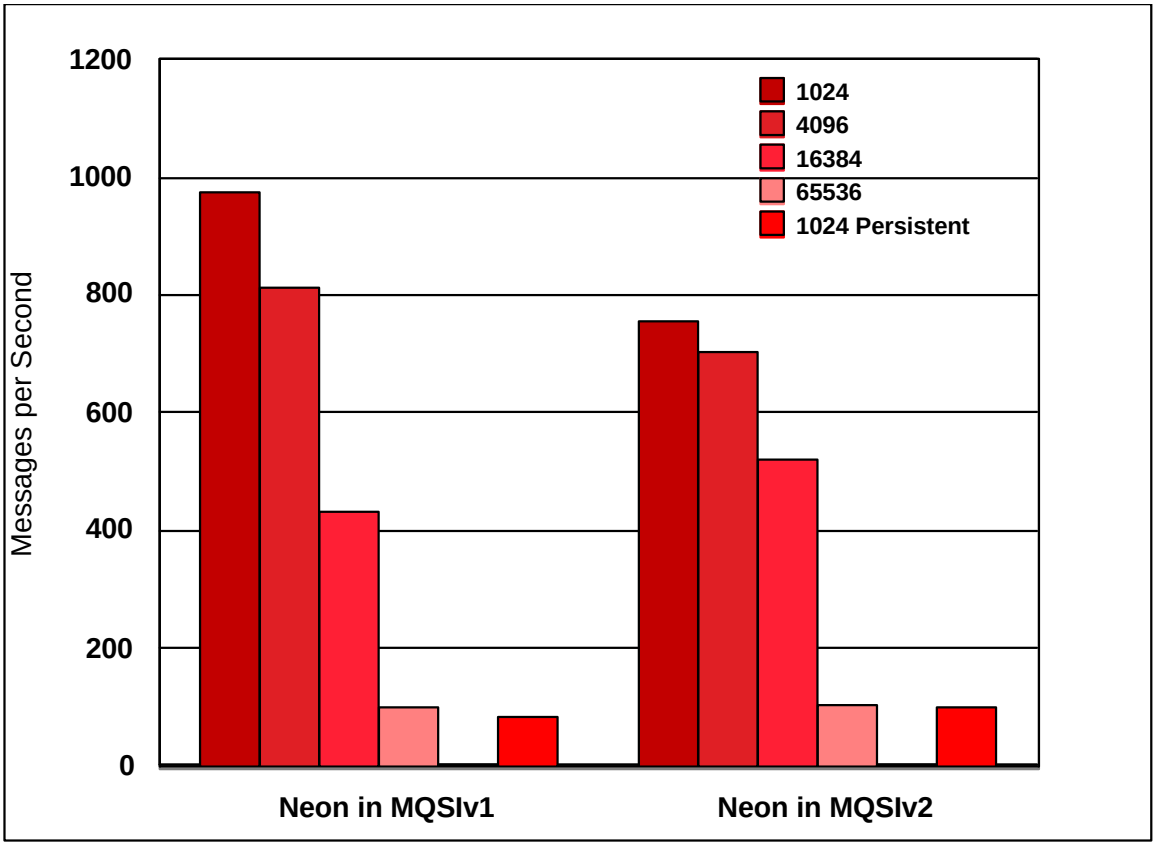

Figure 9: NEONRules Engine With One Rule Throughput Results

The results show that it was possible to achieve about 80% of the message throughput when running NEON in MQSI V2 compared with running it in MQSI V1 for 1K non persistent messages

With 1K persistent messages there is an improvement in the message throughput rate of 21% when running NEON within MQSI V2 compared with running NEON in MQSI V1.

Figure 10 below shows the results which were obtained as a result of running with 100 rules defined but only 1 executed when using the NEONRules engine within MQSI V1 and within MQSI V2.

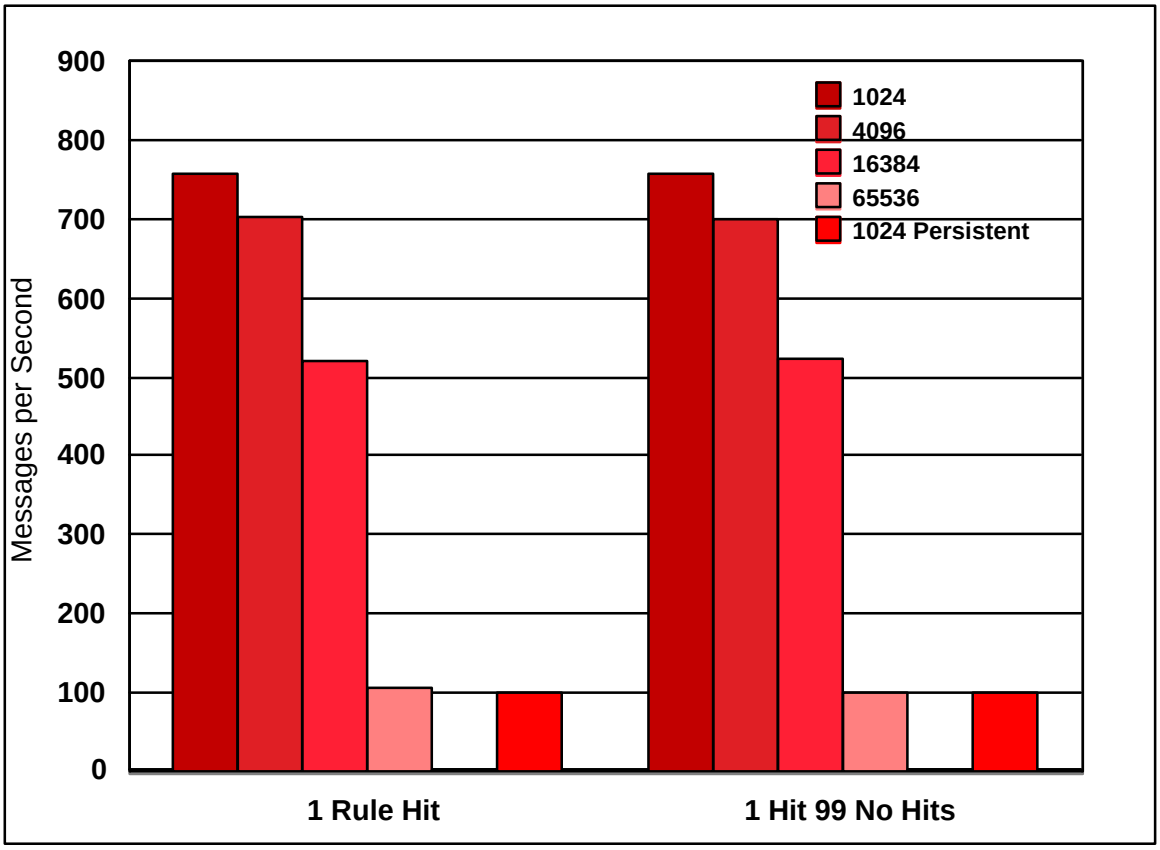

Figure 10: NEONRules Engine With One Hundred Rules Throughput Results

The results show that there was no noticeable difference in message throughput when running with 1 or 100 rules defined within NEONRules when only 1 rule was executed. This was true for both persistent and non persistent messages.

The detailed measurement data showing the effect of running the NEONRules engine within MQSI V2 is available in **Section 7.7 - NEONRules and NEONFormatter Within MQSI V2**.

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

A message flow runs within an execution group. An execution group is implemented as an operating system process. It is possible to have multiple execution groups running the same message flow. Reasons for doing this would be to achieve greater message throughput than is possible with a single execution group and to provide some level of redundancy should an execution group process fail.

Figure 11 below shows the results that were obtained as a result of running one, two and four execution groups for a complex compute message flow 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. Because of the size of the message required to run the test the minimum message size for these measurements was 4K.

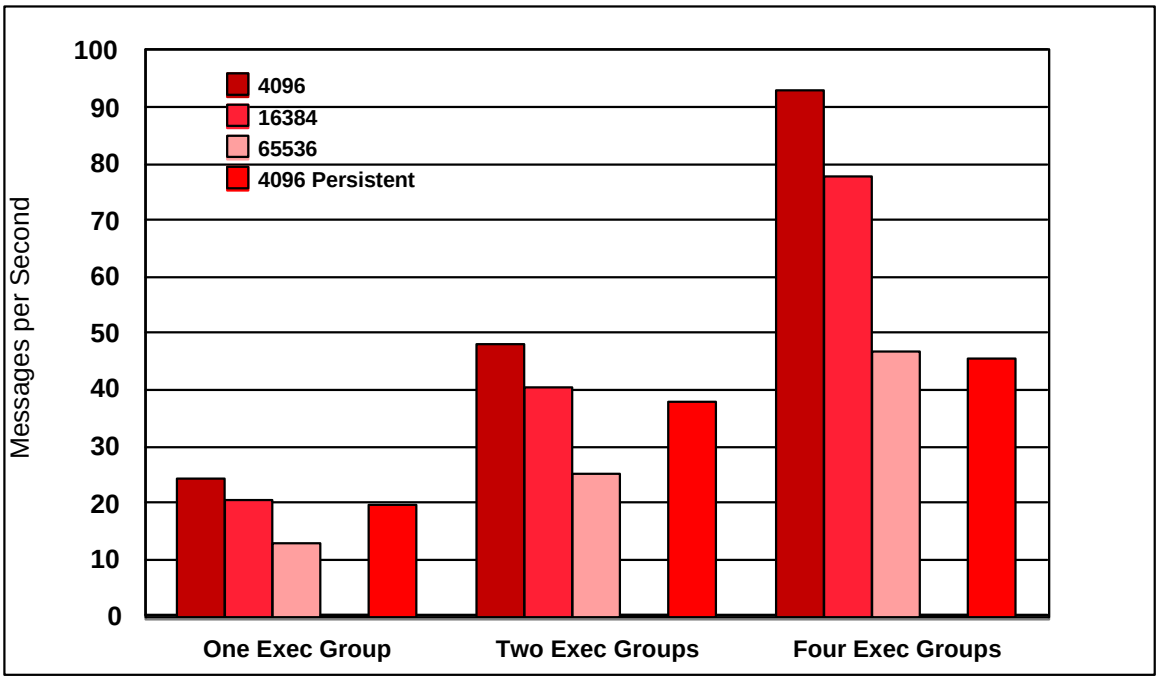

Figure 11: Additional Execution Group Throughput Results

Figure 11 shows that greater message throughput can be achieved by using additional execution groups. The best scaling results were obtained with non persistent messages. For the 4K message size, 4 execution groups gave 3.84 times the throughput of a single execution group. For 64K messages, 4 execution groups gave 3.66 times the throughput of a single execution group.

With persistent messages there was benefit from running with an increasing number of execution groups. The best improvement was obtained with 2 execution groups, when the throughput was 1.93 times that achieved for a single execution group. Running with yet another 2 execution groups gave little additional benefit as the throughput for 4 execution groups was only 2.31 times that achieved for a single execution group. The poor scaling achieved with the third and fourth execution is as a result of contention for the MQSeries queue manager log. The queue manager log in this case was located on a SCSI disk. By locating the queue manager log on a faster device, such as one with a fast write nonvolatile cache it is most likely that an improved scaling ratio would be seen for persistent messages with a higher number of execution groups.

These measurements show that there can be significant improvements in message throughput as a result of running multiple execution groups. The message flow contents will have a significant effect

on the ability to scale throughput as will the message type, (non persistent vs persistent). In the message flow used for these 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.

The detailed measurement data showing the effect of adding execution groups is available in **Section 7.8 - 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 12 below shows the results that were obtained as a result of running the message flow with varying message sizes and persistence.

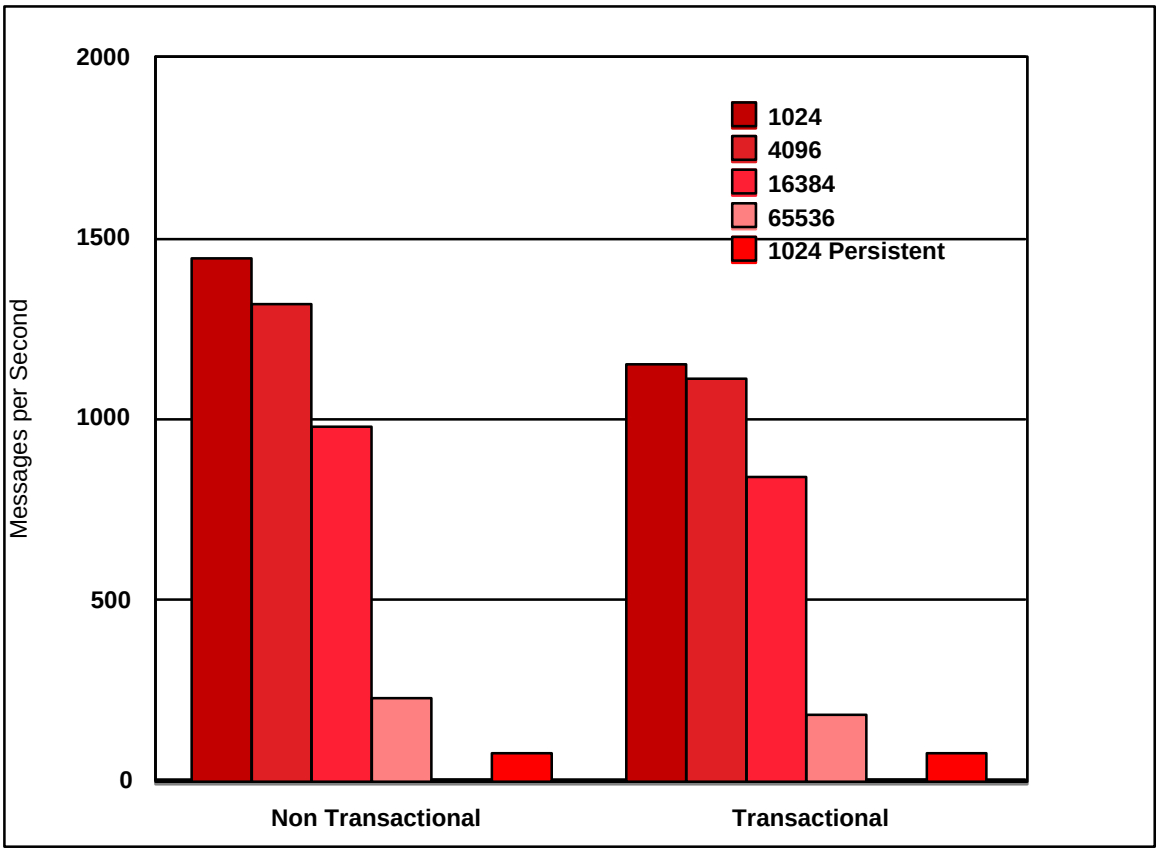

Figure 12: Making a Message Flow Transactional Throughput Results

Making non-persistent messages transactional had a noticeable 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 no difference in throughput as persistent messages would proceed under transaction control anyway with a transaction mode of automatic.

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

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

Specifying a value of yes for the coordinatedTransaction 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. In order to ensure that the global unit of work is coordinated correctly a suitable XA connection must be configured between the MQSeries queue manager and the external relational data manager.

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 costs associated with using the coordinatedTransaction 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 cost of an XA coordinated transaction.

Figure 13 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 an XA connection was configured and a value of yes was specified for coordinatedTransaction on the message flow and transaction mode was set to automatic on the MQInput and MQOutput nodes.

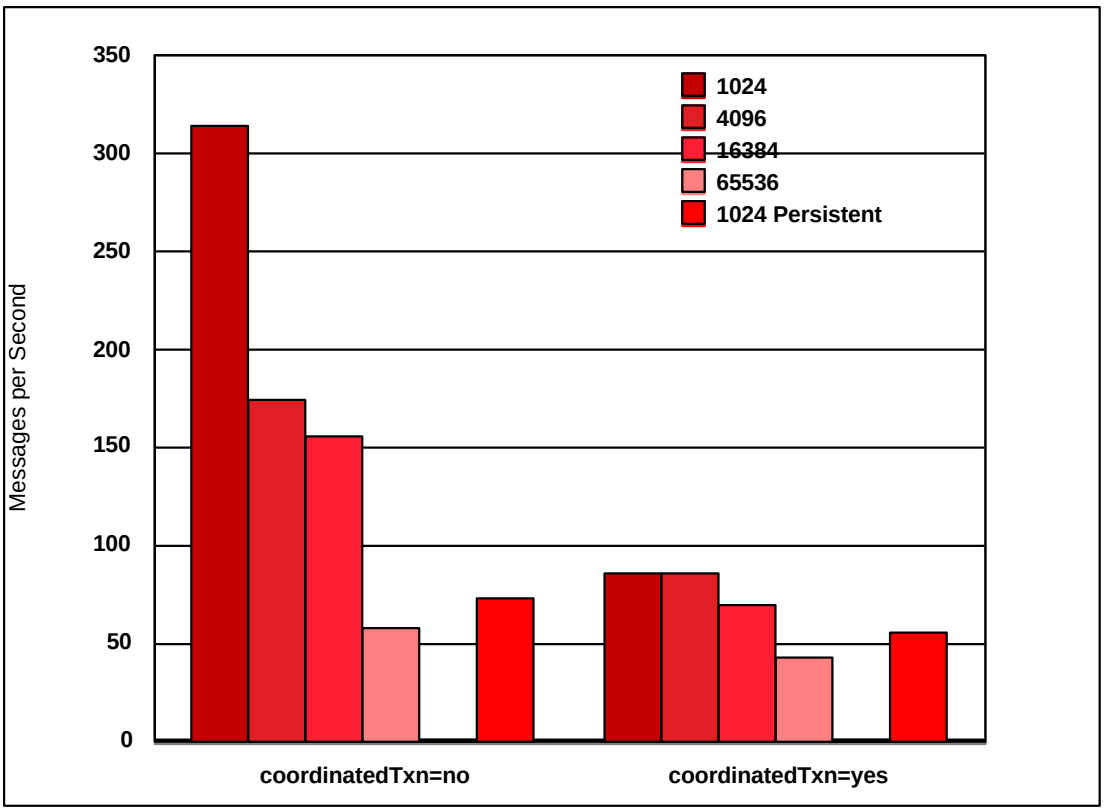

Figure 13: Database Update with XA and coordinatedTransaction=yes.

Figure 13 shows that a message rate of approximately 314 msgs/second was achieved for 1K non-persistent messages and 73 msgs/second for persistent messages when coordinatedTransaction was set to no on the message flow, transaction mode was set to automatic on the MQInput and MQOutput nodes and there was no XA connection configured.

With an XA connection configured and coordinatedTransaction set to yes on the message flow a rate of 85 msgs/second was achieved for non persistent messages and 55 msgs/second for persistent messages.

The difference in message rates represents the overhead of coordinating the updates as a single unit of work. In this case it varies between 73% and 20% based for non persistent messages with the higher overhead being incurred for the smaller messages. For persistent messages the overhead is 25% for a 1K message size. The database processing in these measurements was simple. In practice it would typically be more involved, in which case the overhead imposed as a result of using coordinatedTransaction=yes would be less as a percentage of the total cost.

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 14** 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.

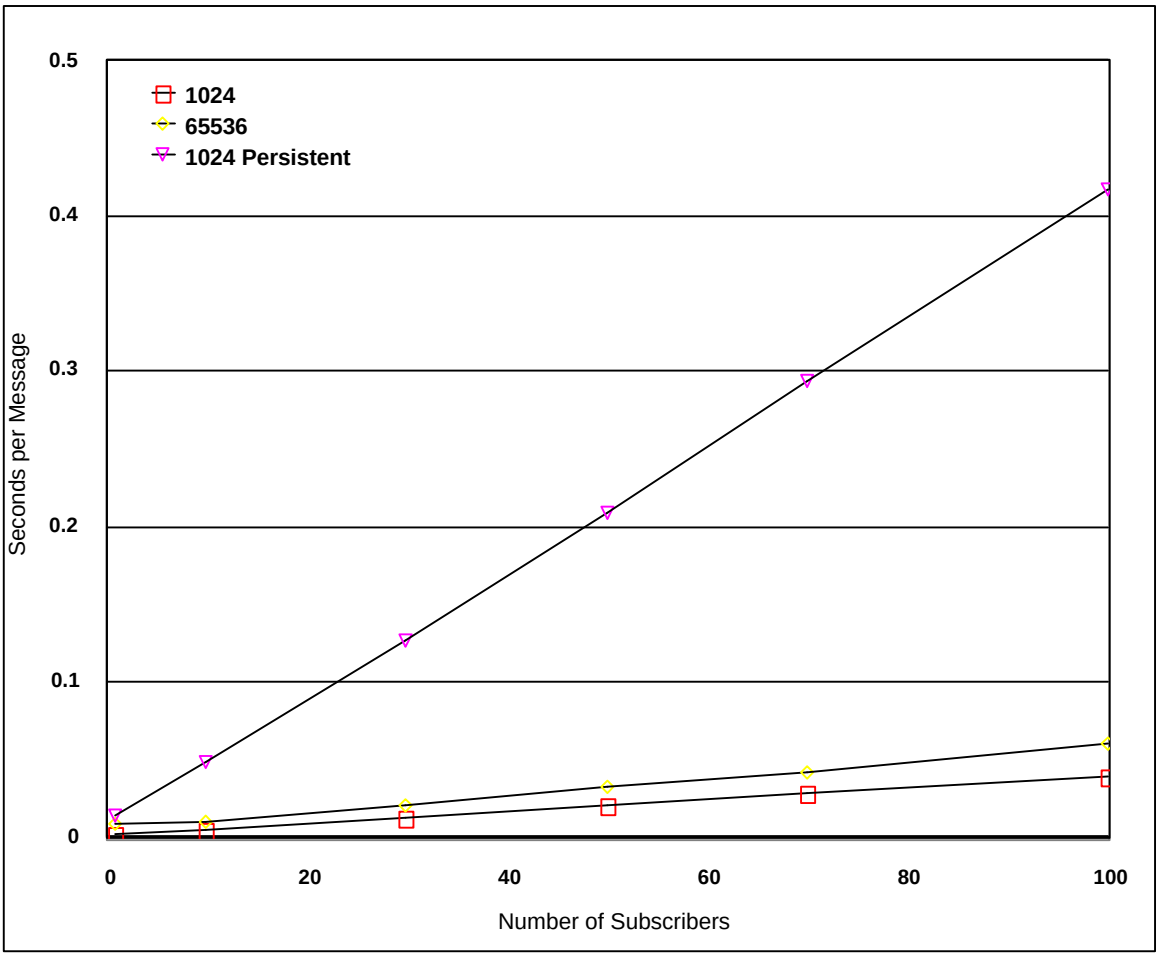

Figure 14: Varying Number of Subscribers

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. This is reflected in the steeper gradient of the 1K persistent message measurements.

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 2 was registered for the purposes of reporting message rates, although the MQSI V2 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.

A measurement taken for 1000 subscribers showed that it was possible to achieve 2.3 msgs/second for non persistent messages and 0.2 msgs/second for persistent messages. These numbers were not shown in Figure 14, since they would have distorted the presentation of the results.

The 1000 subscriber measurements show that the cost of handling between 100 and 1000 subscribers is linear. The measurement with 100 subscribers gave a message rate of 25.3 msgs/second for 1K non persistent messages. Dividing this rate by 10 should give the projected rate for 1000 subscribers. The projection is 25.3/10 = 2.53 msgs/second. When actually measured the result was 2.3 msgs/second. It is safe therefore to make an estimate of varying numbers of subscribers between the limits of 100 and 1000.

From some additional measurements which were taken it has been shown that the performance obtained when using publication nodes is dependent on the number of subscriptions which receive messages as a result of a message being published rather than the total number of registered subscribers. For example consider the case where there are 1000 subscribers registered to receive messages on two topics, with 990 registered for Topic 1 and 10 for Topic 2. When a message is published on Topic 2 it will only be the cost of publishing the message to the 10 subscribers registered for Topic 2 that will be seen. The fact that there are 1000 subscribers in total has no noticeable effect on performance.

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 to exceed the number of registered subscribers.

The mqsichangeproperties command used to increase the cache for the 1000 subscriber measurements was as follows:

mqsichangeproperties CSIM -e pubsub1 -o ComIbmMQConnectionManager -n queueCacheMaxSize -v 1050.

Where CSIM was the name of the MQSI Broker and pubsub1 was the name of the execution group running the message flow. This command was issued on the machine on which the broker was running.

In one measurement there was a 60% increase in message throughput as a result of increasing the cache size.

The command must be issued for each execution 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 7.11 - The Effect of Increasing the Number of Subscribers**.

2.12 What Is The Effect of Using Content Based Publish/Subscribe?

With MQSI V2 it is possible for a subscriber to register to receive publications based on the contents of a message. This is known as content based routing. This is more complex than topic based routing (in which a subscriber receives all messages on the topic to which they have subscribed) because the message contents must be examined.

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

Figure 15 below shows the message throughput rates that were achieved for content and topic based routing with varying message size and persistence. The topic and content based routing used MQRFH2 messages.

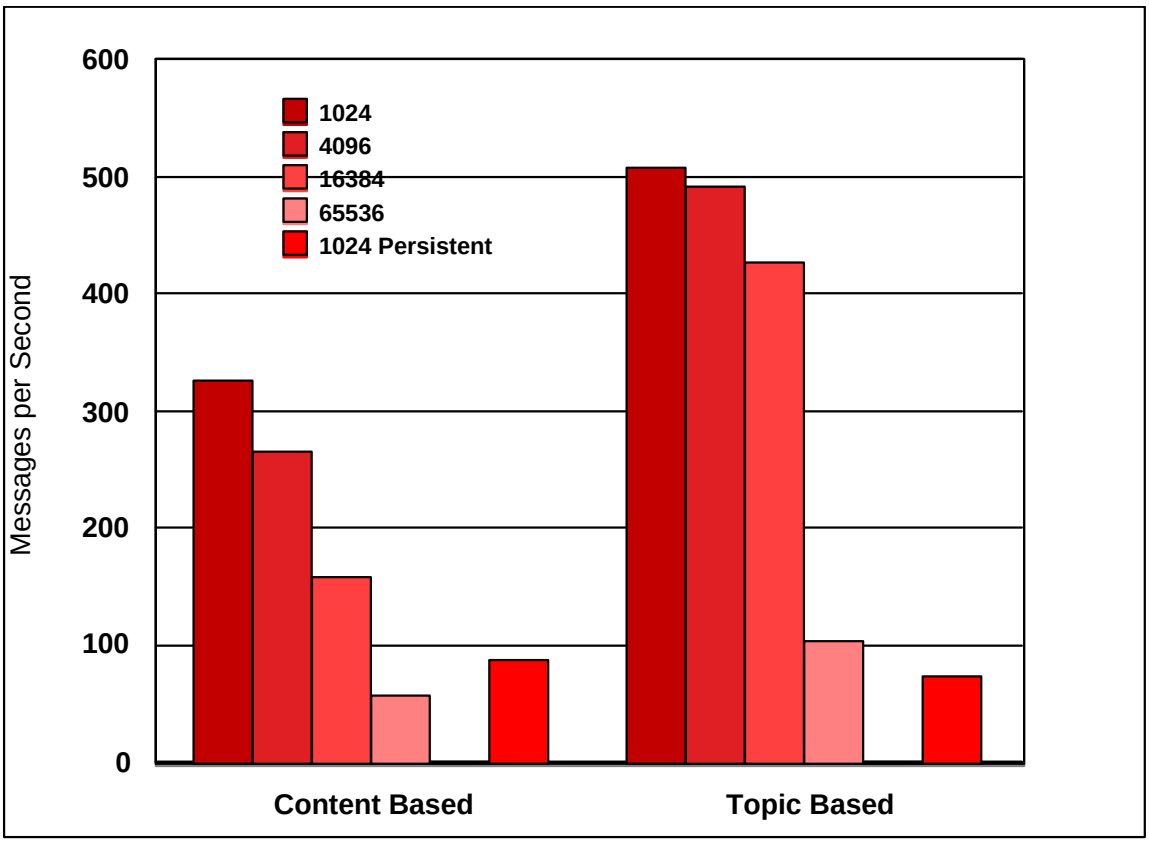

Figure 15: Content vs Topic Based PubSub

Figure 15 shows that it is possible to achieve greater message throughput using topic based routing when compared with content based routing. This is as expected since topic based routing is able to publish a message without regard to the message contents. Content based must parse the message contents, then determine which of the subscribers is to receive the message before finally publishing to those subscribers.

The detailed measurement data showing the comparison of content and topic based PubSub is available in **Section 7.12 - Content vs Topic Based PubSub**.

2.13 What Is The Effect of Examining an Increasing Number of Fields in a Message?

When message manipulation is field related, as opposed to copying the entire message for example, the number of fields makes a difference to the processing that is required to process the message.

In order to illustrate the effect of processing an increasing number of fields in a message, message throughput was measured for an XML message with a single field and compared with the results for a 50 field message when processed by a compute node.

Figure 16 below shows the message throughput rates obtained with varying message size and type.

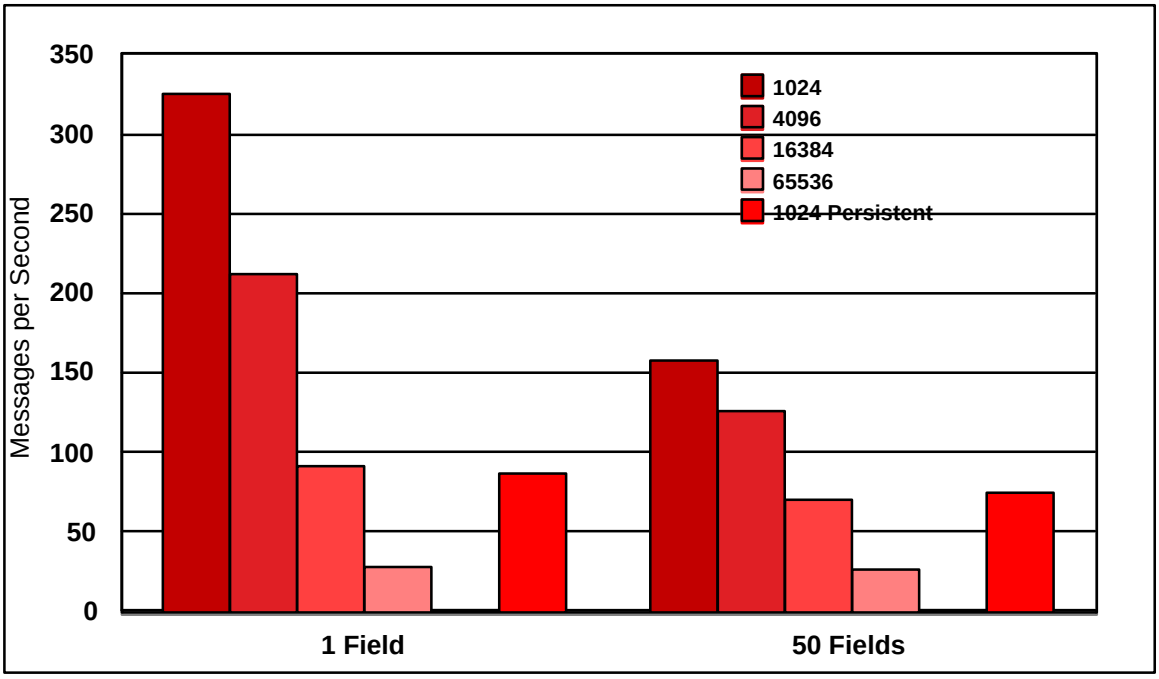

Figure 16: The Effect of an Increasing Number of Fields.

With non persistent messages for all message sizes it was possible to process a greater throughput of messages where there was a single field in the input message. The number of messages processed with persistent messages was very similar. This is because message processing is dominated by MQSeries log I/O.

A recommendation from this measurement is to minimize message size and number of fields where possible.

The detailed measurement data showing the effect of an increasing number of fields in a message is available in **Section 7.13 - The Effect of an Increasing Number of Message Fields**.

2.14 What Is The Effect of Using User Trace?

On occasions it is necessary to trace the course of processing within MQSI V2. There are 2 levels of trace available, system trace and user trace. There are also two modes of trace, safe and fast. With safe mode trace entries are always written to disk. This mode involves a greater amount of I/O activity to the trace file in order to ensure that all entries are written. With fast mode, data is buffered in an internal buffer and subsequently written to disk.

From measurements taken with MQSI for Sun Solaris - V2 there is an overhead of 98% when using user trace with a level of normal and mode of safe. The overhead is most significant for the smaller non persistent message sizes. The overhead is high because of the use of the safe mode, which immediately forces writes of trace data to disk.

2.15 What Is The Effect on Message Throughput of Using Faster Disks?

With persistent messages the limiting factor for message throughput is the speed at which the MQSeries queue manager log is able to operate at. The lower the seek time and latency of the disk on which the MQSeries log is located the more beneficial the effect on message throughout.

A series of measurements were taken with MQSI For Windows NT - V2 using SSA disks with a fast write cache defined for the disk. Although the measurements were taken on the Windows NT platform there is every expectation that the same results would be obtained on a Solaris system with similarly fast disks .

The measurements showed that it was possible to obtain a three to four times increase in message throughput rate. The fast write cache was the key to the improvement in message rate. When running with the MQSeries queue manager log located on a SSA disk without a fast write cache defined the message throughput rate resulted in a similar message rate to that achieved with a SCSI disk.

Further details of the measurements can be obtained from Version 1.1 of the MPI4 Supportpac.

3.0 CAPACITY PLANNING

This section gives general guidelines on capacity planning for MQSI V2. A method to provide detailed capacity planning which takes into account the contents of a message flow will be provided in the near future. The remainder of this section provides general guidelines to assist with the implementation of MQSI V2.

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, 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, will the deployed system be working within known limits ?

The way in which MQSI V2 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 MQSI V2. It is possible, however, to provide a series of guidelines 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 MQSI V2 to process messages at the required rate, examine the various throughput measurements detailed in Section 2.0 - BROKER THROUGHPUT MEASUREMENTS. This section details the highest rates at which MQSI V2 has been run.

Projecting measurement results to other machine types is difficult because performance depends on many factors, such as processor speed and instruction cache sizes. The only method currently available to gauge relative machine performance is to use published performance figures such as SPECInt benchmark results which are released by manufacturers for their hardware.

3.1 Throughput

Key factors affecting the throughput rate that is achievable are:

- Use of non-persistent vs persistent messages.
- The types of MQSI V2 nodes being used.
- The amount of processing in the nodes, simple vs complex compute nodes for example.
- The number of MQSI V2 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 disk 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. The measurements contained in this report are based on the use of Solaris E450 400 Mhz processors. 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 MQSeries queue manager log is located on a dedicated fast disk. When running persistent messages CPU utilization will be lower than for non persistent messages and you would not expect to be able to fully utilize processors in the same way as can be done for non persistent messages.

3.2 Scaling Message Throughput

The addition of multiple execution groups 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 MQSI V2: resolving item 1 above depends on having sufficient hardware of sufficient speed (CPU and disk) available. This is a principally a planning issue; For item 2, MQSI V2 provides the ability to schedule multiple pieces of work in parallel, for example by the use of multiple execution groups; 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. 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. Consider the following cases as an illustration:

Case 1

In a very simple message flow where messages are simply being copied from one queue to another as fast as possible, the potential to keep increasing message throughput by the addition of more execution groups is limited because of contention for access to the queue in MQSeries. This is even more of an issue with persistent message since queues are locked for longer and there must be I/O to the MQSeries log.

Measurements show that when using 1, 2, and 4 execution groups to perform copying of non persistent messages from one queue to another at the maximum rate possible that there can be some benefit (maybe an additional 20% in throughput) from using a second execution group but there after there will be reduced benefit and even a decrease because of the contention for queue access. With a higher number of execution groups context switching between processes can rise significantly and although the processors may become busier the level of useful work does not rise. One way around this contention would be to use multiple message flows each with its own pair of input and output queues.

It is also possible to show an increase in throughput of persistent messages. The maximum number of persistent messages which can be processed will be lower than for non persistent messages since there must be I/O to the MQSeries log. In order to maximize message throughput the MQSeries log I/O processing time must be reduced. This can be done in several ways. The most effective method would be to use a solid state disk on which to locate the MQSeries queue manager log. Another possibility is to use a disk subsystem with a fast write nonvolatile cache.

A message flow that has a significant amount of processing over and above any queue processing has much greater potential for increasing message throughput by the addition of more execution groups for example.

The increased throughput rates are possible because firstly the level of queue access overall is much lower, and secondly there is processing that each execution group can perform independently.

Case 2

The Complex Compute node described in Appendix C - Complex Compute Node is a good example of a message flow that scales well. This message flow was measured with 1, 2 and 4 execution groups. The results are presented in Section 2.8, What Is The Effect Of Increasing the Number of Multiple Execution Groups.

The results of the measurements show that it was possible to achieve almost perfect scaling in message throughput with non persistent messages as a result of adding additional execution groups. By using non persistent messages the processing is CPU bound. The level of queue access is reduced because of the ESQL processing on the message. As a result the contention for queue access seen in Case 1 is not an issue.

The message throughput with persistent messages also increased with more execution groups but at a lower rate than for non persistent messages. The maximum rate achievable is determined by the rate at which the MQSeries queue manager log is able to operate at. This is in turn determined by the speed of I/O for the device on which the log is located.

The cases above show that there is a very wide range of scaling from little benefit to 100%. Given the availability of sufficient resources, the following will determine the scaling ratio that can be achieved:

- The type of message being processed (persistent vs non persistent). Persistent messages have a lower message processing rate limit than non persistent messages. Persistent messages are I/O bound whilst non persistent are CPU bound. Once the I/O limit is reached throughput cannot be increased.

- The level of contention for resources such as MQSeries queues or rows in a database. If there is a common input and common output queue for message processing there is a greater potential for conflict than if there are multiple input and output queues. Similarly if multiple message flows are attempting to update the same row in a database there will be conflict for access to that row. Bypassing these issues largely comes down to application architecture.

- The amount of additional CPU processing in the message flow, such as ESQL in a compute node.

- The message throughput rate that is currently being achieved with a single execution group. If a single execution group is able to fully utilize the I/O capacity of the MQSeries queue manager log or database manager log for example, then it is unlikely that additional instances will allow any more throughput to be obtained because the message flow is already constrained by the rate at which log I/O can take place. Similarly, if a message flow is simply copying messages from one queue to another at a very high rate, it is unlikely that the addition of another copy of this message flow would achieve significantly more throughput as there would be contention for queue access.

In planning your configuration, it is important to establish where the constraints in the configuration will be, since these will affect the ability to scale throughput. Determine whether the system will be CPU or I/O bound. If it is CPU bound, ensure that the fastest processors available are used. If the system is I/O bound, allocate as many physical disks as possible to split log and data. Also use the fastest devices available (disks with fast write nonvolatile cache or solid state disk). If it is not possible to remove the constraints, consider using multiple brokers on the same machine, or multiple brokers over multiple machines, in order to achieve the required message rate.

3.3 Memory

In estimating memory requirements for MQSI V2 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 MQSI V2 implementation.
- The Broker. There may be multiple brokers and within these, multiple execution groups and therfore multiple operating system processes.
- MQSeries Queue Manager. There will be one queue manager per broker. One for the Configuration Manager and one for the User Name Server.
- Relational Database. A database is required to hold information on behalf of the Configuration Manager and each broker. Additional relational databases may be in use which hold business data. In the sizings given below a DB2 database was used.

For the Control Center an 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 256MB of memory available in a development environment, but you are recommended to have more available.

The amount of memory required by a broker will depend on the way in which it is configured. A guideline is to allow 300MB for MQSI V2 and its dependent software(broker related components only, no Configuration Manager or Control Center), with an additional 25 MB per running execution group. This recommendation is based on an MQSeries queue manager configuration consisting of 10 SVRCONN channels and a small number of queue definitions (less than 25). If the number of MQSeries resources (channels, queues etc.) to be configured in a system is different 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 MQSI V2 configuration should be based when running in production. These are only recommendations and are not a substitute for a formal planning and sizing exercise in which requirements are accurately determined.

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

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

By following this organization the brokers can run in a shielded environment as they process messages. It is important that this processing proceeds without competition for resources from other processes in order to ensure the smooth flow of messages through the enterprise.

A recommended machine specification for the Control Center is a fast uni-processor (Pentium III 500 Mhz processor) with 256MB memory.

A recommended machine specification for the Configuration Manager is a fast uni-processor (Pentium III 500Mhz 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 which are to be used, and the level of transaction control that is used. A recommended minimum specification would be a 2 way processor with 512MB memory. The specification may need to be upgraded if message rates are high or there are many execution groups. In such cases more detailed planning would be required. Prototyping and benchmarking should be considered in order to accurately determine resource requirements. The results produced will then be specific, and tailored, to, the individual configuration being built.

If persistent messages are to be used, the use of solid state disks or disks with a nonvolatile fast write cache is recommended for the device on which the MQSeries queue log manager is located. Where the message rate is less than 50 msgs/second per second fast I/O will improve message response time only. Where the rate is greater than 50 msgs/second then there will be an improvement in message throughput.
A separate disk is also recommended for the MQSeries queue manager queue data. This disk need not have a fast write capability.

If business data is accessed from a relational database the database log 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 MQSI V2 configuration.

4.1 Understand Recovery Requirements

In designing message flows within MQSI V2 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 will be undertaken than is actually required. 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 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. You must also establish whether data updated in an external resource manager is to be committed within a global unit of work.

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. In order for this mode to function correctly the MQSeries queue manager associated with the broker must have an XA connection to each of the external resource managers.

Think carefully about 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 MQSI V2.

In order 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 the 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 to, improve the efficiency where possible, consider the following points:

- The MQSeries queue manager log and queue data should be configured on individual, dedicated, disks.

- Use the fastest disks available for the MQSeries log. Solid state disks give the best possible performance but are expensive. The next alternative is to use disks that have a non volatile fast write cache, such as SSA .

- Run with parallel applications rather than a single application. There is some benefit to be obtained from tailcoating on log I/O that occurs when there is more than one application running with the queue manager.

4.3 Configuration Considerations

Consider the following points when building an MQSI V2 configuration:

- You are recommended to use a separate database instance for the Configuration Manager and each broker.
- You are recommended to not use the database instances for the Configuration Manager or broker to hold business data.
- You are recommended to ensure that the database instance for the Configuration Manager is local to the machine on which the Configuration Manager is installed.
- You are recommended to ensure that the database instance for the broker is local to the machine on which the broker runs.
- You are 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. For example 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 you required.
- When creating and deploying large message flows 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.

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 achieved in MQSI V2 by running multiple execution groups or using additional instances as this will add an additional process or thread respectively which provides the potential to increase CPU utilization by using another processor. Where a message flow is I/O bound, because of database I/O in a database node for example, consider using an additional instance of the message flow. 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.
- In maximizing multiprocessor exploitation remember that there are processes associated with the MQSeries queue manager and any databases which may be used from within message flows. For example, the MQSeries queue manager listener process is capable of fully utilizing a processor dependent on the message rate. This was the case with the MQSI V2 performance measurements.
- When creating multiple execution groups, be careful about the number of times messages are written back to an MQSeries queue as a means of communicating between execution groups since this is an expensive operation especially with persistent messages. It may be better to form one larger single execution group.
- Run the broker as a trusted application. This is achieved by using the ‘-t’ flag on the mqsicreatebroker command, and by ensuring the environment variable MQ_CONNECT_TYPE is set to FASTPATH and that it is present in the environment in which the broker is started.
- Compute nodes are expensive in processing costs because they build a representation of the input message. Therfore you are recommended to minimize the number of compute nodes. You

might consider the use, of 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 2.11 *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.

- Monitor the number of application handles in any databases containing business data (non MQSI V2 databases). In DB2 for example this is the MAX_APPLS parameter. Each database node will obtain an application handle. The node will retain this handle until the execution group terminates. If there are multiple message flows each with multiple database nodes then the DB2 default value of 40 can be quickly exceeded.

- 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.

5.0 GLOSSARY

- **Get**

  Get time in seconds as reported by the Requester application. (This is an MQGET with a wait for the reply message)

- **MsgSize**

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

- **Mbits/Sec**

  Number of bits per second being transferred between the client and server.

- **Msgs/sec**

  Messages per second measured by the Server. Each request and each reply is a message.

- **Persistence**

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

- **Put**

  Put time in seconds as reported by the Requester application.

- **Response**

  Response time in seconds as reported by the Requester application. (This is the round trip time from putting the request message on a queue to receiving the reply message. It should be the total of get and put)**.**

- **MQRFH**

  MQSeries Rules and Formatting Header(MQRFH) 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.

- **SSA**

  Serial Storage Architechture (SSA). A serial storage system available from IBM.

6.0 APPENDIX A - MEASUREMENT HARDWARE AND SOFTWARE

All throughput measurements where taken on a single server machine driven by MQSeries clients running on a separate client machine connected by a 100Mb Ethernet link.

MQSeries Clients communicated with the MQSeries queue manager on the server machine using an MQI channel.

Server Machine

The server machine hardware consisted of

- A Solaris E450 with 4 * 400Mhz processors.
- Eight 4.2 GB SCSI hard drives and three 9.0 GB SCSI hard drives.
- 1GB RAM.
- 100Mb Ethernet Card.

The server machine software consisted of:

- Solaris 2.7.
- MQSeries V5.1 and CSD 4.
- MQSeries Integrator for Sun Solaris V2.0.1
- DB2 for Solaris V6.1.

Client Machine

The client machine hardware consisted of

- An IBM Netfinity 5500 with 4 * 550Mhz Pentium III XEON processors.
- Six 8.0 GB SCSI hard drives formatted to use NTFS.
- 1GB RAM.
- 100Mb Ethernet Card.

The client machine software consisted of:

- Microsoft Windows NT V4.0 with Service Pack 5.
- MQSeries V5.1.

Network Configuration

The client and server machines were connected on a full duplex 100Mb Ethernet LAN with a single hub.

Although the nominal data transfer rate is 200 Mb/second the practical limit proved to be around 130 Mb/second. This is only of relevance for the measurements with large message sizes.

7.0 APPENDIX B - MEASUREMENT DATA

This section contains the detailed results of the MQSI V2 performance measurements described in **Section 2.0 - Broker Throughput Measurements**. For an explanation of column headings see **Section 5.0 - Glossary**. The CPU utilization reported for the server machine, under the heading “rem CPU” in each table, is the system CPU utilization on the server machine. This includes **all** processes on the server machine. The figure therefore reflects the cost of the MQSeries queue manager processes, the MQ Listener, DB2 where appropriate etc. in addition to the CPU used by the MQSI V2 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

Persistence	Message Size	Msgs/ Sec	Response	Get	Put	CPU Usage (%)	Pages On Free List	Page In per Second	Page Out per Second	Mbits/ Sec
no	1024	1443	0.042	0.04	0.001	43	369752	0	0	11.3
no	4096	1322	0.052	0.05	0.002	46	369648	0	0	41.3
no	16384	977	0.078	0.073	0.004	57	368040	0	0	122.1
no	65536	230	0.187	0.085	0.102	32	363136	0	0	115
yes	1024	73.7	0.825	0.805	0.02	15	363136	0	0	0.6

7.2 Compute Node Throughput Results

7.2.1 Simple Compute Node

This measurement used a message type of XML.

Persistence	Message Size	Msgs/ Sec	Response	Get	Put	CPU Usage (%)	Pages On Free List	Page In per Second	Page Out per Second	Mbits/ Sec
no	1024	325	0.184	0.183	0.001	30	362488	0	0	2.5
no	4096	212	0.279	0.277	0.001	29	362368	0	0	6.6
no	16384	91.5	0.656	0.653	0.003	28	360648	0	0	11.4
no	65536	27.4	2.193	2.183	0.01	27	355320	0	0	13.7
yes	1024	87.4	0.67	0.656	0.014	22	355320	0	0	0.7

7.2.2 Complex Compute Node

This measurement used a message type of XML.

Persistence	Message Size	Msgs/ Sec	Response	Get	Put	CPU Usage (%)	Pages On Free List	Page In per Second	Page Out per Second	Mbits/ Sec
no	4096	43	1.387	1.386	0.002	25	359136	0	0	1.3
no	16384	32.6	1.84	1.837	0.003	26	357592	0	0	4.1
no	65536	16.4	3.61	3.601	0.009	26	352000	0	0	8.2
yes	4096	30.2	1.985	1.974	0.01	20	352000	0	0	0.9

7.2.3 Multiple Complex Compute Nodes

This measurement used a message type of XML.

Persistence	Message Size	Msgs/ Sec	Response	Get	Put	CPU Usage (%)	Pages On Free List	Page In per Second	Page Out per Second	Mbits/ Sec
no	4096	16	3.709	3.708	0.001	25	357288	0	0	0.5
no	16384	14.3	4.182	4.179	0.003	25	355624	0	0	1.8
no	65536	9.9	6.033	6.024	0.009	26	349776	0	0	5
yes	4096	14	4.26	4.25	0.01	23	349776	0	0	0.4

7.2.4 Very Complex Compute Node

This measurement used a message type of XML.

Persistence	Message Size	Msgs/ Sec	Response	Get	Put	CPU Usage (%)	Pages On Free List	Page In per Second	Page Out per Second	Mbits/ Sec
no	4096	24.2	2.503	2.502	0.001	26	357048	0	0	0.8
no	16384	20.6	2.933	2.93	0.003	26	355536	0	0	2.6
no	65536	12.8	4.667	4.655	0.011	26	350000	0	0	6.4
yes	4096	19.6	3.044	3.034	0.01	22	350000	0	0	0.6

7.3 Database Node Throughput Results

This measurement used XML messages.

Persistence	Message Size	Msgs/ Sec	Response	Get	Put	CPU Usage (%)	Pages On Free List	Page In per Second	Page Out per Second	Mbits/ Sec
no	1024	85.6	0.782	0.782	0.001	11	328776	0	0	0.7
no	4096	83.7	0.607	0.606	0.001	12	327264	0	0	2.6
no	16384	67.3	0.852	0.849	0.003	14	323792	75	0	8.4
no	65536	43	1.374	1.364	0.01	21	316664	23	0	21.5
yes	1024	55.5	1.093	1.082	0.011	11	316272	0	0	0.4

7.4 Filter Node Throughput Results

This measurement used XML messages.

Persistence	Message Size	Msgs/ Sec	Response	Get	Put	CPU Usage (%)	Pages On Free List	Page In per Second	Page Out per Second	Mbits/ Sec
no	1024	786	0.076	0.075	0.001	29	363080	0	0	6.1
no	4096	526	0.12	0.118	0.002	33	362832	0	0	16.4
no	16384	235	0.252	0.249	0.003	31	361216	0	0	29.4
no	65536	57.4	1.018	0.999	0.019	23	356056	0	0	28.7
yes	1024	73.2	0.822	0.801	0.02	16	356056	0	0	0.6

7.5 Publication Node Throughput Results

This measurement used MQRFH2 messages.

Persistence	Message Size	Msgs/ Sec	Response	Get	Put	CPU Usage (%)	Pages On Free List	Page In per Second	Page Out per Second	Mbits/ Sec
no	1024	526	0.118	0.117	0.001	30	366304	0	0	4.1
no	4096	499	0.121	0.12	0.001	34	366224	0	0	15.6
no	16384	442	0.132	0.128	0.004	37	364656	0	0	55.3
no	65536	109	0.533	0.388	0.134	17	360080	0	0	54.5
yes	1024	74.3	0.775	0.756	0.018	17	360080	0	0	0.6

7.6 Converting Messages Between Formats

Conversion of Generic XML to Generic XML.

This measurement used MQRFH2 messages.

Persistence	Message Size	Msgs/ Sec	Response	Get	Put	CPU Usage (%)	Pages On Free List	Page In per Second	Page Out per Second	Mbits/ Sec
no	4096	96.9	0.619	0.618	0.001	27	359000	0	0	3
no	16384	56.3	1.054	1.051	0.003	26	357424	0	0	7
no	65536	21.1	2.802	2.791	0.01	27	351968	0	0	10.6
yes	4096	55.3	1.077	1.066	0.01	19	351976	0	0	1.7

Conversion of Generic XML to CWF.

This measurement used MQRFH2 messages.

Persistence	Message Size	Msgs/ Sec	Response	Get	Put	CPU Usage (%)	Pages On Free List	Page In per Second	Page Out per Second	Mbits/ Sec
no	4096	59.4	1.017	1.016	0.001	27	357008	0	0	1.9
no	16384	40.5	1.465	1.462	0.003	27	355288	0	0	5.1
no	65536	18.2	3.283	3.27	0.014	27	349608	0	0	9.1
yes	4096	39.1	1.591	1.581	0.01	20	349560	0	0	1.2

Conversion of Generic XML to MRM XML.

This measurement used MQRFH2 messages.

Persistence	Message Size	Msgs/ Sec	Response	Get	Put	CPU Usage (%)	Pages On Free List	Page In per Second	Page Out per Second	Mbits/ Sec
no	4096	59.6	1.005	1.004	0.002	27	358120	0	0	1.9
no	16384	41.4	1.448	1.445	0.003	27	356416	0	0	5.2
no	65536	18.3	3.306	3.294	0.011	27	350672	0	0	9.2
yes	4096	39.5	1.536	1.526	0.01	20	350680	0	0	1.2

Conversion of CWF to Generic XML.

This measurement used MQRFH2 messages.

Persistence	Message Size	Msgs/ Sec	Response	Get	Put	CPU Usage (%)	Pages On Free List	Page In per Second	Page Out per Second	Mbits/ Sec
no	4096	46.8	1.274	1.273	0.001	26	358840	0	0	1.5
no	16384	46.7	1.278	1.277	0.001	26	358672	0	0	5.8
no	65536	46.7	1.284	1.283	0.001	26	358552	0	0	23.4
yes	4096	36.7	1.628	1.619	0.009	22	358344	0	0	1.1

Conversion of CWF to CWF.

This measurement used MQRFH2 messages.

Persistence	Message Size	Msgs/ Sec	Response	Get	Put	CPU Usage (%)	Pages On Free List	Page In per Second	Page Out per Second	Mbits/ Sec
no	4096	31	1.936	1.935	0.001	26	358768	0	0	1
no	16384	31.3	1.936	1.935	0.001	26	358632	0	0	3.9
no	65536	30.9	1.933	1.933	0.001	26	358664	0	0	15.5
yes	4096	26.8	2.219	2.208	0.01	24	358480	0	0	0.8

Conversion of CWF to MRM XML.

This measurement used MQRFH2 messages.

Persistence	Message Size	Msgs/ Sec	Response	Get	Put	CPU Usage (%)	Pages On Free List	Page In per Second	Page Out per Second	Mbits/ Sec
no	4096	31	1.953	1.952	0.001	26	357096	0	0	1
no	16384	30.8	1.949	1.949	0.001	26	356960	0	0	3.9
no	65536	30.8	1.938	1.937	0	26	356864	0	0	15.4
yes	4096	26.8	2.236	2.228	0.009	24	356680	0	0	0.8

Conversion of MRM XML to Generic XML.

This measurement used MQRFH2 messages.

Persistence	Message Size	Msgs/ Sec	Response	Get	Put	CPU Usage (%)	Pages On Free List	Page In per Second	Page Out per Second	Mbits/ Sec
no	4096	54.6	1.099	1.098	0.001	27	358200	0	0	1.7
no	16384	39.2	1.555	1.552	0.002	27	356560	0	0	4.9
no	65536	18.1	3.35	3.339	0.011	26	350760	0	0	9.1
yes	4096	37.4	1.594	1.584	0.009	20	350768	0	0	1.2

Conversion of MRM XML to CWF.

This measurement used MQRFH2 messages.

Persistence	Message Size	Msgs/ Sec	Response	Get	Put	CPU Usage (%)	Pages On Free List	Page In per Second	Page Out per Second	Mbits/ Sec
no	4096	40.8	1.471	1.47	0.001	26	358336	0	0	1.3
no	16384	31.2	1.937	1.934	0.003	27	356640	0	0	3.9
no	65536	15.9	3.775	3.766	0.01	27	350792	0	0	8
yes	4096	30.5	1.971	1.962	0.009	21	350800	0	0	1

Conversion of MRM XML to MRM XML.

This measurement used MQRFH2 messages.

Persistence	Message Size	Msgs/ Sec	Response	Get	Put	CPU Usage (%)	Pages On Free List	Page In per Second	Page Out per Second	Mbits/ Sec
no	4096	41.2	1.476	1.475	0.001	27	361432	0	0	1.3
no	16384	31.3	1.934	1.93	0.004	27	359736	0	0	3.9
no	65536	15.8	3.768	3.757	0.011	27	353912	0	0	7.9
yes	4096	30.4	1.97	1.96	0.01	22	353920	0	0	1

7.7 NEONRules and NEONFormatter Within MQSI V2

NEONFormatter in MQSI V1 With 50 Fields

This measurement used MQRFH messages.

Persistence	Message Size	Msgs/ Sec	Response	Get	Put	CPU Usage (%)	Pages On Free List	Page In per Second	Page Out per Second	Mbits/ Sec
no	1024	975	0.06	0.059	0.002	71	374904	0	0	7.6
no	4096	813	0.071	0.069	0.002	57	374760	0	0	25.4
no	16384	432	0.14	0.136	0.004	37	375192	0	0	54
no	65536	98.8	0.597	0.546	0.049	18	370272	0	0	49.4
yes	1024	81.6	0.746	0.728	0.018	17	370280	0	0	0.6

NEONFormatter in MQSI V2 With 50 Fields

This measurement used MQRFH messages.

Persistence	Message Size	Msgs/ Sec	Response	Get	Put	CPU Usage (%)	Pages On Free List	Page In per Second	Page Out per Second	Mbits/ Sec
no	1024	757	0.071	0.07	0.001	71	300760	0	0	5.9
no	4096	704	0.087	0.085	0.002	69	300688	0	0	22
no	16384	521	0.111	0.107	0.004	63	299056	0	0	65.1
no	65536	105	0.542	0.4	0.146	23	293832	0	0	52.5
yes	1024	98.7	0.611	0.594	0.018	34	293800	0	0	0.8

NEONFormatter in MQSI V2 With 100 Rules Defined, One Rule Executed

This measurement used MQRFH messages.

Persistence	Message Size	Msgs/ Sec	Response	Get	Put	CPU Usage (%)	Pages On Free List	Page In per Second	Page Out per Second	Mbits/ Sec
no	1024	759	0.067	0.065	0.001	72	215112	0	0	5.9
no	4096	720	0.11	0.109	0.001	73	215040	0	0	22.5
no	16384	518	0.108	0.103	0.005	65	213368	0	0	64.8
no	65536	99.2	0.607	0.447	0.184	24	208216	0	0	49.6
yes	1024	100	0.594	0.578	0.016	34	208160	0	0	0.8

7.8 The Effect of Increasing The Number Of Execution Groups

One Execution Group Running a Complex Compute Message Flow

Persistence	Message Size	Msgs/ Sec	Response	Get	Put	CPU Usage (%)	Pages On Free List	Page In per Second	Page Out per Second	Mbits/ Sec
no	4096	24.2	2.503	2.502	0.001	26	357048	0	0	0.8
no	16384	20.6	2.933	2.93	0.003	26	355536	0	0	2.6
no	65536	12.8	4.667	4.655	0.011	26	350000	0	0	6.4
yes	4096	19.6	3.044	3.034	0.01	22	350000	0	0	0.6

Two Execution Groups Running a Complex Compute Message Flow

Persistence	Message Size	Msgs/ Sec	Response	Get	Put	CPU Usage (%)	Pages On Free List	Page In per Second	Page Out per Second	Mbits/ Sec
no	4096	48.3	1.248	1.246	0.002	51	344568	0	0	1.5
no	16384	40.4	1.474	1.471	0.003	51	342888	0	0	5.1
no	65536	25.1	2.386	2.375	0.01	52	336608	0	0	12.6
yes	4096	37.8	1.593	1.582	0.013	43	336616	0	0	1.2

Four Execution Groups Running a Complex Compute Message Flow

Persistence	Message Size	Msgs/ Sec	Response	Get	Put	CPU Usage (%)	Pages On Free List	Page In per Second	Page Out per Second	Mbits/ Sec
no	4096	93	0.679	0.678	0.001	99	325120	0	0	2.9
no	16384	77.6	0.787	0.783	0.004	98	323056	0	0	9.7
no	65536	46.8	1.26	1.25	0.01	99	315432	0	0	23.4
yes	4096	45.4	1.363	1.332	0.03	61	315440	0	0	1.4

7.9 The Effect of Making a Message Flow Transactional

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

Persistence	Message Size	Msgs/ Sec	Response	Get	Put	CPU Usage (%)	Pages On Free List	Page In per Second	Page Out per Second	Mbits/ Sec
no	1024	1443	0.042	0.04	0.001	43	369752	0	0	11.3
no	4096	1322	0.052	0.05	0.002	46	369648	0	0	41.3
no	16384	977	0.078	0.073	0.004	57	368040	0	0	122.1
no	65536	230	0.187	0.085	0.102	32	363136	0	0	115
yes	1024	73.7	0.825	0.805	0.02	15	363136	0	0	0.6

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

Persistence	Message Size	Msgs/ Sec	Response	Get	Put	CPU Usage (%)	Pages On Free List	Page In per Second	Page Out per Second	Mbits/ Sec
no	1024	1150	0.053	0.049	0.001	42	368664	0	0	9
no	4096	1110	0.055	0.053	0.002	46	368560	0	0	34.7
no	16384	838	0.067	0.062	0.006	49	366920	0	0	104.8
no	65536	182	0.459	0.295	0.169	25	362088	0	0	91
yes	1024	74.6	0.82	0.799	0.022	15	362088	0	0	0.6

7.10 The Effect of using coordinatedTransaction=yes

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

Persistence	Message Size	Msgs/ Sec	Response	Get	Put	CPU Usage (%)	Pages On Free List	Page In per Second	Page Out per Second	Mbits/ Sec
no	1024	314	0.185	0.183	0.001	21	321304	0	0	2.5
no	4096	174	0.331	0.329	0.002	14	321176	0	0	5.4
no	16384	156	0.38	0.376	0.003	25	319608	0	0	19.5
no	65536	57.2	1.025	1.016	0.009	26	314264	0	0	28.6
yes	1024	73.3	0.829	0.815	0.014	17	313016	0	0	0.6

coordinatedTransaction=yes, XA connection, Using XML Messages

Persistence	Message Size	Msgs/ Sec	Response	Get	Put	CPU Usage (%)	Pages On Free List	Page In per Second	Page Out per Second	Mbits/ Sec
no	1024	85.3	0.641	0.64	0.001	9	328640	0	0	0.7
no	4096	85.6	0.67	0.668	0.002	11	327128	0	0	2.7
no	16384	68.9	0.927	0.922	0.004	14	325520	0	0	8.6
no	65536	42.4	1.325	1.314	0.009	20	320176	0	0	21.2
yes	1024	55.2	1.101	1.09	0.011	10	320176	0	0	0.4

7.11 The Effect of Increasing the Number of Subscribers

One Subscriber Receiving MQRFH2 Type Published Message

Persistence	Message Size	Msgs/ Sec	Response	Get	Put	CPU Usage (%)	Pages On Free List	Page In per Second	Page Out per Second	Mbits/ Sec
no	1024	526	0.118	0.117	0.001	30	366304	0	0	4.1
no	4096	499	0.121	0.12	0.001	34	366224	0	0	15.6
no	16384	442	0.132	0.128	0.004	37	364656	0	0	55.3
no	65536	109	0.533	0.388	0.134	17	360080	0	0	54.5
yes	1024	74.3	0.775	0.756	0.018	17	360080	0	0	0.6

10 Subscribers Receiving MQRFH2 Type Published Messages

Persistence	Message Size	Msgs/ Sec	Response	Get	Put	CPU Usage (%)	Pages On Free List	Page In per Second	Page Out per Second	Mbits/ Sec
no	1024	189	0.313	0.312	0.002	32	365632	0	0	1.5
no	4096	172	0.346	0.345	0.001	34	365528	0	0	5.4
no	16384	161	0.361	0.358	0.003	36	363288	0	0	20.1
no	65536	96.7	0.605	0.59	0.017	34	358592	0	0	48.4
yes	1024	20.5	2.982	2.962	0.02	14	358592	0	0	0.2

30 Subscribers Receiving MQRFH2 Type Published Messages

Persistence	Message Size	Msgs/ Sec	Response	Get	Put	CPU Usage (%)	Pages On Free List	Page In per Second	Page Out per Second	Mbits/ Sec
no	1024	78.5	0.761	0.76	0.001	33	363512	0	0	0.6
no	4096	72.1	0.836	0.835	0.001	35	363160	0	0	2.3
no	16384	67.8	0.89	0.887	0.003	36	361456	0	0	8.5
no	65536	49.1	1.195	1.184	0.011	38	356536	0	0	24.6
yes	1024	7.9	7.63	7.614	0.015	10	356520	0	0	0.1

50 Subscribers Receiving MQRFH2 Type Published Messages

Persistence	Message Size	Msgs/ Sec	Response	Get	Put	CPU Usage (%)	Pages On Free List	Page In per Second	Page Out per Second	Mbits/ Sec
no	1024	49.5	1.218	1.217	0.001	32	364592	0	0	0.4
no	4096	45.3	1.333	1.332	0.001	35	364544	0	0	1.4
no	16384	42.7	1.425	1.422	0.004	36	362552	0	0	5.3
no	65536	31	1.884	1.873	0.011	37	357664	0	0	15.5
yes	1024	4.8	12.549	12.531	0.018	9	357664	0	0	0

70 Subscribers Receiving MQRFH2 Type Published Messages

Persistence	Message Size	Msgs/ Sec	Response	Get	Put	CPU Usage (%)	Pages On Free List	Page In per Second	Page Out per Second	Mbits/ Sec
no	1024	35.9	1.669	1.667	0.001	32	364128	0	0	0.3
no	4096	33	1.808	1.807	0.002	34	364168	0	0	1
no	16384	31.3	1.918	1.916	0.003	35	362344	0	0	3.9
no	65536	23.6	2.537	2.526	0.011	36	357280	0	0	11.8
yes	1024	3.4	17.39	17.376	0.016	9	357264	0	0	0

100 Subscribers Receiving MQRFH2 Type Published Messages

Persistence	Message Size	Msgs/ Sec	Response	Get	Put	CPU Usage (%)	Pages On Free List	Page In per Second	Page Out per Second	Mbits/ Sec
no	1024	25.3	2.373	2.372	0.001	32	363136	0	0	0.2
no	4096	23	2.602	2.601	0.002	35	362768	0	0	0.7
no	16384	21.5	2.733	2.73	0.003	36	361088	0	0	2.7
no	65536	16.5	3.618	3.604	0.014	37	356384	0	0	8.3
yes	1024	2.4	25.11	25.092	0.018	9	356376	0	0	0

1000 Subscribers Receiving MQRFH2 Type Published Messages

Persistence	Message Size	Msgs/ Sec	Response	Get	Put	CPU Usage (%)	Pages On Free List	Page In per Second	Page Out per Second	Mbits/ Sec
no	1024	2.3	25.896	25.895	0	31	311464	0	0	0
no	4096	2.1	28.127	28.124	0.003	33	310768	0	0	0.1
no	16384	2	29.606	29.603	0.003	34	309304	0	0	0.3
no	65536	1.5	39.531	39.518	0.012	37	304440	0	0	0.8
yes	1024	0.2	98.105	98.098	0.004	13	303352	0	0	0

7.12 Content vs Topic Based PubSub

Topic Based Routing, Single Subscriber Receiving MQRFH2 Type Messages

Persistence	Message Size	Msgs/ Sec	Response	Get	Put	CPU Usage (%)	Pages On Free List	Page In per Second	Page Out per Second	Mbits/ Sec
no	1024	508	0.118	0.116	0.001	31	282104	0	0	4
no	4096	491	0.117	0.117	0.001	33	282016	0	0	15.3
no	16384	427	0.132	0.129	0.004	36	280464	0	0	53.4
no	65536	105	0.55	0.454	0.105	17	275848	0	0	52.5
yes	1024	74.6	0.826	0.806	0.021	17	275752	0	0	0.6

Content Based Routing, Single Subscriber Receiving MQRFH2 Type Messages

Persistence	Message Size	Msgs/ Sec	Response	Get	Put	CPU Usage (%)	Pages On Free List	Page In per Second	Page Out per Second	Mbits/ Sec
no	1024	327	0.185	0.184	0.001	29	368600	0	0	2.6
no	4096	266	0.223	0.221	0.001	29	368472	0	0	8.3
no	16384	158	0.388	0.384	0.003	29	366792	0	0	19.8
no	65536	57.9	1.031	1.018	0.014	27	361608	0	0	29
yes	1024	87.9	0.679	0.664	0.014	22	361608	0	0	0.7

7.13 The Effect of an Increasing Number of Message Fields

One Field in an Input Message to a Compute Node Using XML Type Messages

Persistence	Message Size	Msgs/ Sec	Response	Get	Put	CPU Usage (%)	Pages On Free List	Page In per Second	Page Out per Second	Mbits/ Sec
no	1024	325	0.184	0.183	0.001	30	362488	0	0	2.5
no	4096	212	0.279	0.277	0.001	29	362368	0	0	6.6
no	16384	91.5	0.656	0.653	0.003	28	360648	0	0	11.4
no	65536	27.4	2.193	2.183	0.01	27	355320	0	0	13.7
yes	1024	87.4	0.67	0.656	0.014	22	355320	0	0	0.7

50 Fields in an Input Message to a Compute Node Using XML Type Messages

Persistence	Message Size	Msgs/ Sec	Response	Get	Put	CPU Usage (%)	Pages On Free List	Page In per Second	Page Out per Second	Mbits/ Sec
no	1024	157	0.381	0.38	0.001	27	360904	0	0	1.2
no	4096	126	0.478	0.477	0.001	27	360768	0	0	3.9
no	16384	70.9	0.853	0.849	0.003	27	358984	0	0	8.9
no	65536	25.8	2.332	2.322	0.011	26	353688	0	0	12.9
yes	1024	74.4	0.804	0.794	0.01	20	353688	0	0	0.6

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;

SET i = 1;

WHILE i &lt;= CARDINALITY(OutputRoot.XML.CSIM.TestCase.Stack.ProcessingPath.Element[]) 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 which 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