Websphere MQ Integrator for Sun Solaris V2.1 Performance Report

Version 1.1

April, 2002

Tim Dunn

Websphere MQ 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".

Third Edition, April 2002

This edition applies to Version 1.1 of *Websphere MQ Integrator for Sun Solaris - V2.1 Performance Report* and to all subsequent releases and modifications until otherwise indicated in new editions.

Notices

This report is intended to help the customer understand the performance characteristics and perform capacity planning for Websphere MQ Integrator for Sun Solaris - V2.1 at the CSD2 level of code. The information is not intended as the specification of any programming interfaces that are provided by MQSeries or Websphere MQ Integrator for Sun Solaris - V2.1.

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

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

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

Trademarks and service marks

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

- IBM
- Netfinity
- MQSeries
- Websphere MQ Integrator
- DB2

The following terms are trademarks of other companies:

- Solaris, Java Sun Corporation
- Windows NT, Visual Studio Microsoft Corporation
- NEONFormatter, NEONRules, NEONTransform, NEONRulesEvaluation New Era of Networks

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

Summary of Amendments

Date	Changes
14 December 2001	Initial Release
20 February 2002	Minor typographic corrections. No change to performance measurements or results.
2 April 2002	Updated for CSD2 level code

Contents

1.0 CONCEPTS

Websphere MQ Integrator for Sun Solaris - V2.1 (WMQI V2.1) is IBMs' message broker product, addressing the needs of business and application integration. Business integration is the coordination of all of a company's processes. Application integration is the coordination of its applications. This process of integration involves the bringing together of the data and processes within an organization to maximize the sharing of data and applications in order to cope with changing organization structure (merger, acquisition etc.) and increase the effectiveness of the organization.

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

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

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

1.1.1 The Configuration Manager

The Configuration Manager is the main component of the WMQI environment. The components and resources managed by the Configuration Manager constitute the broker domain. The Configuration Manager 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 initialization and deployment of brokers and message processing operations in response to actions initiated through the Control Center. It communicates with other components in the broker domain using MQSeries transport services.
- It checks the authority of defined user IDs to initiate those actions.

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

1.1.2 The Brokers

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

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

Within each broker it is possible to define execution groups that are responsible for running the message flows. An execution group is implemented as an operating system process. Within an execution group it is possible to define additional threads that will also perform the processing of the message flows, these are known as additional instances.

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.

Websphere MQ 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 one 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 execution group 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 WMQI messages are always in one of two broad categories:

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

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

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

Predefined Messages

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

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

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

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

Message Templates

A message template is made up of four values contained within the header information:

1. *Message Domain* which identifies the message parser that will interpret the bit-stream of the message. By default WMQI supports the values MRM, XML, BLOB, NEON and NEONMSG. 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 and NEONMSG are for New Era of Networks 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 New Era of Networks.

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.

   TAG denotes data structures in which the fields are either fixed length or separated by tags. Data structures for TAG messages are defined in the message repository

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

Self-defining messages

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

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

Self defining messages are said to use generic XML.

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

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

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

1.4 Message Parsers

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

Message parsers are provided for :

- Predefined XML. Such messages have an MQRFH or MQRFH2 header.

- The standard MQSeries headers: MQCIH, MQDLH, MQIIH, MQMD, MQMDE, MQRFH, MQRFH2, MQRMH, MQSAPH ,MQCFH, and MQWIH.

- Record-orientated C and COBOL language structures.
- Self-defining (generic XML) messages.
- Messages whose formats are defined in the NEON dictionary (These messages are defined using the New Era of Networks interface not the Control Center).

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

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

2.0 BROKER THROUGHPUT MEASUREMENTS

In order to understand the processing characteristics of WMQI V2 a number of performance measurements have been taken using multiple aspects of the product. The test cases used have been deliberately made trivial in order to be able to report the cost of using WMQI V2, 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 predominately non persistent messages as well as having a compensating program to ensure that the queue manager queue cache did not overflow to disk. If these two actions were not taken, the throughput possible with WMQI V2 would have been constrained by the necessary I/O processing of the queue manager with which the WMQI 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, what are the relative costs of the different node types and how much CPU and memory are required.

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

- MQInput and MQOutput
- Compute
- Database
- Filter
- RouteToLabel
- Publication
- NEONTransform
- NEONRulesEvaluation
- C & Java Plug-ins

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.

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

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

All measurements were conducted in the same measurement environment. This is described in Section 5.0 APPENDIX A - MEASUREMENT HARDWARE AND SOFTWARE.

The MQSeries queue manager listener process was run as an authorized MQ application in order to improve message throughput. This was achieved by ensuring that the environment variable

MQ_CONNECT_TYPE=FASTPATH was present in the environment in which the listener was started.

All measurements were driven by a multithreaded MQSeries Client program written in C. The number of threads used for each measurement was tailored to meet the processing capabilities of the flow so as not to have several idle threads during slower measurements. All threads sent 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 roundtrips 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 or another way of viewing it is as the message arrival rate on the input queue for the MQInput node. This method of reporting makes this report consistent with MQSeries performance reports.

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

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

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

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

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

In order to show optimal performance of WMQI V2 all the throughput measurements in this document used a value of automatic for the transaction parameter unless otherwise specified,.

2.1 WMQI V2.1 Performance Improvements

In our measurements we have not seen any degradation in message throughput with WMQI V2.1 compared with MQSI V2.0.2 CSD level 1. The measurements for this release of WMQI V2 are taken on the same hardware and use the same level of MQSeries that was used for MQSI V2.0.2 so comparisons are easy this time. We do not have to allow for environmental changes.

There are increases in performance for most nodes; the average measurement showed that we are getting 1.3 times the message rate over MQSI V2.0.2 CSD level 1. Most messages should therefore obtain benefit from migration to WMQI V2 since the Filter and Compute nodes are the most popular in message Flows.

Node Type	Average improvement
MQinput and MQoutput	1.2
PubSub	1.1
Filter	1.3
Database	1.0
Compute (non MRM)	1.7
Compute(MRM)	1.1
RoutetoLabel	1.6
All	1.3

Table 1: Maximum message rate improvements

2.2 MQInput/MQOutput Throughput

A message flow consisting of a single MQInput and MQOutput node represents the simplest message flow. Measuring the throughput achievable with such a message flow shows the maximum message rate that can be achieved using WMQI 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 that were obtained as a result of running the message flow with varying message sizes and persistence. There was a single instance and single execution group running the message flow.

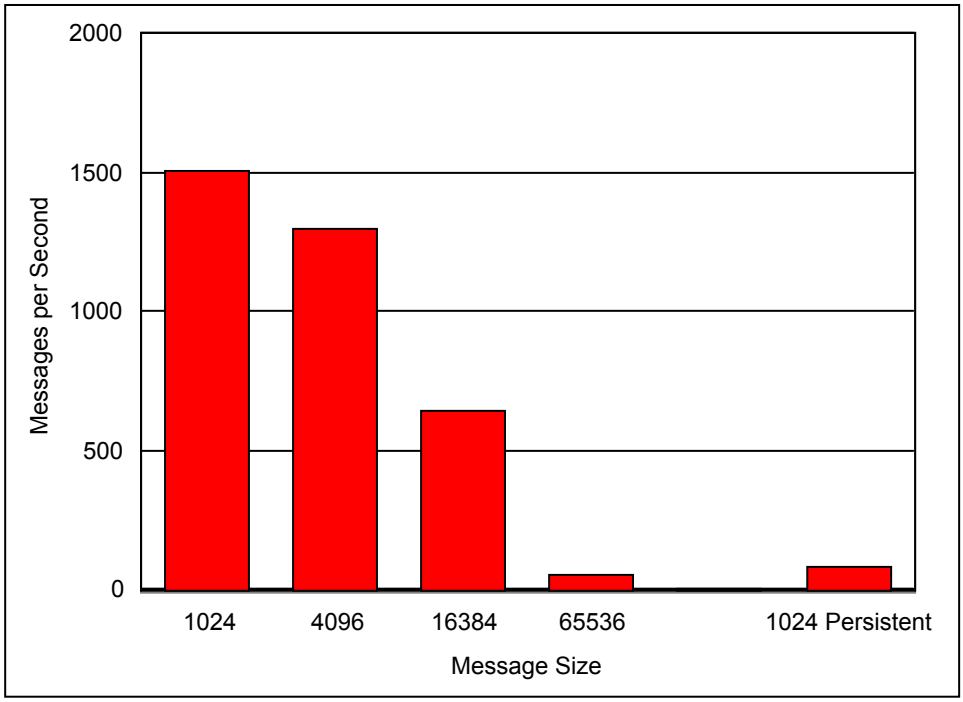

Figure 1: MQInput/MQOutput Throughput Results

With a 1K non persistent message it was possible to process approximately 1500 msgs/second. Increasing the message size had a significant effect on the maximum message throughput that could be achieved. This decrease in throughput is as a result of the additional volume of data that must be managed by the associated MQSeries queue manager and network.

As the input message was non persistent there was no transactional control. We are, therefore observing the maximum rate at which WMQI 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 approximately 80 msgs/second. This rate was attained using an SCSI disk for the MQ log and an SCSI disk for the input and output queue.

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

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

2. The MQSeries queue manager must make the message persistent, which involves a synchronous write to the MQSeries log 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 6.1 - MQInput/MQoutput Throughput Results**.

2.3 Compute Node Throughput

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

Each test case consisted of an MQInput and MQOutput node with varying numbers of compute nodes in between. The level of complexity in the compute nodes was also varied. The following cases were measured:

- A simple compute node that copied the input message to an output message. The purpose of this measurement was to show the message throughput that is achievable when copying a message and modifying a single field. A single field was modified in order to ensure that the compute node built a new output message based on the input. If no field is modified WMQI V2 optimises the process and simply repeats the input message which can give an over optimistic message rate**.** This represents the simplest form of compute node.

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

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

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

- In these measurements the input message consisted of a single field. The transaction mode on the MQInput and MQOutput nodes was set to automatic.

2.3.1 Simple Compute Node

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

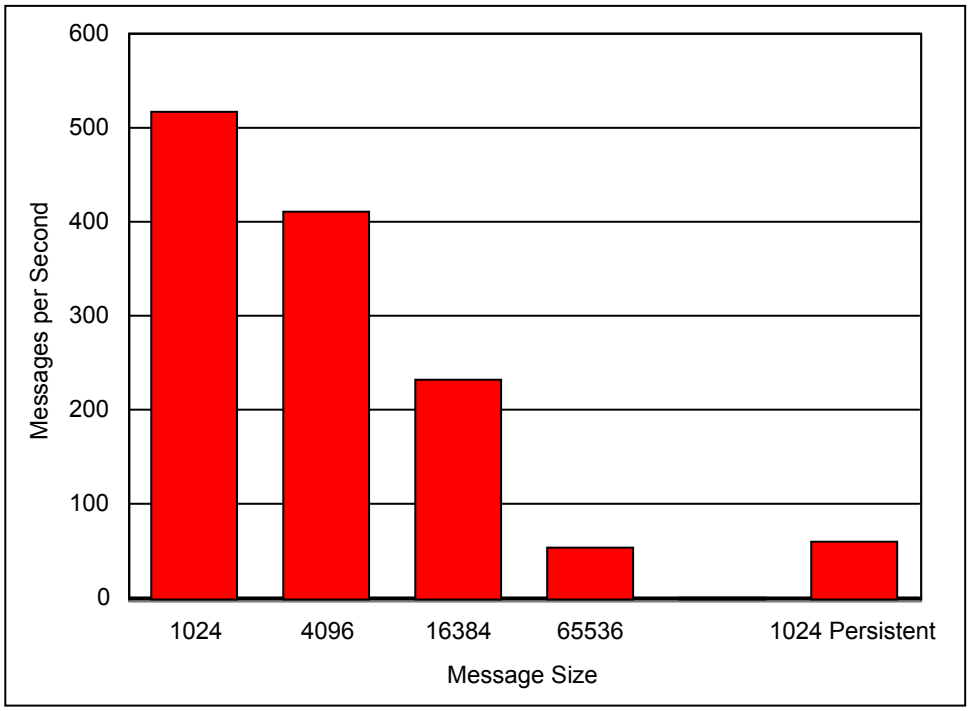

Figure 2: Simple Compute Node Throughput Results

With a 1K non persistent message it was possible to process approximately 520 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 60 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 6.2 - Compute Node Throughput Results**.

2.3.2 Complex Compute Node

Figure 3 below shows the results that were obtained as a result of running a complex message flow with varying message sizes and persistence. See Appendix C for a description of this complex flow. There was a single instance and single execution group running the message flow. Due to the message complexity, the minimum size message that would run successfully was 4k

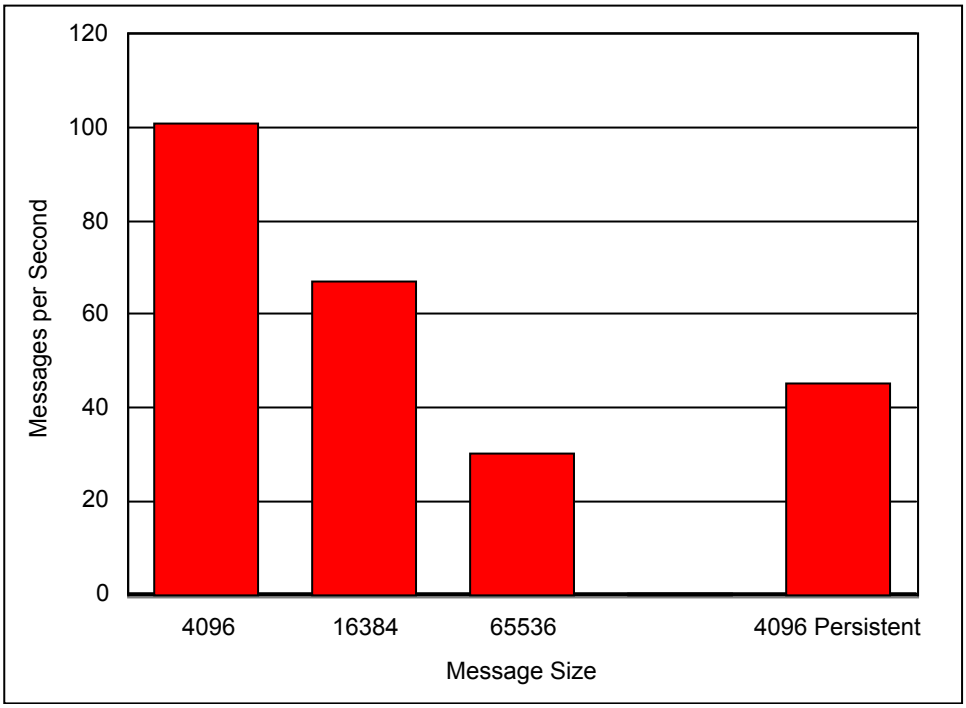

Figure 3: Complex Compute Node Throughput Results

With a 4K non persistent message it was possible to process approximately 100 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 approximately 45 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 6.2 - Compute Node Throughput Results**.

2.3.3 Multiple Complex Compute Nodes

Figure 4 below shows the results that were obtained as a result of running five of the above complex nodes daisy chained together for varying message sizes and persistence. See Appendix C for a description of this complex flow. There was a single instance and single execution group running the message flow. Due to the message complexity, the minimum size message that would run successfully was 4k.

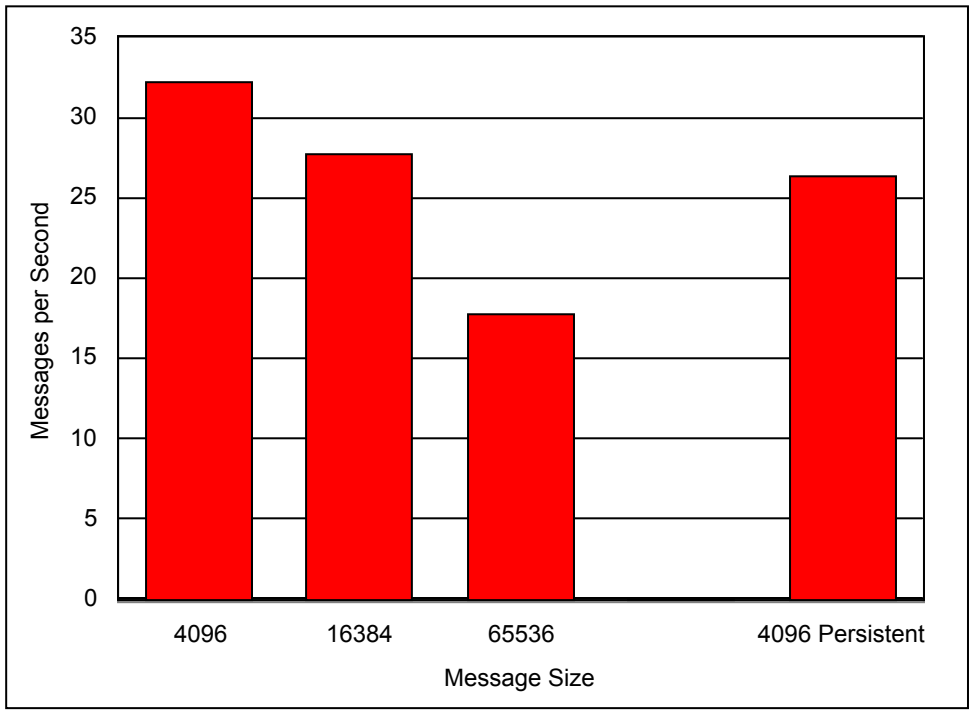

Figure 4: Multiple Complex Compute Node Throughput Results

With a 4K non persistent message it was possible to process approximately 32 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 26 msgs/second.

The detailed measurement data for the Compute Node throughput measurements is available in **Section 6.2 - Compute Node Throughput Results**.

2.3.4 Very Complex Compute Node

Figure 5 below shows the results that were obtained as a result of running a very complex message flow with varying message sizes and persistence. Appendix C has a description of a very complex flow. Briefly, a very complex flow is defined as the complex flow repeated 5 times in the same node. There was a single instance and single execution group running the message flow. Due to the message complexity, the minimum size message that would run successfully was 4k.

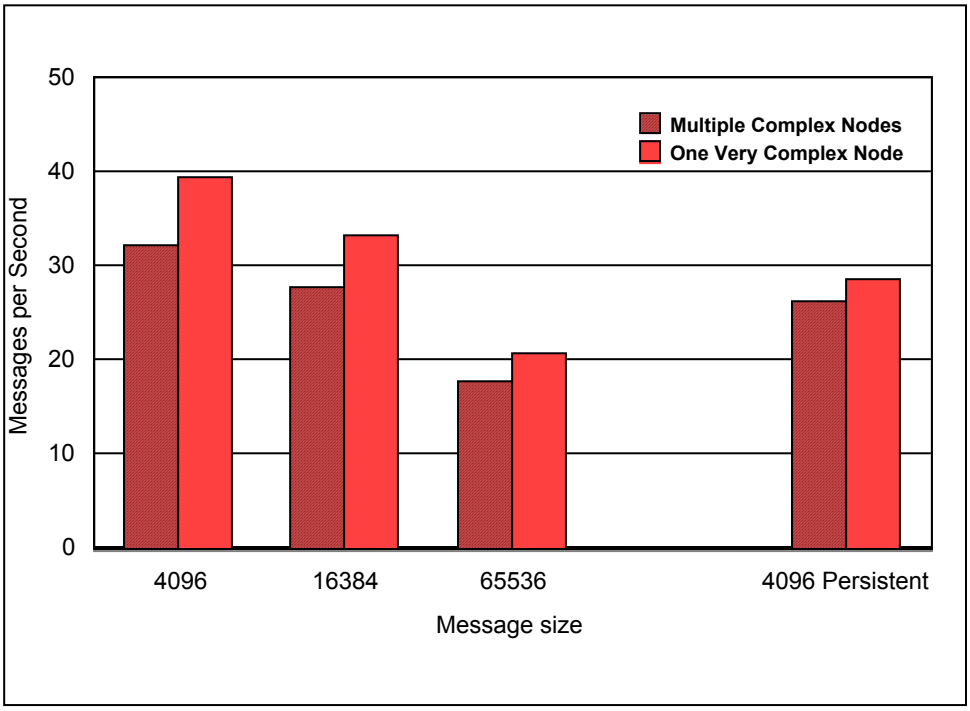

Figure 5: Very Complex Compute Node VS Multiple Complex Compute Node Throughput Results

With a 4K non persistent message it was possible to process approximately 40 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 29 msgs/second.

For comparison purposes Figure 5 also shows the message throughput rates that were achieved for the multiple complex compute node case detailed in **Section 2.3.3 - Multiple Complex Compute Nodes**.

For 4K non persistent messages there was a 1.18 times improvement in message throughput as a result of using a single compute node for the processing, rather than using 5 nodes. For performance reasons it is clearly better to have one node that does the work of several less complex nodes. This performance improvement has to be offset against the management and support of more complex nodes.

The detailed measurement data for the Very Complex Compute Node throughput measurements is available in **Section 6.2 - Compute Node Throughput Results**.

2.4 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 WMQI V2 broker must have a suitable X/Open XA interface connection with the database. The MQSeries queue manager associated with the WMQI V2 broker acts as the transaction coordinator. The database is a resource manager.

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 could result in data in 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 to automatic. The coordinatedTransaction value for the message flow was set to yes. The effect of doing this is to specify that the message flow should be a globally coordinated unit of work.

The maximum possible message throughput rates were determined for a single instance and single execution group running the message flow. **Figure 6** below shows the results that were obtained for varying message size and persistence.

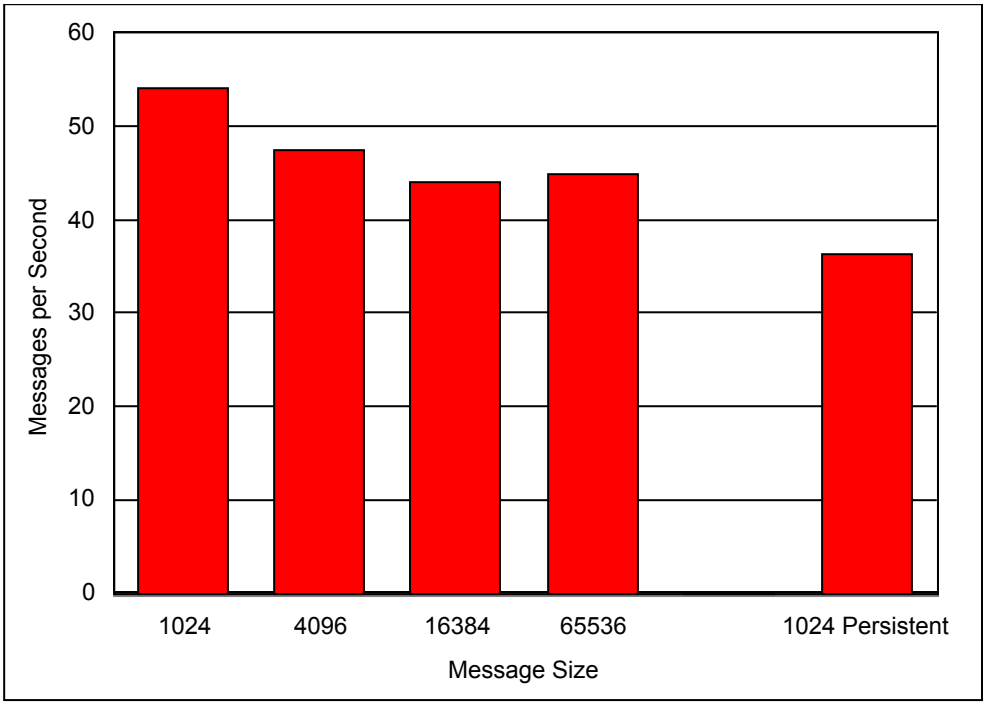

Figure 6: Database Insert/Delete Throughput Results

With 1K non persistent messages it was possible to achieve a message throughput rate of approximately 54 msgs/second. This is 54 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 36 msgs/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**.

2.5 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 MQRFH2 message with two tags specified following the header. The transaction mode on the MQInput and MQOutput nodes was set to automatic.

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

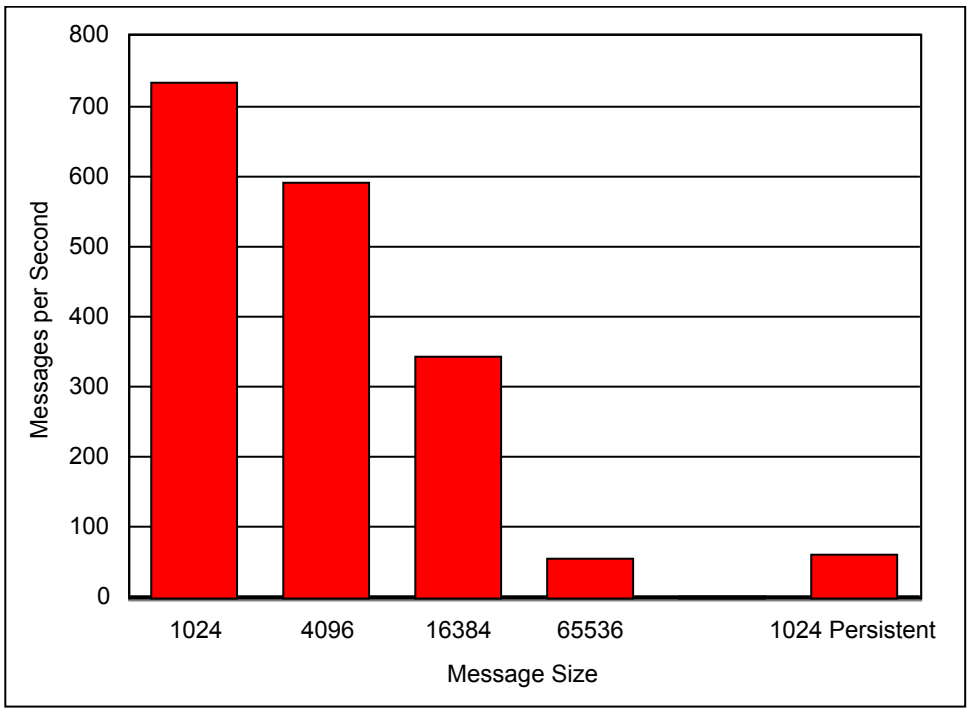

Figure 7: Filter Node Throughput Results

With 1K non-presistent messages it was possible to run approximately 730 msgs/second. 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 approximately 60 msgs/second. The reduction in throughput is as a result of using persistent messages that involves additional logging within the MQSeries manager as well as the fact that the message is processed under MQSeries transaction control.

The detailed measurement data for the Filter Node Throughput measurements is available in **Section 6.4 - Filter Node Throughput Results.**

2.6 RouteToLabel Node Throughput

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

The destination list which is used to control the routing must have been created and included in a previous compute node. Consequently a RouteToLabel node is more expensive to process than a single Filter node, but may be cheaper than many Filter nodes. . For a better understanding of how to choose, please read the recommendations in **Supportpac IP04, Designing Message Flows for Performance.**

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

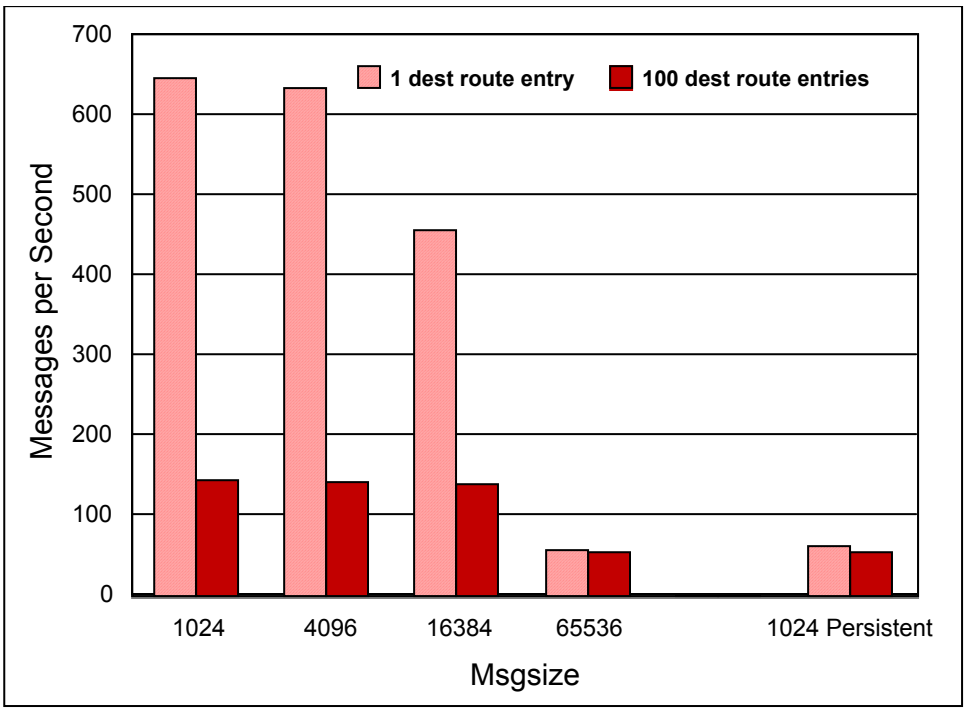

Figure 8: RouteToLabel Node Throughput Results

With 1K non-presistent messages it was possible to run approximately 650 msgs/second when we had only one destination in the list and only 140 msgs/sec when we had 100 entries in the destination list.

With 1K persistent messages it was possible to run approximately 60 msgs/second when we had only one destination in the list and only 53 msgs/sec when we had 100 entries in the destination list.

The detailed measurement data for the RouteToLabel Node Throughput measurements is available in **Section 6.5 - RouteToLabel Node Throughput Results.**

2.7 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 node was set to automatic. The measurement used topic routing with MQRFH2 format messages.

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

With this measurement we had only one subscriber.

Figure 9 below shows the results that were obtained as a result of running the message flow with varying message sizes and persistence. There was a single instance and single execution group running the message flow. The rates shown are the rate at which messages are being published.

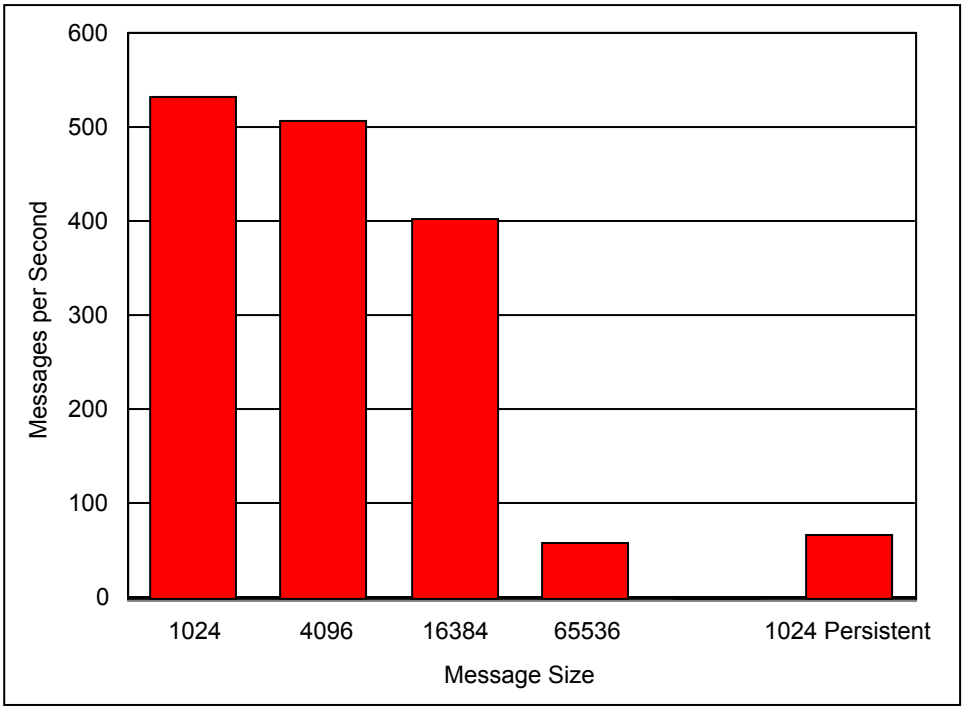

Figure 9: Publication Node Throughput Results

With 1K non-presistent messages it was possible to publish approximately 530 msgs/second. This can equally viewed as a subscription rate of 530 messages per second per subscriber

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

With 1K persistent messages the published throughput was approximately 67 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 6.6 - 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 gave 0.89 times the message throughput.

2.8 NEONRulesEvaluation Throughput

WMQI V2.1 provides the NEONRulesEvaluation node. The NEONRulesEvaluation node was new with MQSI V2.0.2 and was introduced as a result of the support for New Era of Networks Rules and Formatter Version 5.2. The original NEONRules node is still shipped for compatibility reasons.

Throughput measurements were taken in order to evaluate performance of the NEONRulesEvaluation node. An input message with 50 fields was scanned and tested against the contents of one field. When the NEON Rule was satisfied the input message was placed on an output queue. In this test the rule was always satisfied.

Figure 10 below shows the results that were obtained as a result of running with one rule defined when using the NEONRulesEvaluation node.

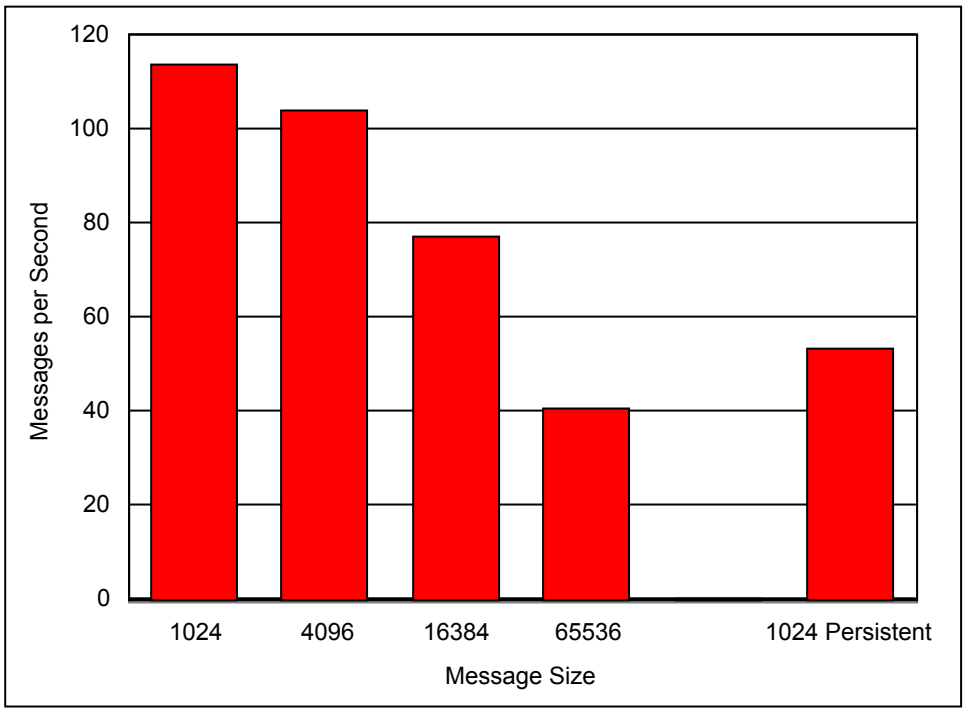

Figure 10: NEONRulesEvaluation Node Throughput Results

The detailed measurement data showing the effect of running the New Era of Networks nodes within WMQI V2 is available in **Section 6.13 - New Era of Networks Throughput Results**.

2.1 NEONTransform Throughput

WMQI V2.1 provides the NEONFormatter and NEONTransform nodes. The NEONTransform node was new with MQSI V2.0.2 and was introduced as a result of the support for New Era of Networks Rules and Formatter Version 5.2. The original NEONFormatter node is shipped for compatibility reasons.

The NEONTransform node is intended as a direct replacement for the NEONFormatter node. It is capable of producing all of the function of the NEONFormatter node. .

Throughput measurements were taken in order to evaluate performance of the NEONTransform node. An input message with 100 fields was transformed from one format to another and the resultant message was placed on an output queue. Due to limitations of the test, only one message size (4K) was used.

Figure 11 below shows the results that were obtained as a result of running the NEONTransform node.

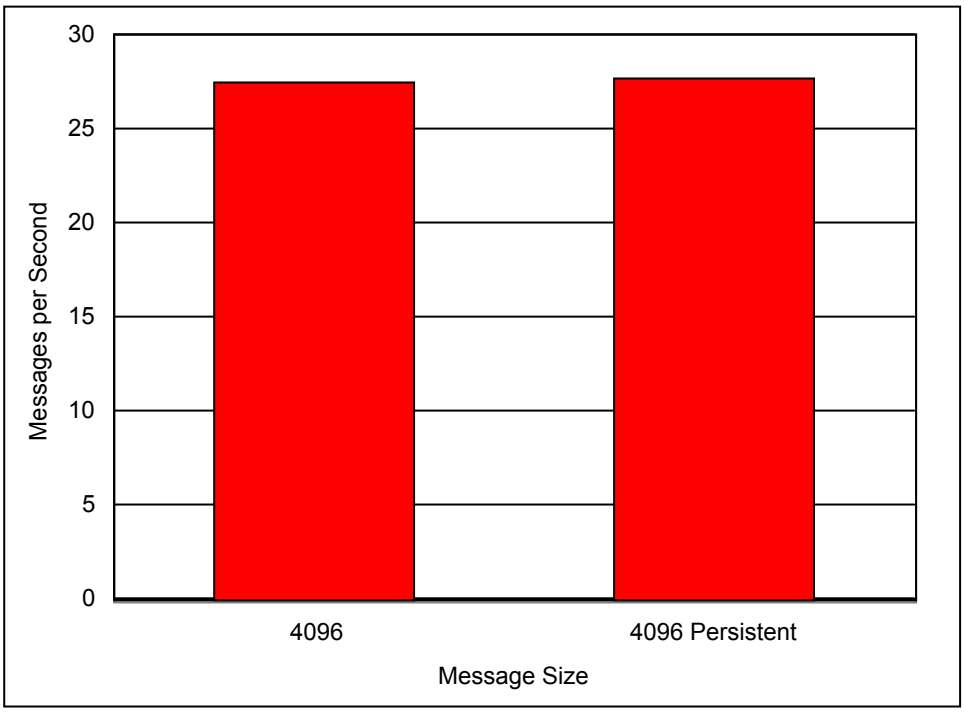

Figure 11: NEONTransform Node Throughput Results

The detailed measurement data showing the effect of running the New Era of Networks nodes within WMQI V2 is available in **Section 6.13 - New Era of Networks Throughput Results**.

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

WMQI 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 WMQI V2 to convert messages between MRM XML, Generic XML , CWF, and MRM TAG formats where MRM XML refers to the predefined XML used within the MRM, Generic XML refers to self-defining XML , CWF denotes a legacy data structure such as a C structure or COBOL copybook, and MRM TAG refers to a predefined structure of fields of fixed length or separated by tags within MRM.

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, MRM XML and MRM TAG and the message throughput achieved was measured. There was a single execution group running the message flow and no additional instances specified. The results are presented in **Table 2** below.

Conversion	**TO**	Generic XML	MRM CWF	MRM XML	MRM TAG
FROM					
Generic XML		160.0	94.5	67.7	58.8
MRM CWF		114.0	101.0	77.7	61.2
MRM XML		78.2	69.4	60.2	48.7
MRM TAG		32.0	30.1	27.6	25.8

Table 2: Message Rates in messages per second, when Converting Between Different Formats

Even when the output message is set to have the same format as the input message there are still significant costs in processing messages because the messages must be parsed, deconstructed by WMQI and then reconstructed into the required output format. However the cost of converting between two formats occurs on a once per message flow basis and not in each node.

It should be noted that TAG format input messages are very costly to process as every byte of the message has to inspected to look for the TAG delimiters.

The detailed measurement data for cost of message conversion is available in **Section 6.7 - Converting Messages Between Formats Throughput Results**.

2.2 Parallel Processing options

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

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

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

3. Run multiple execution groups each processing one or more copies of a message flow. This option uses the most memory as each execution group is a process. If the machine has plenty of memory, this option gives the best throughput as running separate operating system processes requires less storage management.

The use of multiple execution groups is recommended unless there is a sepecific requirement to sequence message processing by userid or message order on the queue, that is Order Mode=By Userid or Order Mode=By Queue Order on the MQInput node.

This section shows the effect of running option 3 for the very complex compute message flow.

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

Figure 12 below shows the results that were obtained as a result of running one, two, and four execution groups for a message flow containing the very complex compute node for varying message size and persistence. The message flow is described in Appendix C Complex Compute Node. The transaction mode values on the MQInput and MQOutput node were set to the value of automatic. The same input and output queues were used for all measurements.

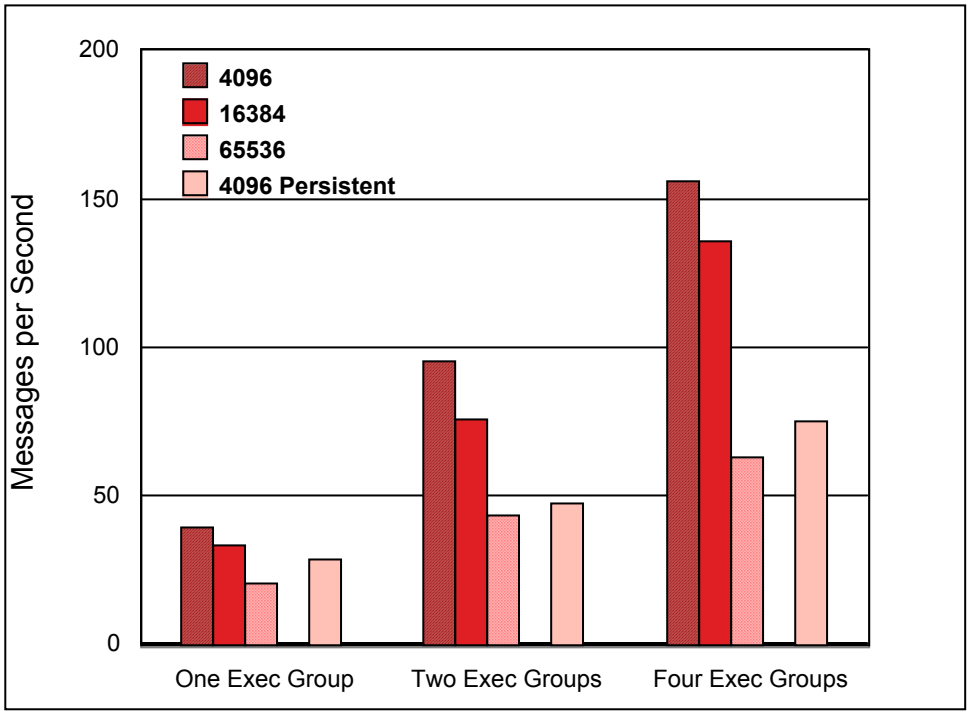

Figure 12: Additional Execution Group Throughput Results

Figure 12 shows that greater message throughput can be achieved by using additional execution groups. With non persistent 4K messages and two execution groups it was possible to achieve over twice the throughput that was achieved for a single execution group. When using four execution groups it was possible to achieve 4 times the throughput that was achieved with a single execution group. All 4 processors were at full capacity (100% CPU busy).

The benefits of running multiple execution groups in this case was significant. There was good scaling of message processing. This is principally because of the nature of the message flow that was used for the measurements. There was a significant amount of ESQL processing in the node. This meant that the level of queue access as a proportion of all processing was low and so the potential for conflicts on queue access was low and consequently multiple execution groups were able to achieve greater throughput.

The detailed measurement data showing the effect of adding execution groups is available in **Section 6.8.1 - The Effect of Increasing The Number Of Execution Groups**.

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

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

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

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

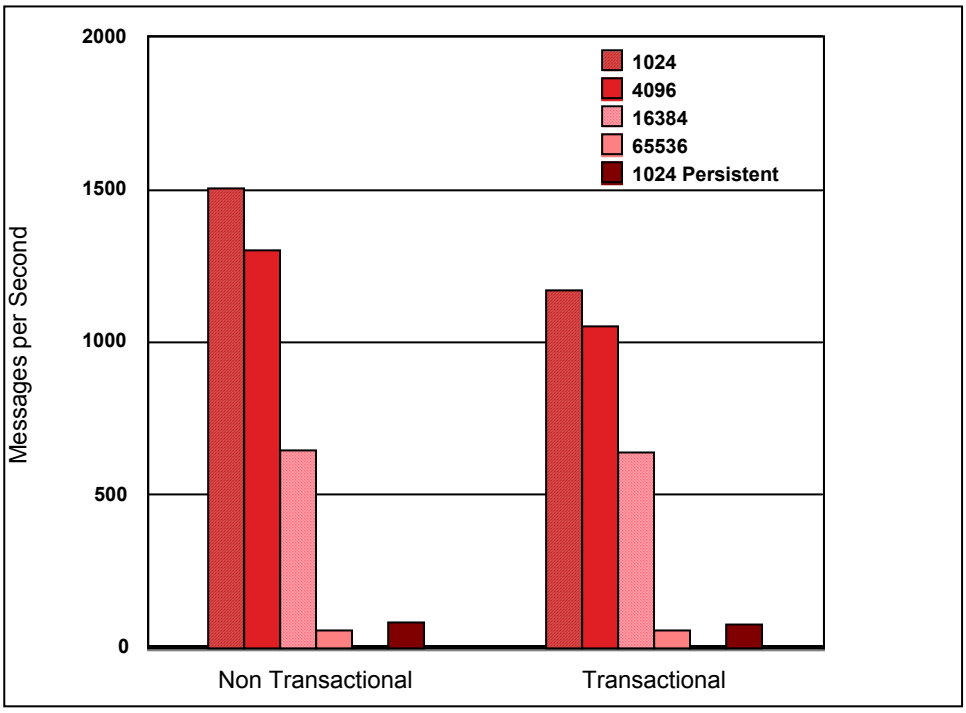

Figure 13: Making a Message Flow Transactional Throughput Results

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

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

The detailed measurement data for showing the effect of making a node transactional is available in **Section 6.1** and **Section 6.9 - The Effect of Making a Message Flow Transactional**.

2.4 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 14 below shows two sets of results. The first is for the case when coordinatedTransaction was set to no on the message flow and transaction mode was set to automatic on the MQInput and MQOutput nodes. The second case shows the results that were obtained when an XA connection was configured and a value of yes was specified for coordinatedTransaction on the message flow.

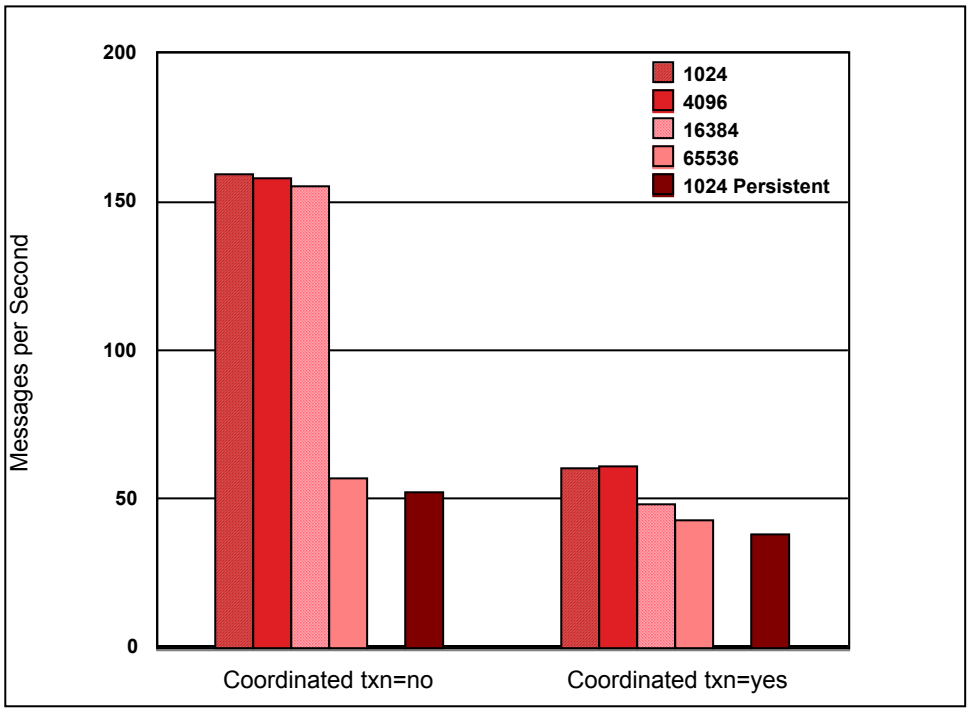

Figure 14: Database Update with XA and coordinatedTransaction=yes.

Figure 14 shows that a message rate of approximately 160 msgs/second was achieved for 1K non-persistent messages and 52 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 approximately 60 msgs/second was achieved for non persistent messages and 38 msgs/second for persistent messages.

The difference in message rates represents the overhead of coordinating the updates as a single unit of work. The database processing in these measurements was simple. In practice it would typically

be more involved and there would be more nodes in the message flow, 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 6.10 - The Effect of using coordinatedTransaction=yes**.

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

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

In order to illustrate the effect of coping with an additional number of subscribers for a given topic a series of measurements were taken with 1, 10, 30, 50, 70, 100, and 1000 subscribers. Messages of varying size and persistence were published to a single topic. The results obtained are presented in **Figure 15** below. The X axis shows the number of subscribers. The Y axis shows the number of seconds taken to process a message. It is derived from the reciprocal of the message rate.

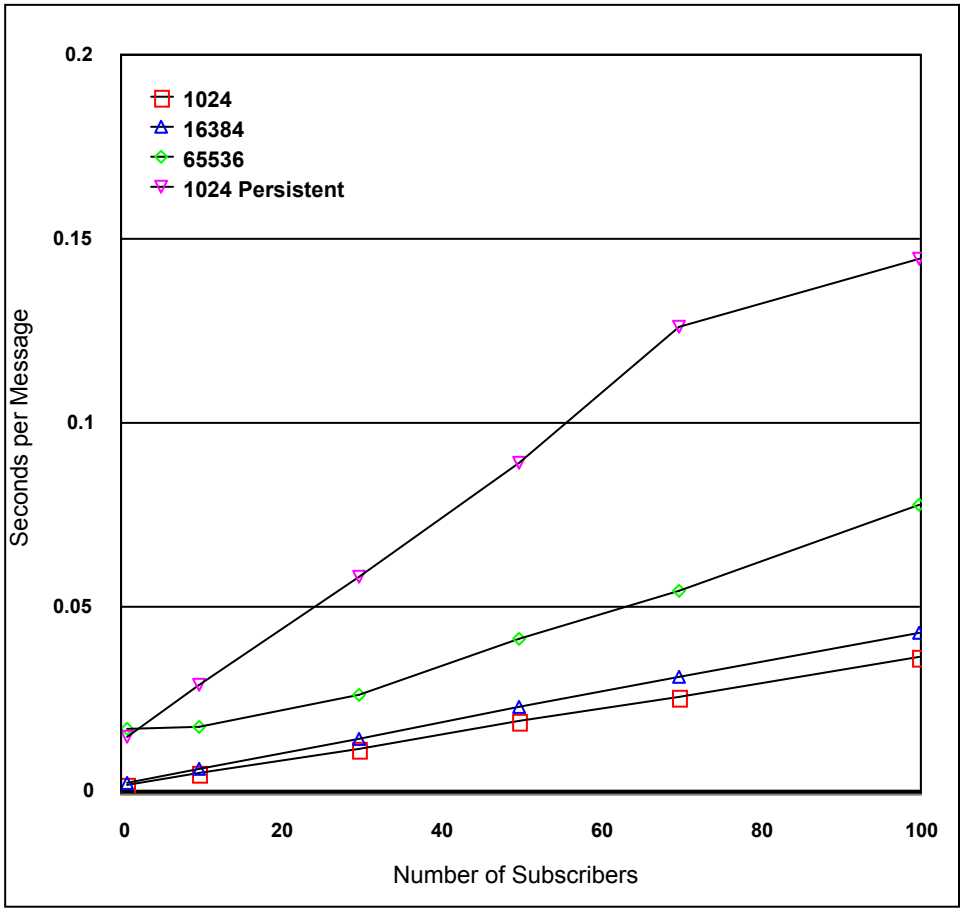

Figure 15: Varying Number of Subscribers

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

The cost of publishing persistent messages is significantly higher as the processing is dominated by the necessary I/O processing. Before examining the measurement data for varying number of subscribers it is important to understand the way in which the measurement was taken.

For each subscriber that registered to receive publications the published message was written to a queue for that subscriber. With 30 subscribers for example, a single message was written to each of 30 queues. In the measurement environment there was a background program consuming all but one of the published messages. Taking the example of 10 subscribers, 9 of the published messages were consumed by this program. The remaining message was read by the client program emulating the subscriber. In this situation a message count of 1 was registered for the purposes of reporting message rates, although the WMQI V2 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.

An extrapolation from 100 to 1000 subscribers for a non-persistent 1k message would predict a message rate of approximately 2.7msgs second. An actual measured rate was 2.3 messages per second.

From some additional measurements that were taken it has been shown that the performance obtained when using publication nodes is dependent on the number of subscriptions that 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.

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

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

The detailed measurement data showing the effect of an increasing number of subscribers is available in **Section 6.5** and **Section 6.11 - The Effect of Increasing the Number of Subscribers.**

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

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

Figure 16 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.

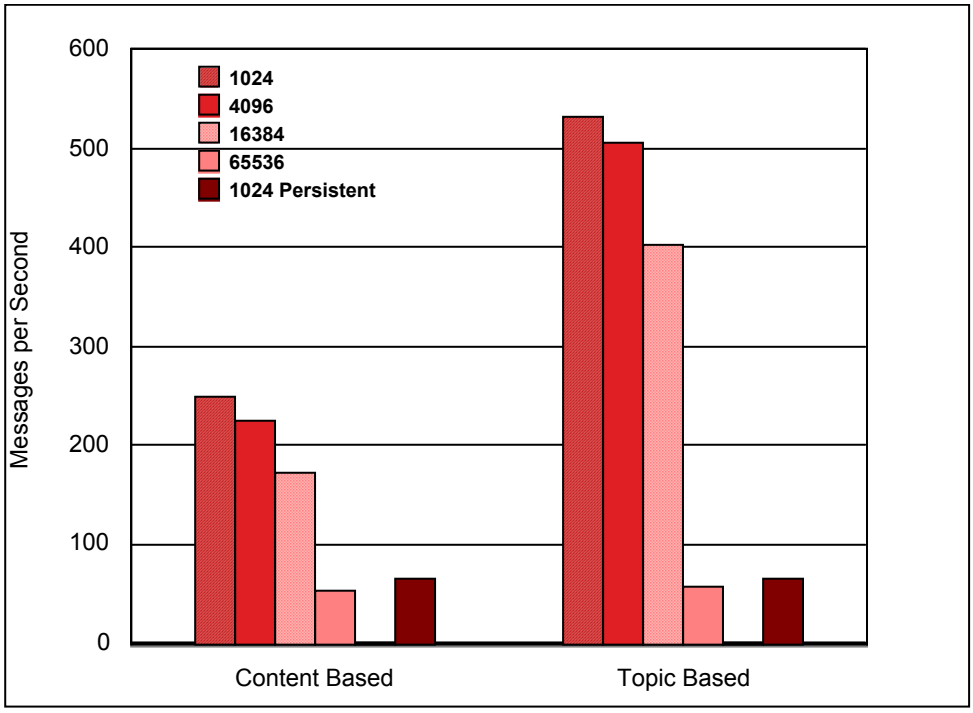

Figure 16 Content vs Topic Based PubSub

Figure 16 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 6.5** and **Section 6.12 - Content vs Topic Based PubSub**.

2.7 What Is The Effect of Using A Trusted Broker?

Use of a trusted broker can, in some situations, lead to an improvement in the message rate which is achieved when processing messages with WMQI V2. This improvement comes as a result of the reduced pathlength between the broker and MQSeries (a consequence of the broker being considered to be trusted by MQSeries). The improvement is only obtained when messages within MQSeries are accessed by the broker. For WMQI V2 this means the execution of MQInput and MQOutput nodes.

The amount of benefit observed when using a trusted broker is dependent on the situation. The gain is proportional to the frequency of use of MQInput and MQOutput nodes in the message flow.

Table 3 below shows the effects on message throughput of using a trusted broker in three different cases. The first was a very simple message flow consisting of an MQInput/MQOutput message flow. The second case consists of a Publish/Subscribe message flow where there was one publisher and 100 subscribers. The third case was the very complex compute node described in **Section 2.3.4 - Very Complex Compute Node.**

Test Case	% Improvement
MQInput/MQOutput	46
Publish/Subscribe with 100 subscribers	53
Very Complex Compute Node	3

Table 3: Use of a Trusted Broker For Different Message Flows

Where the level of queue access is very high and there is little additional processing in the message flow it is possible to gain significant benefit from using a trusted broker as is shown by the MQInput/MQOutput message flow. This message flow contains only MQInput and MQOutput nodes.

In the very complex compute node case the majority of the processing is spent in the ESQL processing and as a result the level of activity by MQInput and MQOutput nodes is much lower. There was no measurable benefit from using a trusted broker in this case.

2.8 How Do NEONTransform and Compute nodes Compare?

2.8.1 NEONTransform Node

New Era of Networks Rules and Formatter Version 5.2 provides a new parser, NEONMSG. This is the domain parser for messages defined in the Rules and Formats database. The old domain parser remains and is supplied for compatibility reasons. However it cannot be used to access any of the new Rules and Formats support. The NEONMSG parser generates a two-dimensional message tree. It creates vertical levels in the message tree to represent the components of a compound message format, whereas the old NEON parser flattened all of the fields to a single vertical level.

NEONMSG also has the ability to serialize a WMQI message tree into a wire format message. This was not possible with the previous parser. This serialization is restricted to output formats defined in the Rules and Formats database. The WMQI message tree could be built using ESQL in a compute node prior to serialization. This was not possible with the previous domain parser.

With WMQI V2 it is possible to transform a message defined in a New Era of Networks input format to one defined in a New Era of Networks output format in one of two ways: Firstly using a NEONTransform node and secondly using a compute node with uses the NEONMSG domain parser to parse the input message and write the output message format. The conversion using the NEONTransform node was performed using a message flow which consisted of an MQInput, NEONTransform and an MQOutput node

Using a compute node and the NEONMSG domain parser, it is possible to generate a Generic XML output message from a message described by a New Era of Networks input format. The results labeled NEONTransform in **Figure 11** show the results that were obtained as a result of using a NEONTransform node to perform message conversion

2.8.2 Compute Node NEONMSG Transformation

Throughput measurements were taken in order to compare the relative performance of the two methods of message transformation and to also show the cost of conversion to XML. An input message with 100 fields was transformed from one format to another. A single message size of 4096 bytes was measured using non persistent and persistent messages.

The conversion using the compute node and NEONMSG parser was performed using a message which consisted of an MQInput, Compute and MQOutput node.

Figure 17 below shows the results that were obtained when converting the message using a compute node and the NEONMSG domain parser. The results labeled NEON to NEON show the results which were obtained when the output message was written using a New Era of Networks output format. The results labeled NEON to XML show the results which were obtained when the output message was written as Generic XML.

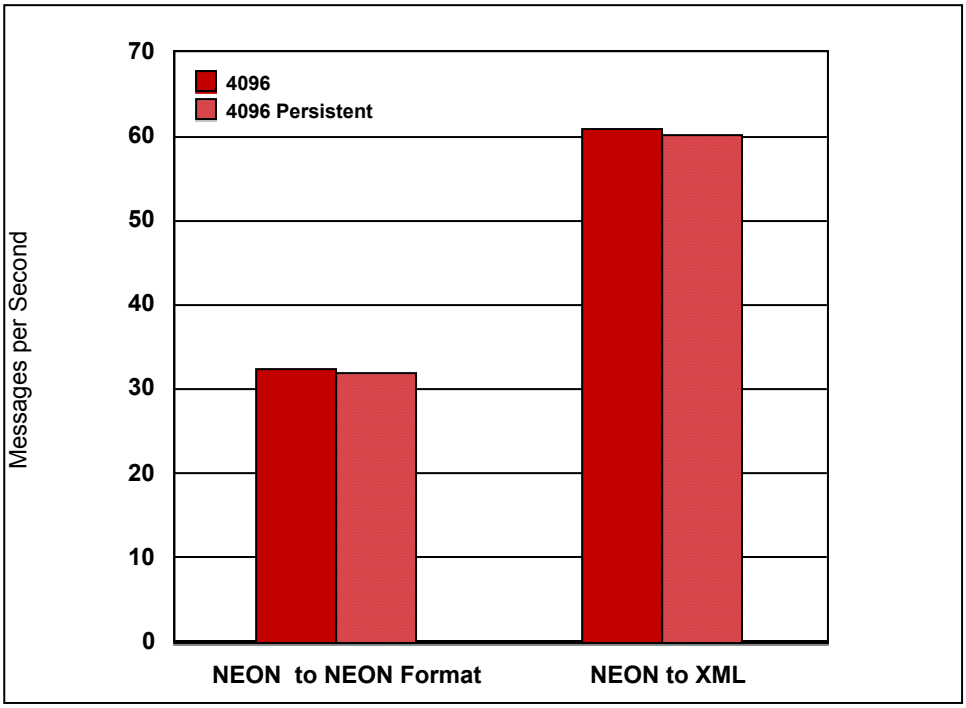

Figure 17: NEONMSG Domain Compute node Throughput Results

The results show that the conversion of a message from a New Era of Networks input format to a New Era of Networks output format was performed more efficiently using a compute node and the NEONMSG parser (Compare Compute and NEONMSG in **Figure 17** with NEONTransform in **Figure 11**).

Writing the output message as generic XML instead of as a serialized, byte format, message was more efficient and so achieved a higher message rate.

The detailed measurement data showing the effect of running the New Era of Networks nodes within WMQI V2 is available in **Section 6.13 - New Era of Networks Throughput Results**.

2.1 *What is the cost of running a Plug-in ?*

The function of a WMQI V2 broker can be extended through the use of additional processing nodes. These nodes are know as *plug-in* nodes and complement the supplied IBM primitives such as the MQInput or Trace nodes.

Plug-in nodes can be written in one of two programming languages: C and Java. See the *Websphere MQ Integrator Programming Guide* for details on how to create a plug-in node.

The plug-in node has the potential to vary from simple to complex in its processing. This is exactly the same as for the compute node. The degree of complexity specified has a direct bearing on the message throughput rates which can be achieved using nodes of that type. The complexity will vary from plug-in to plug-in.

A series of measurements were taken to illustrate the cost of using a plug-in node. The function implemented in the plug-in was minimal. It changed the value of a single field in an incoming message. This is similar to the Simple Compute node which is discussed in Section 2.3.1, Simple Compute Node.

This test was implemented using both a C and Java plug-in. The code for the C plug-in was that given in the SampleTransform example which is shipped with WMQI V2.1 Java code was written for the Java plug-in in order to produce the same effect as the C plug-in.

In order to run the tests two message flows were defined. The first consisted of an MQInput, C Plug-in and an MQOutput node. The second message flow consisted of an MQInput, Java Plug-in and an MQOutput node.

Figure 18 below shows the results that were obtained as a result of running the message flow with varying message sizes and persistence. There was a single instance and single execution group running the message flow.

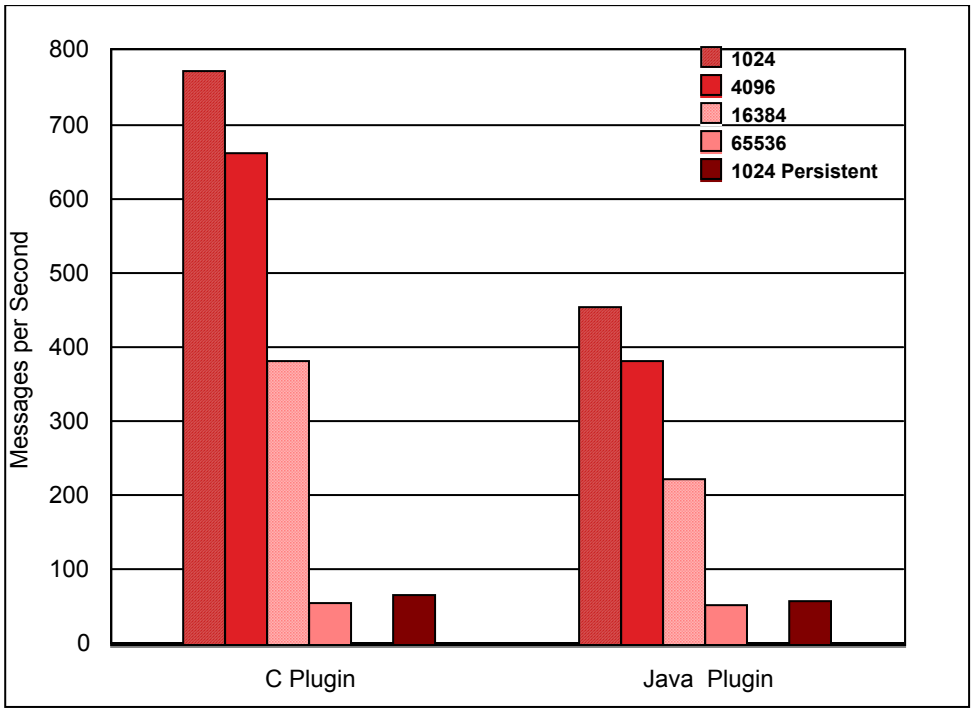

Figure 18: Plug-in node Throughput Results

The detailed measurement data showing the effect of running the plugin nodes within WMQI V2 is available in **Section 6.14 - Plug-in Nodes**.

3.0 CAPACITY PLANNING

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

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

The way in which WMQI 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 WMQI V2. It is possible however to provide a series of guidelines in order to get an initial estimate of the required capacity. Once a prototype implementation has been developed, future resource requirements can be based on measurement and observation of the prototype and its successor implementations, which is the only meaningful exercise for you. The problem is invariably one of getting started. This section helps with that process.

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

Projecting measurement results to other machine types is difficult because performance depends on many factors, such as processor speed and instruction cache sizes for example. The only method currently available to gauge relative machine performance is to use published performance figures such as SPECInt benchmark results that 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 WMQI V2 nodes being used.
- The amount of processing in the nodes, simple vs complex compute nodes for example.
- The number of WMQI 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 Pentium III Xeon (700Mhz) 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. Ideally one with a fast write nonvolatile cache. When using 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. In order to increase CPU utilisation and message throughput you should examine the benefits of running multiple copies of the message flow.

3.2 Scaling Message Throughput

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

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

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

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

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

The level of contention in any implementation is largely determined by the nature of the application being implemented and minimizing the effects of contention is an essential part of the application architecture, design and implementation. This is a large subject and is not covered in any level of detail in this report. 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 30% 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 with persistent messages. The maximum number of persistent messages that 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 disk subsystem with a fast write nonvolatile cache on which to locate the MQSeries queue manager log. Another possibility is to use a solid state disk.

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 2.9.3, 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.

With persistent messages the increase in message throughput 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.

These 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 over 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, 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 WMQI 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 WMQI V2 implementation.
- The Broker. There may be multiple brokers and within these multiple execution groups and so multiple operating system processes.
- MQSeries Queue Manager. There will be one queue manager per broker.
- Relational Database. A DB2 system is required to hold information on behalf of the Configuration Manager and broker. Additional relational databases may be in use which hold business data.

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

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

The amount of memory required by a broker will depend on the way in which it is configured. An guideline is to allow 300MB for WMQI V2 and its dependent software(broker related components only, no Configuration Manager or Control Center), with an additional 40 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 or you have a large number of nodes in a flow or very large messages being processed you must make an allowance and amend the amount of memory required accordingly.

3.4 Recommended Minimum Configurations

This section contains recommendations on the type of hardware on which an WMQI V2 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 WMQI 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 with 256MB memory.

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

The specification of the broker machine is more difficult to determine since it requires knowledge of the expected message rate, the types of node that are to be used and the level of transaction control that is used. A recommended minimum specification would be a 2 way processor with 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 non volatile fast write cache is recommended for the device on which the MQSeries queue log manager is located. Where the message rate is less than 25 msgs/second per second fast I/O will improve message response time only. Where the rate is greater than 25 msgs/second then there will be an improvement in message throughput.
A separate disk is also recommended for the 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 WMQI V2 configuration.

4.1 Understand Recovery Requirements

In designing messageflows within WMQI 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 than is actually required will be undertaken. This will lead to a drop in the throughput rate that is achievable with a message flow as the flow becomes I/O bound.

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

If a message flow is to involve data held in an external resource manager, i.e. not a queue manager, then consider using the coordinatedTransaction parameter in order to make all changes to external data within the scope of a single unit of work. 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 when deciding to use MQSeries transactional control on messageflows. Maybe this is something that is only required for persistent messages.

4.2 Optimize Queue Manager

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

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

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

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

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

- Use the fastest disks available for the MQSeries log. Possible choices are disks with a fast write non volatile cache capability (such as SSA) or a solid state disk. Such devices are capable of delivering I/O times of the order of 1ms compared with 10-16ms for a SCSI disk. Whichever type is selected it is extremely important to ensure that data will not be lost in the event of a power failure. The loss of log data is likely to compromise the integrity of the queue manager.

- Do not use IDE disks, as recovery cannot be guaranteed. This is because the IDE controller indicates that the write has completed when in fact it has not yet been attempted and could possibly fail.

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

with the queue manager. This can be achieved by using multiple copies of a message flow, additional instances or multiple execution groups.

4.3 Configuration Considerations

Consider the following points when building an WMQI V2 configuration:

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

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 WMQI V2 by running multiple copies of a message flow. Doing this will result in the creation and use of another thread or process which provides the potential to increase CPU utilization by using another processor. These approaches are only effective on a machine that has multiple processors. With a single processor machine it will not be possible to improve throughput in this way. However, it may still be necessary to configure multiple execution groups for other reasons.
- In maximizing multiprocessor exploitation remember that there are processes associated with the MQSeries queue manager and any databases that may be used from within messageflows. 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 WMQI V2 performance measurements.
- Avoid small message flows which use MQSeries queue to communicate. Writing a message to a queue is a relatively expensive operation when compared with moving a message between nodes in the same message flow. It would be better to form one larger single message flow, do not move to the other extreme though and put all processing into one single flow.
- Run the broker as a trusted application where appropriate. This is achieved by using the ‘-t’ flag on the mqsicreatebroker command and by ensuring that the e environment variable MQ_CONNECT_TYPE is set to FASTPATH in the environment in which the broker is started.

- Run the MQSeries queue manager listener process as a trusted application. This is achieved by ensuring the environment variable MQ_CONNECT_TYPE is set to FASTPATH in the environment in which the listener is started .

- Compute nodes are expensive in processing costs because they build a representation of the input message. Aim to minimize the number of compute nodes therefore. Dependent on the circumstances, you may consider using a filter node instead of a compute node if message selection is required.

- When using publication nodes ensure that the open queue cache size is set appropriately. See section 2.12 *What Effect Does an Increasing Number of Subscribers Have on Publish/Subscriber Throughput?* for more details.

- When designing messages, make them as simple as possible. Large and more complex messages require more parsing. This consumes more CPU.

- Monitor the number of application handles in any databases containing business data (non WMQI 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.

- Use fast disks for the queue manager log. See Section 4.2 - Optimize Queue Manager above.

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

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

5.0 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

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

The server machine software consisted of:

- Solaris 2.8.
- MQSeries V5.2.
- WebSphere MQ Integrator for Sun Solaris V2.1.
- DB2 for Solaris V7.1.

Client Machine

The client machine hardware consisted of

- An IBM Netfinity 5500 M20 with 4 * 550Mhz Pentium III Xeon processors.
- Six 8.0 GB SCSI hard drives formatted to use NTFS.
- 1GB RAM.
- 100 Mb Ethernet Card.

The client machine software consisted of:

- Microsoft Windows NT V4.0 with Service Pack 6a.
- MQSeries V5.2.

Network Configuration

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

6.0 APPENDIX B - MEASUREMENT DATA

This section contains the detailed results of the WMQI V2 performance measurements described in **Section 2.0 - Measurement Results**. For an explanation of column headings see **Section 5.0 - Glossary**. The CPU utilization reported for the 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 MQSeries Listener, DB2 where appropriate etc. in addition to the CPU used by the WMQI 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.

6.1 MQInput/MQOutput Throughput Results

Persist	Msgsize	msgs/sec	Mbits/sec	cpu
no	1024	1504	23.5	50
no	4096	1300	81.2	57
no	16384	646	161.5	50
no	65536	56.2	56.2	4
yes	1024	79.9	1.2	8

6.2 Compute Node Throughput Results

Simple Compute				
Persistence	Message size	Messages / Second	Mbits / Second	CPU utilisation
no	1024	518	8	35
no	4096	412	25.7	35
no	16384	233	58.2	35
no	65536	53.6	53.6	17
yes	1024	61	0.9	11
Complex Compute				
Persistence	Message size	Messages / Second	Mbits / Second	CPU utilisation
no	4096	101	6.3	28
no	16384	67.1	16.7	29
no	65536	30.1	30.1	30
yes	4096	45.3	2.8	16
Multiple Complex Compute				
Persistence	Message size	Messages / Second	Mbits / Second	CPU utilisation
no	4096	32.2	2	26
no	16384	27.7	6.9	26
no	65536	17.8	17.8	27
yes	4096	26.3	1.6	24
Very Complex Compute				
Persistence	Message size	Messages / Second	Mbits / Second	CPU utilisation
no	4096	39.5	2.4	26
no	16384	33.4	8.3	27
no	65536	20.7	20.7	28
yes	4096	28.6	1.7	24

6.3 Database Node Throughput Results

DB2 Insert and delete - coordinated transaction = yes				
Persistence	Message size	Messages / Second	Mbits / Second	CPU utilisation
no	1024	54.1	0.8	10
no	4096	47.5	2.9	10
no	16384	44.1	11	11
no	65536	44.8	44.8	17
yes	1024	36.4	0.5	11

6.4 Filter Node Throughput Results

Persistence	Message size	Messages / Second	Mbits / Second	CPU utilisation
no	1024	733	11.4	37
no	4096	591	36.9	38
no	16384	343	85.7	39
no	65536	56.7	56.7	4
yes	1024	60.1	0.9	10

6.5 RouteToLabel Node Throughput Results

1 Destination Entry				
Persistence	Message size	Messages / Second	Mbits / Second	CPU utilisation
no	1024	647	10.1	41
no	4096	633	39.5	44
no	16384	455	113.7	49
no	65536	56.6	56.6	7
yes	1024	60.5	0.9	10
100 Destination entries				
Persistence	Message size	Messages / Second	Mbits / Second	CPU utilisation
no	1024	144	2.2	28
no	4096	142	8.8	29
no	16384	138	34.5	32
no	65536	53.7	53.7	21
yes	1024	52.8	0.8	15

6.6 Publication Node Throughput Results

Persistence	Message size	Messages / Second	Mbits / Second	CPU utilisation
no	1024	533	8.3	35
no	4096	507	31.6	36
no	16384	403	100.7	44
no	65536	57.5	57.5	3
yes	1024	66.6	1	10

6.7 Converting Messages Between Formats

Generic XML message in to generic XML out				
Persistence	Message size	Messages / Second	Mbits / Second	CPU utilisation
no	4096	160	10	29
no	16384	93.2	23.3	30
no	65536	33	33	30
yes	4096	50	3.1	14
Generic XML message in to CWF out				
Persistence	Message size	Messages / Second	Mbits / Second	CPU utilisation
no	4096	94.5	5.9	28
no	16384	74.4	18.6	28
no	65536	40.8	40.8	31
yes	4096	47.3	2.9	17
Generic XML message in to MRM XML out				
Persistence	Message size	Messages / Second	Mbits / Second	CPU utilisation
no	4096	67.7	4.2	28
no	16384	45.2	11.3	28
no	65536	19.2	19.2	28
yes	4096	49.9	3.1	25
Generic XML message in to MRM TAG out				
Persistence	Message size	Messages / Second	Mbits / Second	CPU utilisation
no	4096	58.8	3.6	26
no	16384	51.6	12.9	28
no	65536	31.2	31.2	30
yes	4096	36	2.2	18

CWF message in to generic XML out				
Persistence	Message size	Messages / Second	Mbits / Second	CPU utilisation
no	4096	114	7.1	28
no	16384	78.9	19.7	30
no	65536	33.2	33.2	30
yes	4096	45	2.8	16
CWF message in to CWF out				
Persistence	Message size	Messages / Second	Mbits / Second	CPU utilisation
no	4096	101	6.3	28
no	16384	85.5	21.3	30
no	65536	48.5	48.5	30
yes	4096	50.3	3.1	19
CWF message in to MRM XML out				
Persistence	Message size	Messages / Second	Mbits / Second	CPU utilisation
no	4096	77.7	4.8	28
no	16384	52.3	13	29
no	65536	22.5	22.5	29
yes	4096	45.7	2.8	22
CWF message in to MRM TAG out				
Persistence	Message size	Messages / Second	Mbits / Second	CPU utilisation
no	4096	61.2	3.8	28
no	16384	54.1	13.5	28
no	65536	37	37	31
yes	4096	40.1	2.5	20

MRM XML message in to generic XML out				
Persistence	Message size	Messages / Second	Mbits / Second	CPU utilisation
no	4096	78.2	4.8	28
no	16384	56.4	14.1	27
no	65536	25.2	25.2	28
yes	4096	50	3.1	22
MRM XML message in to CWF out				
Persistence	Message size	Messages / Second	Mbits / Second	CPU utilisation
no	4096	69.4	4.3	27
no	16384	56.8	14.2	27
no	65536	33.6	33.6	30
yes	4096	46.3	2.8	21
MRM XML message in to MRM XML out				
Persistence	Message size	Messages / Second	Mbits / Second	CPU utilisation
no	4096	60.2	3.7	28
no	16384	42.1	10.5	28
no	65536	17.8	17.8	28
yes	4096	36.5	2.2	19
MRM XML message in to MRM TAG out				
Persistence	Message size	Messages / Second	Mbits / Second	CPU utilisation
no	4096	48.7	3	26
no	16384	42.1	10.5	27
no	65536	26	26	28
yes	4096	37	2.3	21

MRM TAG message in to generic XML out				
Persistence	Message size	Messages / Second	Mbits / Second	CPU utilisation
no	4096	32	2	26
no	16384	16.2	4	26
no	65536	5.7	5.7	26
yes	4096	25	1.5	25

MRM TAG message in to CWF out				
Persistence	Message size	Messages / Second	Mbits / Second	CPU utilisation
no	4096	30.1	1.8	26
no	16384	16.5	4.1	26
no	65536	5.8	5.8	26
yes	4096	24.5	1.5	25

MRM TAG message in to MRM XML out				
Persistence	Message size	Messages / Second	Mbits / Second	CPU utilisation
no	4096	27.6	1.7	26
no	16384	15.2	3.8	26
no	65536	5.2	5.2	26
yes	4096	22.7	1.4	24

MRM TAG message in to MRM TAG out				
Persistence	Message size	Messages / Second	Mbits / Second	CPU utilisation
no	4096	25.8	1.6	26
no	16384	15.5	3.8	26
no	65536	5.8	5.8	26
yes	4096	22.7	1.4	26

6.8 Parallel Processing

This section contains the results of running with multiple execution groups for a message flow with a very complex compute node.

6.8.1 The Effect of Increasing The Number Of Execution Groups

One Execution Group running a single message flow				
Persistence	Message size	Messages / Second	Mbits / Second	CPU utilisation
no	4096	39.5	2.4	26
no	16384	33.4	8.3	27
no	65536	20.7	20.7	28
yes	4096	28.6	1.7	24
Two Execution Groups running a single message flow				
Persistence	Message size	Messages / Second	Mbits / Second	CPU utilisation
no	4096	95	5.9	53
no	16384	75.5	18.8	53
no	65536	43.4	43.4	55
yes	4096	47.5	2.9	28
Four Execution Groups Running a single message flow				
Persistence	Message size	Messages / Second	Mbits / Second	CPU utilisation
no	4096	156	9.7	99
no	16384	136	34	99
no	65536	63.2	63.2	89
yes	4096	74.9	4.6	55

6.9 The Effect of Making a Message Flow Transactional

Persistence	Message size	Messages / Second	Mbits / Second	CPU utilisation
no	1024	1168	18.2	52
no	4096	1054	65.8	57
no	16384	638	159.5	53
no	65536	56.6	56.6	5
yes	1024	78.4	1.2	9

6.10 The Effect of using coordinatedTransaction=yes

DB2 Update - coordinated transaction = no				
Persistence	Message size	Messages / Second	Mbits / Second	CPU utilisation
no	1024	159	2.4	14
no	4096	158	9.8	16
no	16384	155	38.7	21
no	65536	57.1	57.1	9
yes	1024	51.9	0.8	10
DB2 Update - coordinated transaction = yes				
Persistence	Message size	Messages / Second	Mbits / Second	CPU utilisation
no	1024	60	0.9	10
no	4096	61.1	3.8	11
no	16384	48.1	12	10
no	65536	42.6	42.6	15
yes	1024	38.1	0.5	10

6.11 The Effect of Increasing the Number of Subscribers

1 Subscriber Receiving MQRFH2 Type Published Messages				
Persistence	Message size	Messages / Second	Mbits / Second	CPU utilisation
no	1024	533	8.3	35
no	4096	507	31.6	36
no	16384	403	100.7	44
no	65536	57.5	57.5	3
yes	1024	66.6	1	10
10 Subscribers Receiving MQRFH2 Type Published Messages				
Persistence	Message size	Messages / Second	Mbits / Second	CPU utilisation
no	1024	196	3	39
no	4096	193	12	41
no	16384	161	40.2	42
no	65536	56.5	56.5	32
yes	1024	34.2	0.5	24
30 Subscribers Receiving MQRFH2 Type Published Messages				
Persistence	Message size	Messages / Second	Mbits / Second	CPU utilisation
no	1024	84.1	1.3	42
no	4096	80.2	5	43
no	16384	68.3	17	43
no	65536	37.7	37.7	49
yes	1024	17.1	0.2	24
50 Subscribers Receiving MQRFH2 Type Published Messages				
Persistence	Message size	Messages / Second	Mbits / Second	CPU utilisation
no	1024	51.9	0.8	43
no	4096	51	3.1	43
no	16384	43.7	10.9	44
no	65536	24.2	24.2	48
yes	1024	11.2	0.1	28
70 Subscribers Receiving MQRFH2 Type Published Messages				
Persistence	Message size	Messages / Second	Mbits / Second	CPU utilisation
no	1024	38.6	0.6	42
no	4096	36.6	2.2	43
no	16384	31.9	7.9	43
no	65536	18.4	18.4	49
yes	1024	7.9	0.1	24

100 Subscribers Receiving MQRFH2 Type Published Messages				
Persistence	Message size	Messages / Second	Mbits / Second	CPU utilisation
no	1024	27.2	0.4	43
no	4096	25.9	1.6	43
no	16384	23	5.7	44
no	65536	12.8	12.8	47
yes	1024	6.9	0.1	31
1000 Subscribers Receiving MQRFH2 Type Published Messages				
Persistence	Message size	Messages / Second	Mbits / Second	CPU utilisation
no	1024	2.3	0	40
no	4096	2.3	0.1	40
no	16384	2	0.5	41
no	65536	1.3	1.3	45
yes	1024	0.6	0	31

6.12 Content vs Topic Based PubSub

Persistence	Message size	Messages / Second	Mbits / Second	CPU utilisation
no	1024	250	3.9	30
no	4096	226	14.1	31
no	16384	172	43	32
no	65536	54.1	54.1	17
yes	1024	65.5	1	14

6.13 New Era of Networks Throughput Results

NEON Rules				
Persistence	Message size	Messages / Second	Mbits / Second	CPU utilisation
no	1024	114	1.7	28
no	4096	104	6.5	28
no	16384	77.3	19.3	29
no	65536	40.6	40.6	32
yes	1024	53.5	0.8	18
NEON Transform				
Persistence	Message size	Messages / Second	Mbits / Second	CPU utilisation
no	1024	27.5	0.4	26
no	4096	27.7	1.7	26
Compute NEONMSG Reformat				
Persistence	Message size	Messages / Second	Mbits / Second	CPU utilisation
no	1024	32.3	0.5	26
no	4096	31.9	1.9	26
Compute NEONMSG Reformat to XML				
Persistence	Message size	Messages / Second	Mbits / Second	CPU utilisation
no	1024	60.9	0.9	28
no	4096	60.2	3.7	26

6.14 Plug-in Nodes

C plugin				
Persistence	Message size	Messages / Second	Mbits / Second	CPU utilisation
no	1024	771	12	40
no	4096	662	41.3	38
no	16384	380	95	43
no	65536	56.4	56.4	7
yes	1024	64.7	1	10
Java Plugin				
Persistence	Message size	Messages / Second	Mbits / Second	CPU utilisation
no	1024	454	7	38
no	4096	382	23.8	38
no	16384	223	55.7	40
no	65536	53.6	53.6	20
yes	1024	59.2	0.9	11

7.0 APPENDIX C - COMPLEX COMPUTE NODE

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

7.1 Complex Compute Node

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

```
Set OutputRoot=InputRoot;

DECLARE i INTEGER;

DECLARE C INTEGER;

SET C=CARDINALITY(OutputRoot.XML.CSIM.TestCase.Stack.ProcessingPath.Element[]);

SET i = 1;

WHILE i &lt;= C DO

  SET OutputRoot.XML.CSIM.TestCase.ProcessingPath.Component[i].Name =

     OutputRoot.XML.CSIM.TestCase.Stack.ProcessingPath.Element[i].COMPONENT;

  SET OutputRoot.XML.CSIM.TestCase.ProcessingPath.Component[i].Transport.(XML.attr)Type='A';

  SET OutputRoot.XML.CSIM.TestCase.ProcessingPath.Component[i].Transport.Queue =
OutputRoot.XML.CSIM.TestCase.Stack.ProcessingPath.Element[i].QUEUE;

  SET i = i + 1;

END WHILE;
```

7.2 Multiple Complex Compute Node

The multiple complex compute nodes consisted of five identical complex compute nodes that were daisy chained. The logic within each of the complex compute nodes was the same as that for the complex compute node given in **Section 8.1 - Complex Compute Node**.

7.3 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