WebSphere®MQ for JMS V7.1 - Performance Evaluations

Version 1.0

December 2011

Rachel Norris

WebSphere MQ Performance
IBM UK Laboratories
Hursley Park
Winchester
Hampshire
SO21 2JN

Property of IBM

Please take Note!

Before using this report, please be sure to read the paragraphs on “disclaimers”, “warranty and liability exclusion”, “errors and omissions”, and the other general information paragraphs in the "Notices" section below.

First Edition, December 2011.

This edition applies to *WebSphere MQ V7.1* (and to all subsequent releases and modifications until otherwise indicated in new editions).

© Copyright International Business Machines Corporation 2011. All rights reserved.

Note to U.S. Government Users
Documentation related to restricted rights.
Use, duplication or disclosure is subject to restrictions set forth in GSA ADP Schedule contract with IBM Corp.

Notices

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

You should not assume that the information contained in this report has been submitted to any formal testing by IBM.

Any use of this information and implementation of any of the techniques are the responsibility of the licensed user. Much depends on the ability of the licensed user to evaluate the data and to project the results into their own operational environment.

WARRANTY AND LIABILITY EXCLUSION
The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law:

INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION “AS IS” WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY, OR FITNESS FOR A PARTICULAR PURPOSE.

Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore this statement may not apply to you.

In Germany and Austria, notwithstanding the above exclusions, IBM's warranty and liability are governed only by the respective terms applicable for Germany and Austria in the corresponding IBM program license agreement(s).

ERRORS AND OMISSIONS
The information set forth in this report could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; any such change will be incorporated in new editions of the information. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this information at any time and without notice.

INTENDED AUDIENCE
This report is intended for architects, systems programmers, analysts and programmers wanting to understand the performance characteristics of *WebSphere MQ JMS V7.1*. The

information is not intended as the specification of any programming interface that is provided by WebSphere. It is assumed that the reader is familiar with the concepts and operation of WebSphere MQ V7.1.

LOCAL AVAILABILITY
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. Consult your local IBM representative for information on the products and services currently available in your area.

ALTERNATIVE PRODUCTS AND SERVICES
Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

USE OF INFORMATION PROVIDED BY YOU
IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

TRADEMARKS AND SERVICE MARKS
The following terms used in this publication are trademarks of International Business Machines Corporation in the United States, other countries or both:

- IBM
- WebSphere
- JAVA™

Microsoft and Windows are trademarks of Microsoft Corporation in the United States, other countries, or both.

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

EXPORT REGULATIONS
You agree to comply with all applicable export and import laws and regulations.

Preface

This report presents the results of performance evaluations using the JMS clients supplied with WebSphere MQ for Windows V7.1, Linux64 V7.1, AIX V7.1, and z/OS V7.1 and is intended to assist with programming and capacity planning.

Target audience

This SupportPac is designed for people who:

- Will be designing and implementing JMS solutions using WebSphere MQ.
- Want to understand the performance limits of WebSphere MQ JMS.
- Want to understand what actions may be taken to tune WebSphere MQ JMS.

The reader should have a general awareness of the Java programming language, the Java Message Service API, the Windows 2003, Linux64, z/OS, and/or AIX operating systems and of WebSphere MQ in order to make best use of this SupportPac.

The contents of this SupportPac

This SupportPac includes:

- Charts and tables describing the performance headlines of JMS using WebSphere MQ V7.1
- WebSphere MQ JMS messaging using Windows, Linux64 and AIX
- z/OS WebSphere MQ JMS messaging using Client bindings
- Advice on programming with WebSphere MQ JMS for performance

Feedback on this SupportPac

We welcome constructive feedback on this report.

- Does it provide the sort of information you want?
- Do you feel something important is missing?
- Is there too much technical detail, or not enough?
- Could the material be presented in a more useful manner?

Please direct any comments of this nature to **WMQPG@uk.ibm.com**.

Specific queries about performance problems on your WebSphere MQ system should be directed to your local IBM Representative or Support Centre.

Introduction

This report uses the IBM Performance Harness for JMS (available from **https://www.ibm.com/developerworks/community/groups/service/html/communityview?communityUuid=1c020fe8-4efb-4d70-afb7-0f561120c2aa**) to produce message throughput and CPU information on AIX, Linux64, Windows, and z/OS.

The scenarios used in this report to generate the performance data are:

- Local queue manager scenario.
- Client channel scenario.
- Publish-Subscribe scenarios.
- Requester-Responder scenarios.
- Asynchronous scenarios.
- Synchronous scenarios.

Unless otherwise specified, the standard message sized used for all the measurements in this report is 2KB (2,048 bytes).

CONTENTS

TABLES

FIGURES

1 Overview

The four scenarios used in Chapters 2, 3 and 4 in this report are :-

1) Requester/Responder
2) Multiple sets of (Producer, Queue, Consumer)
3) Publish Subscribe (single Publisher, single Topic, multiple Subscribers)
4) Multiple sets of (Publisher, Topic, Subscriber)

These are measured and reported with Persistent and non-Persistent messages on Windows, Linux64, AIX, and z/OS systems.

Due to the differences between the hardware used for each operating system, it is not possible to compare throughput across operating systems.

- The message format used is a 2048 byte JMSTextMessage.
- Persistent messages are transactional. (session = connection.createSession(true, Session.AUTO_ACKNOWLEDGE); This also significantly improves throughput when multiple threads are processing messages on the same queue especially when using non cached disks for the MQ Log.
- The 'multiple sets' message producers insert messages at a fixed rate.
- The client used is the "IBM Performance Harness for Java Message Service".
- Message Producers/Consumers are co-located on the Queue manager system for 'Local Bindings' Measurements and on Linux64 driver systems for 'Client' measurements.
- Each sample point reported is the average of two minutes of data collection.

The graphs in this report show the number of messages per second that are processed by all the connected applications or clients. So for example a Publisher publishing at 100 msgs/sec with two Subscribers would result in a throughput of 300 msgs/sec. The only exception to this is the requester/responder scenario where the throughput measured is the number of messages produced by the requester.

The tables in the report show the peak throughput achieved and the number of connected applications and CPU usage at the peak throughput.

2 Local Bindings

.

2.1 Local Queue Manager Requester-Responder Scenario

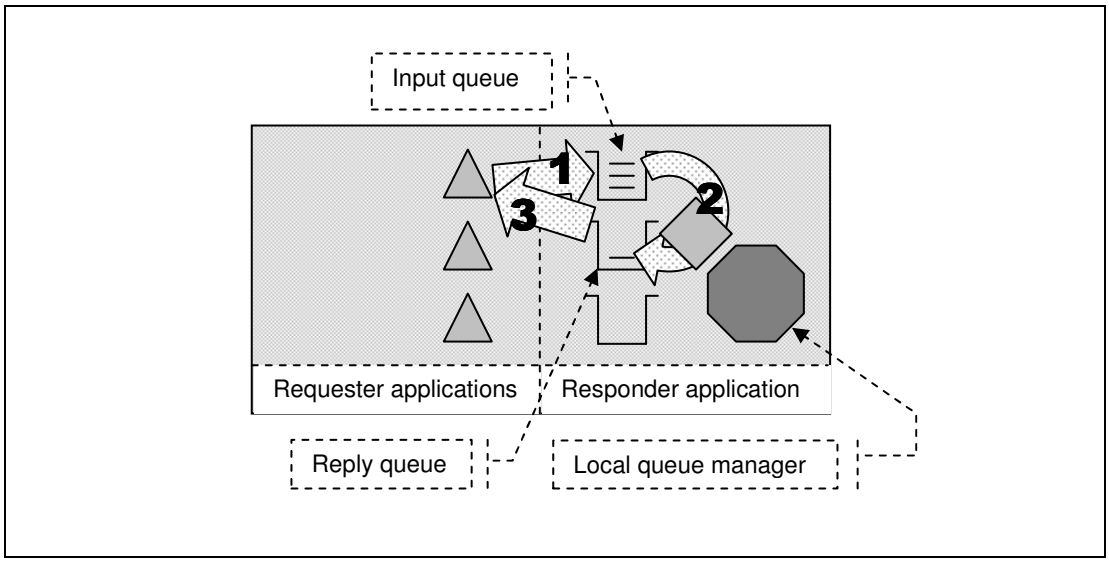

Figure 1 – Connections into a local queue manager

1) The Requester application puts a message to the common input queue on the local queue manager, and holds on to the message identifier returned in the message descriptor. The Requester application then waits indefinitely for a reply to arrive on the common reply queue.

2) The Responder application gets messages from the common input queue and places a reply to the common reply queue. The queue manager copies over the message identifier from the request message to the correlation identifier of the reply message.

3) The Requester application gets a reply from the common reply queue using the message identifier held from when the request message was put to the common input queue, as the correlation identifier in the message descriptor.

Message count is the number of messages produced by the Requester. Since the Requester can only produce a new message when it has retrieved a reply to the previous message from the Reply queue, the number of messages produced by the Requester is a measure of the number of round-trips.

Non-persistent and persistent messages were used in the local queue manager tests, with a message size of 2KB. The effect of message throughput with larger messages sizes is investigated in the “Large Messages” section.

Application Bindings of both the requester and responder programs are ‘Shared’.

2.1.1 Requester-Responder Non persistent Messages – Local

The graphs below show the non-persistent message throughput achieved using an increasing number of requesting applications with a local queue manager for Linux64, Windows, and AIX.

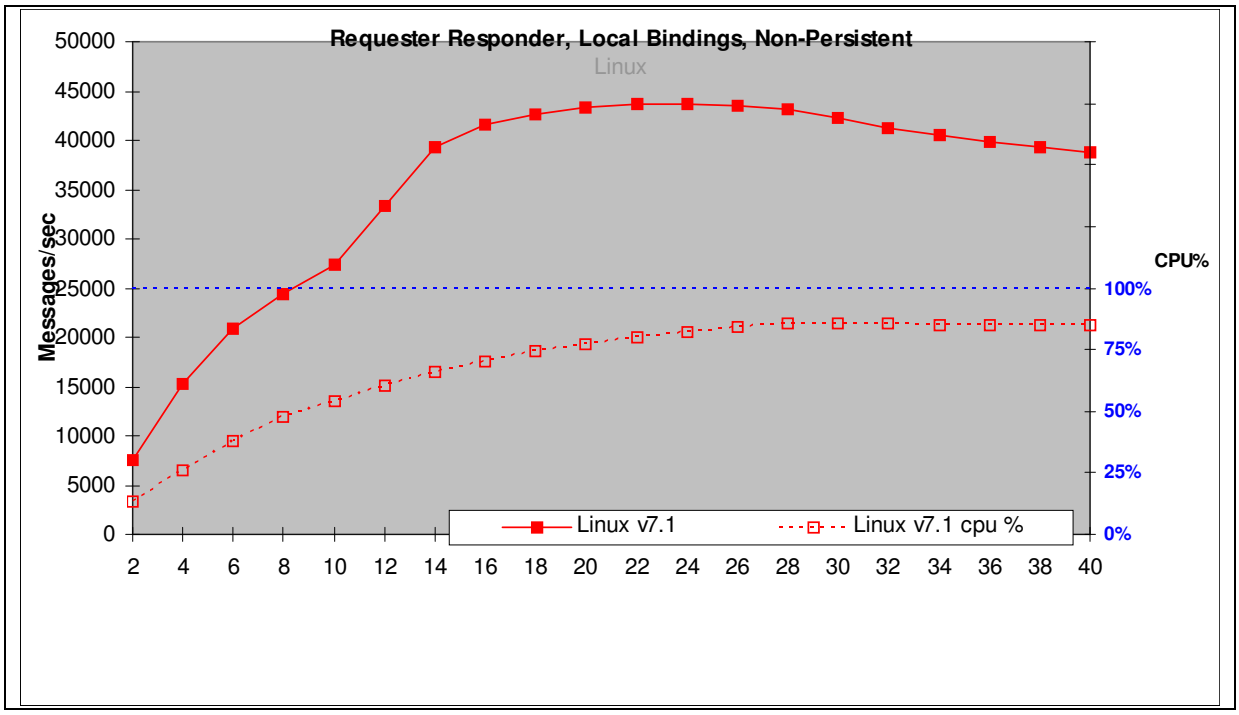

Figure 2 – Requester-Responder, non persistent messages, local queue manager, Linux64

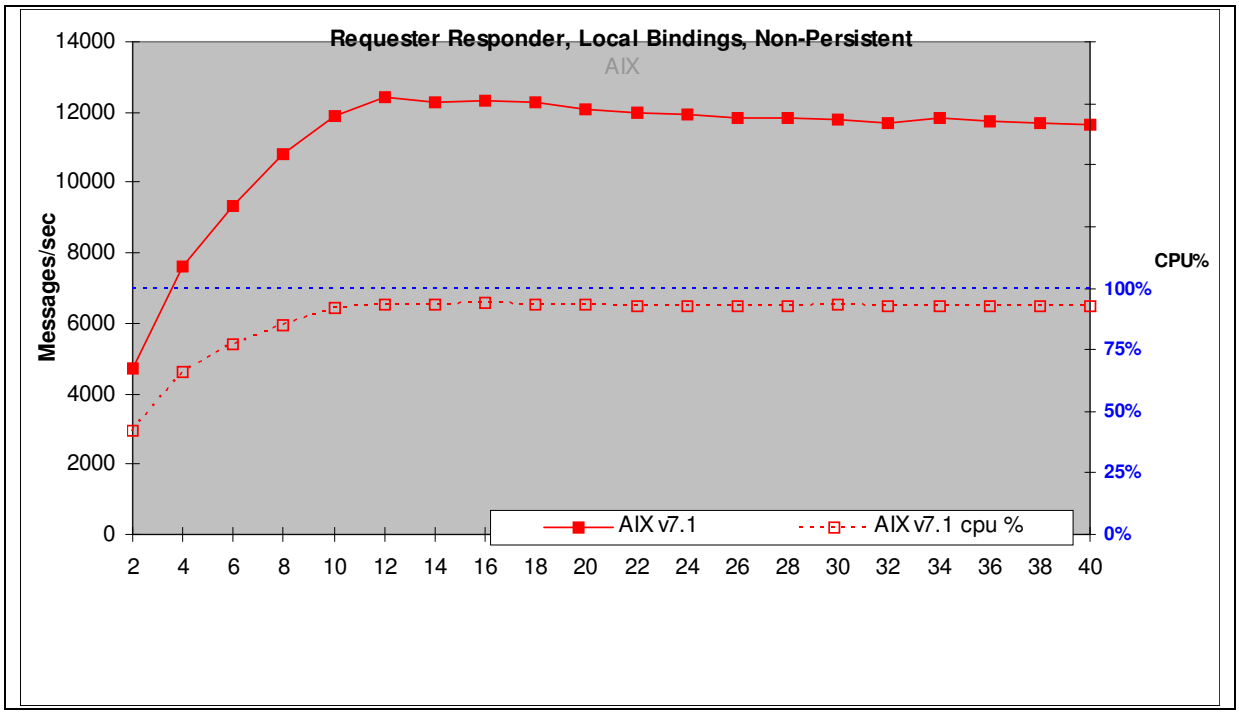

Figure 3 – Requester-Responder, non persistent messages, local queue manager, AIX

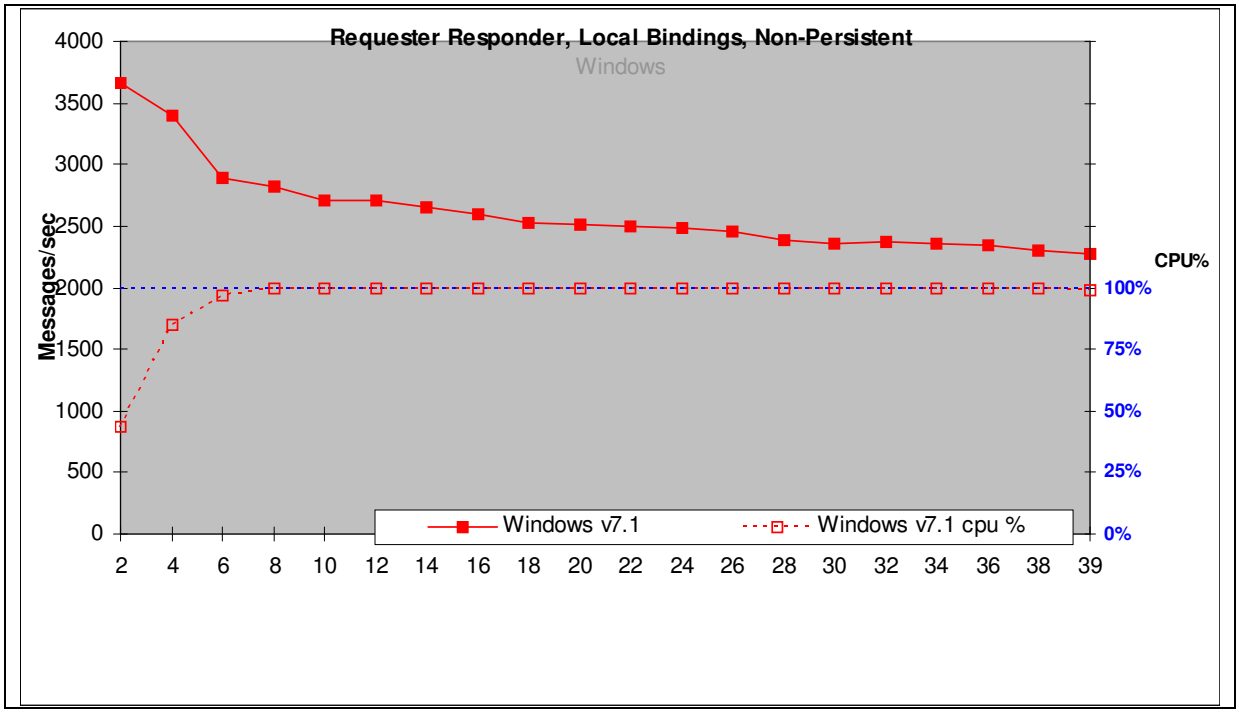

Figure 4 – Requester-Responder, non persistent messages, local queue manager, Windows

Test name: **RRLN**	**Apps**	**Round Trips/sec**	**CPU**
Linux64	**24**	**43668**	**82%**
AIX	**12**	**12440**	**94%**
Windows	**2**	**3670**	**43%**

Table 1 – Requester-Responder, non-persistent messages, local queue manager

The AIX and Linux64 operating systems continue to process close to the peak message load as additional work requests are submitted while Windows gradually processes less work as additional work requests are submitted.

2.1.2 Requester-Responder Persistent Messages – Local

The graphs below show the persistent message throughput achieved using an increasing number of requesting applications with a local queue manager for Linux64, Windows, and AIX.

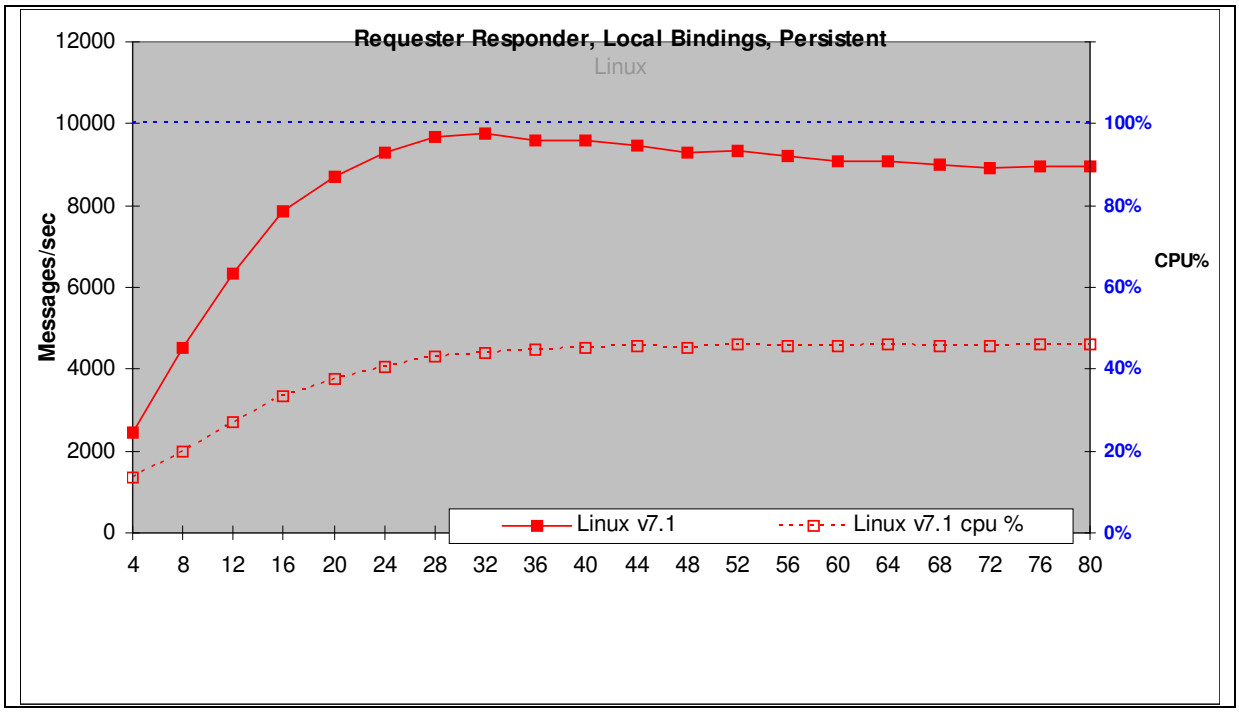

Figure 5 – Requester-Responder, Persistent messages, local queue manager, Linux64

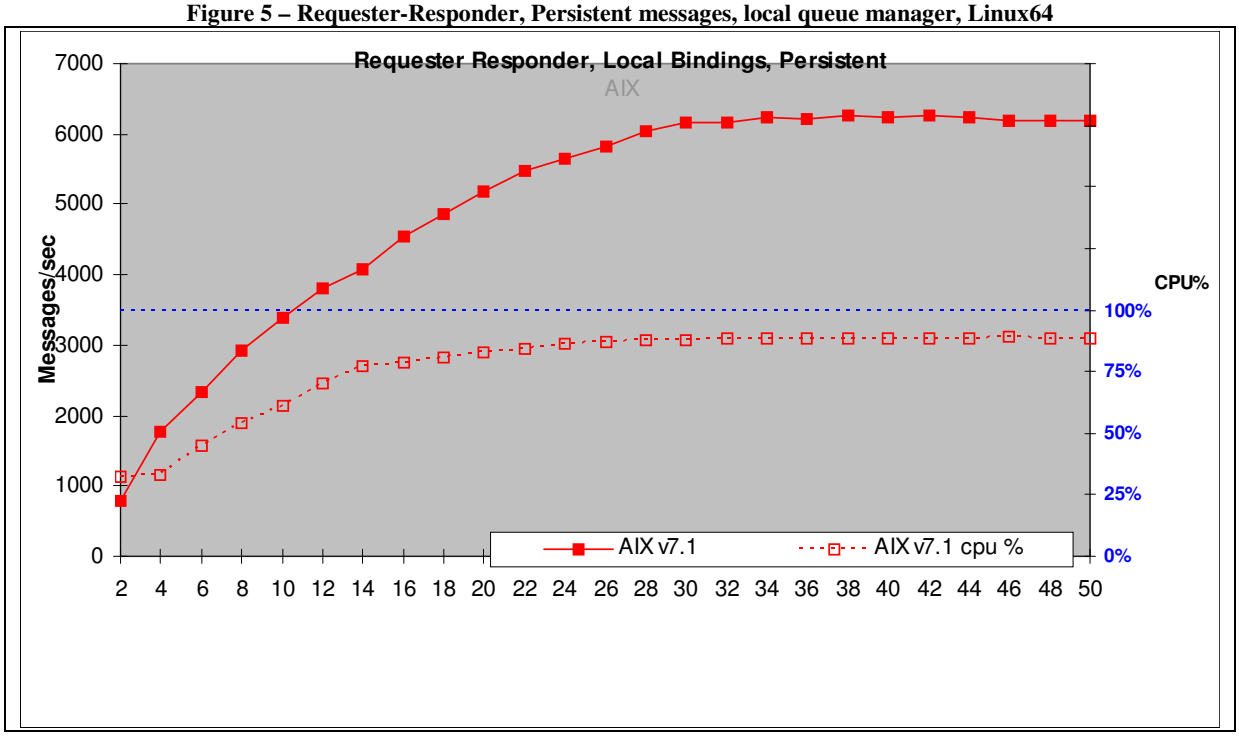

Figure 6 – Requester-Responder, Persistent messages, local queue manager, AIX

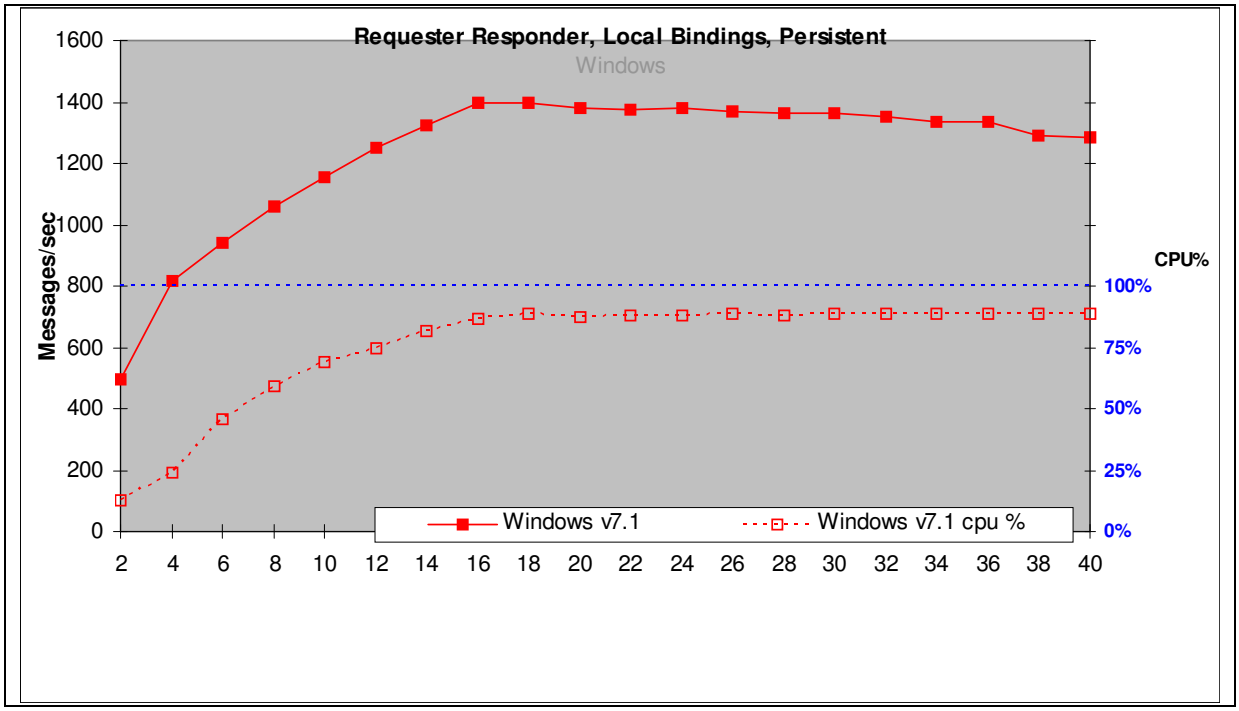

Figure 7 – Requester-Responder, Persistent messages, local queue manager, Windows

Test name: **RR4QLN**	**Apps**	**Round Trips/sec**	**CPU**
Linux64	**32**	**9743**	**44%**
AIX	**34**	**6232**	**88%**
Windows	**16**	**1396**	**87%**

Table 2 – Requester-Responder, Persistent messages, local queue manager

2.2 Point to Point , Multiple (Producer, Consumer, Queue) Scenario

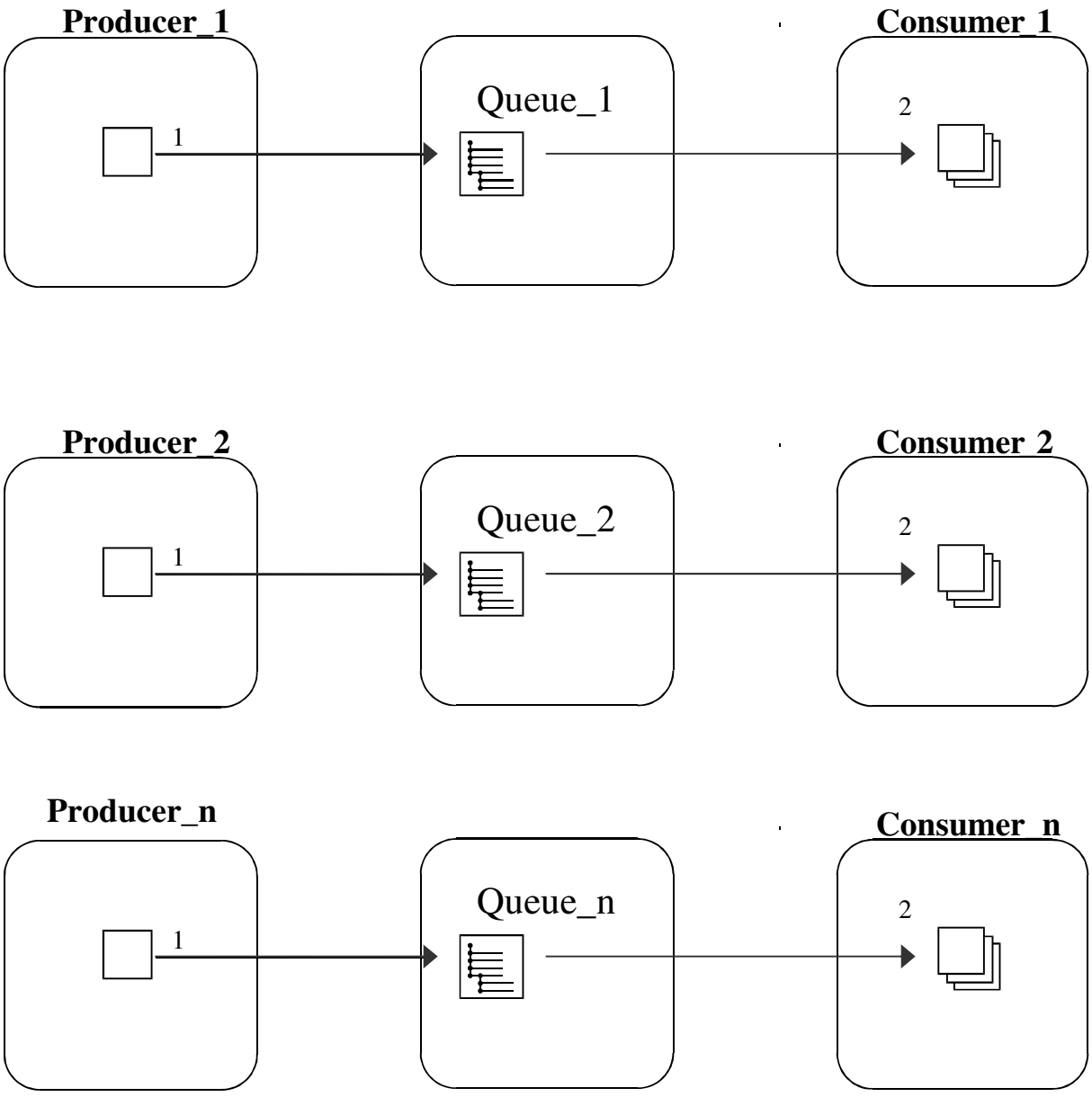

Figure 8 – Producer/Consumer, non persistent

Each Queue is used by only one Producer and one Consumer. The message Producer inserts messages at a predefined rate. The message production rate is 1600 per second for non-persistent and 400 per second for persistent messages. The number of (Producer, Consumer, Queue) triplets are gradually increased and the maximum rate occurs when the consumers prevent the Queue from exceeding a queue depth of one. This test case uses asynchronous messaging hence there is no connection between the number of messages in the system and the number of producers or consumers. Message count is the number of messages put to queue by the producer plus the number of messages retrieved by the consumer. Hence, a message created by the producer and received by the consumer results in a message count of two.

2.2.1 Producer Consumer, Non Persistent Messages, Local

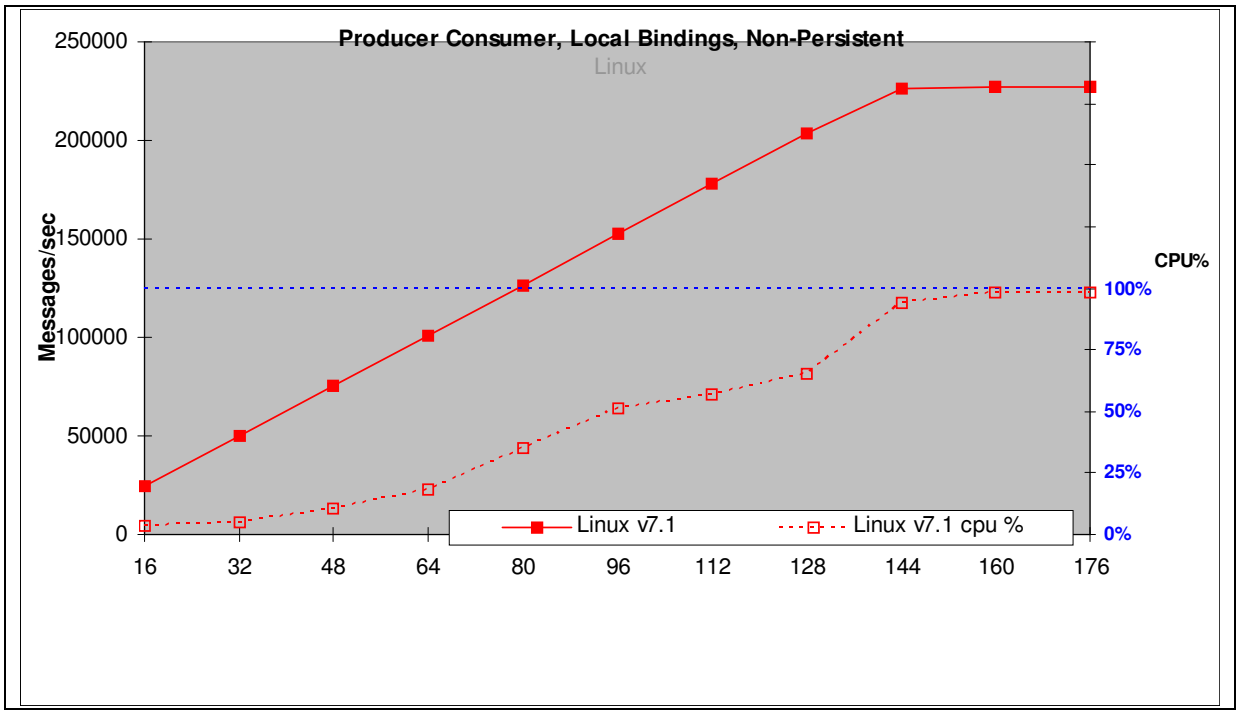

Figure 9 – Producer/Consumer, non persistent messages, local queue manager, Linux64

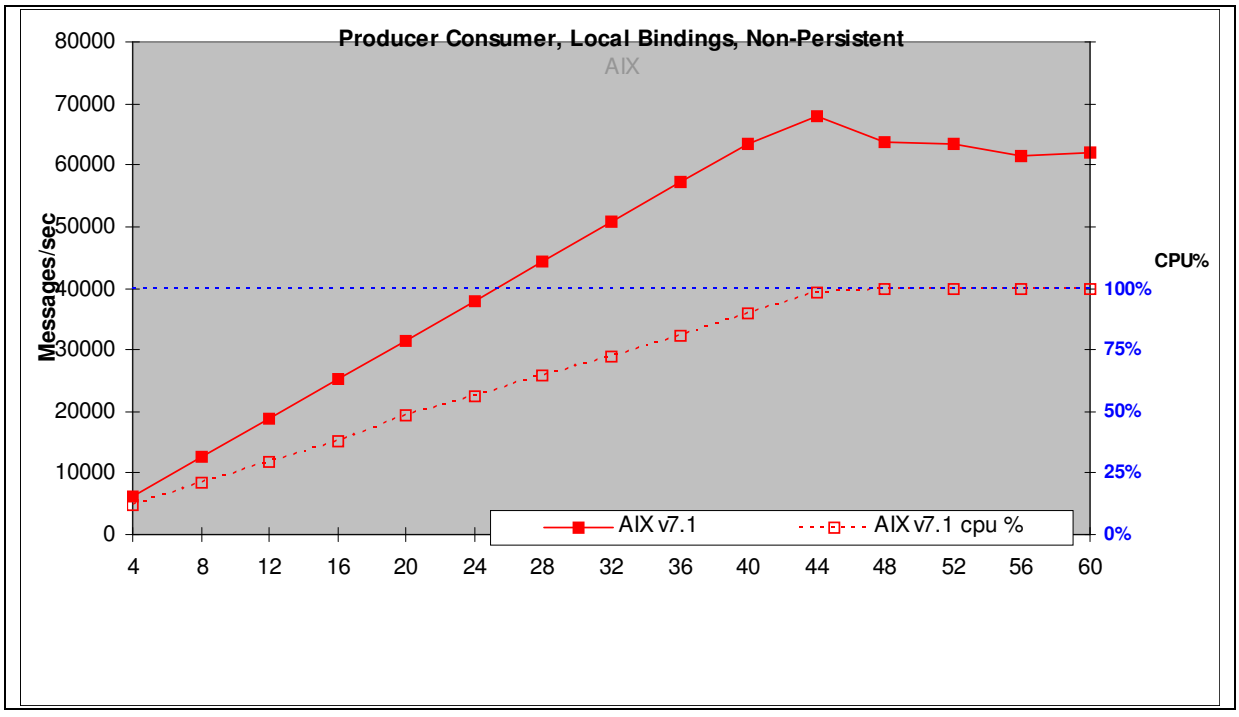

Figure 10 – Producer/Consumer, non persistent messages, local queue manager, AIX

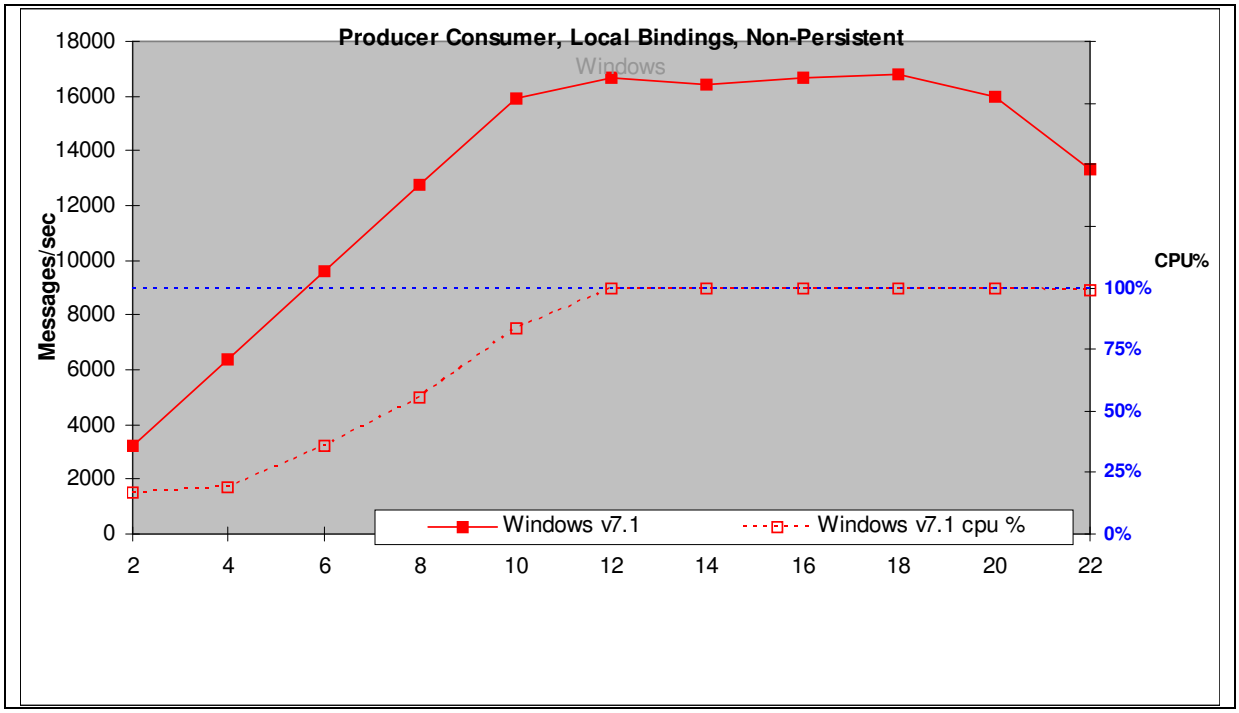

Figure 11 – Producer/Consumer, non persistent messages, local queue manager, Windows

Test name: **PCLN**	**Apps**	**Messages Per second**	**CPU**
Linux64	**144**	**226143**	**94%**
AIX	**44**	**67839**	**98%**
Windows	**12**	**16683**	**99%**

Table 3 – Producer/Consumer, non-persistent messages, local queue manager

Each message producer produces 1600 non-persistent messages per second and the system throughput increases as a straight diagonal line until the system capacity is reached. With 72 producers and 72 consumers (144 Applications) on Linux64, the expected throughput is 1600*72*2=230,400 whereas the measured throughput is 226,143 messages per second.

2.2.2 Producer Consumer, Persistent Messages, Local

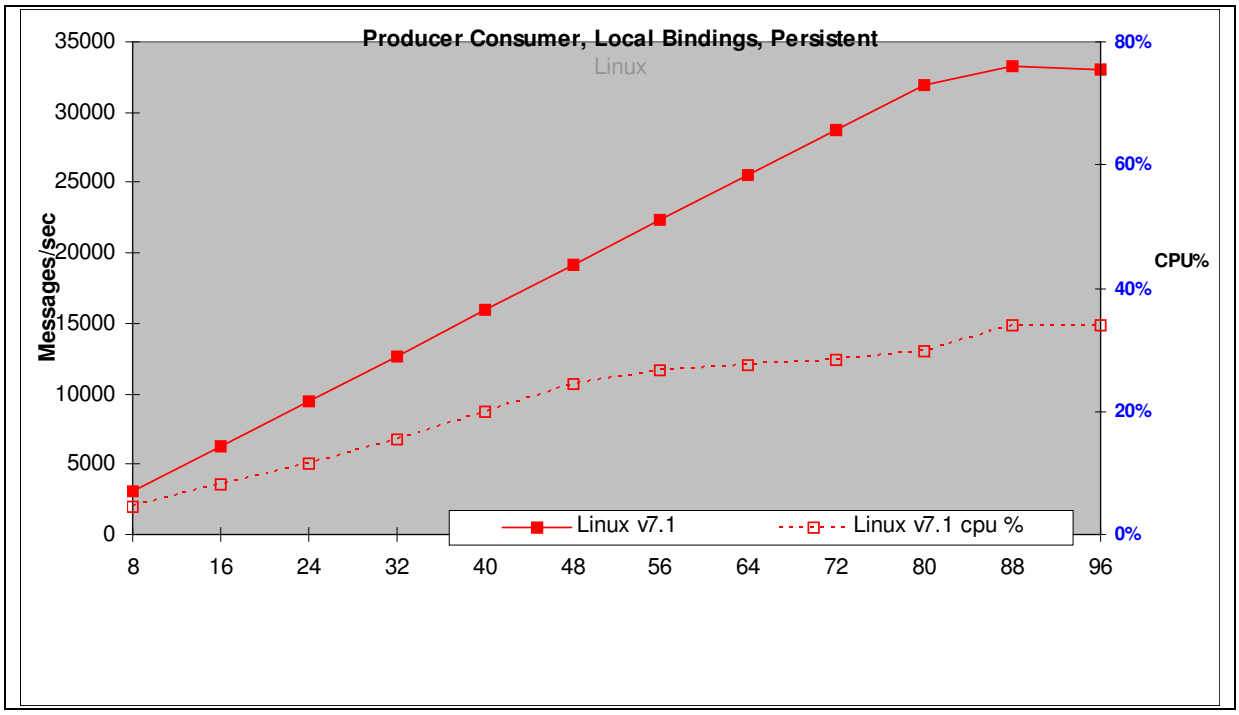

Figure 12 – Producer/Consumer, persistent messages, local queue manager, Linux64

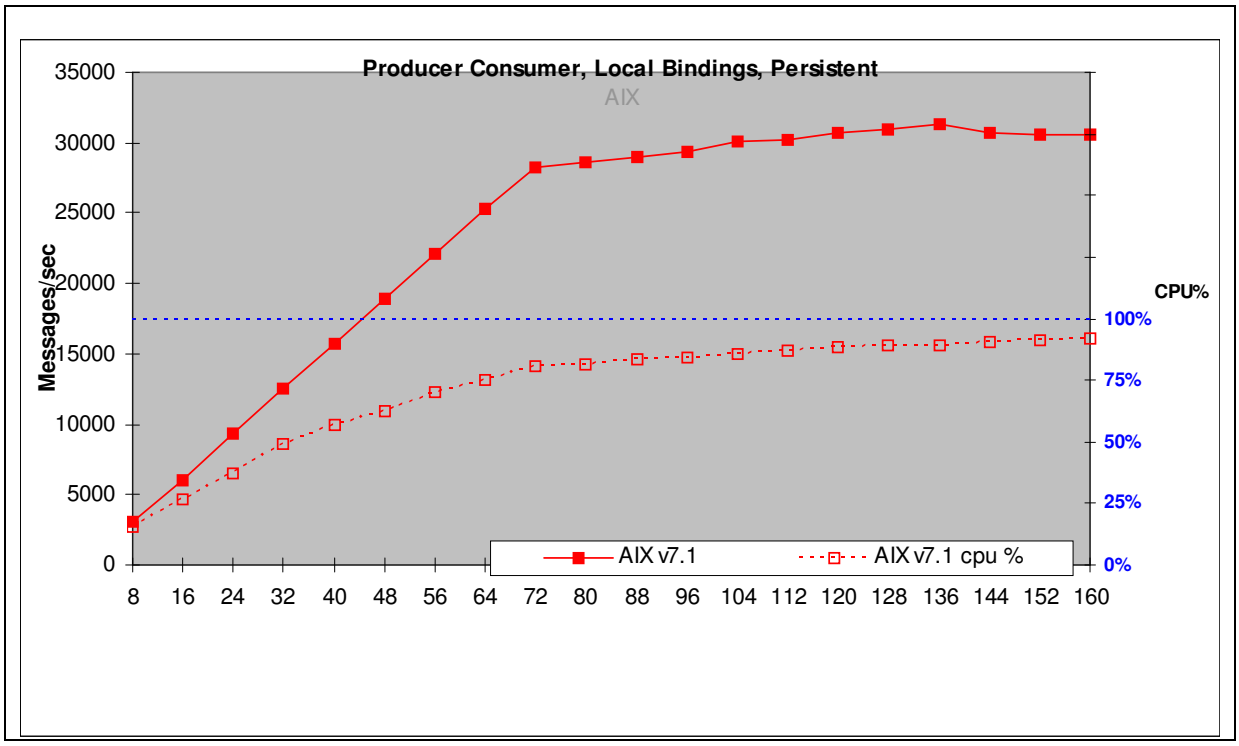

Figure 13 – Producer/Consumer, persistent messages, local queue manager, AIX

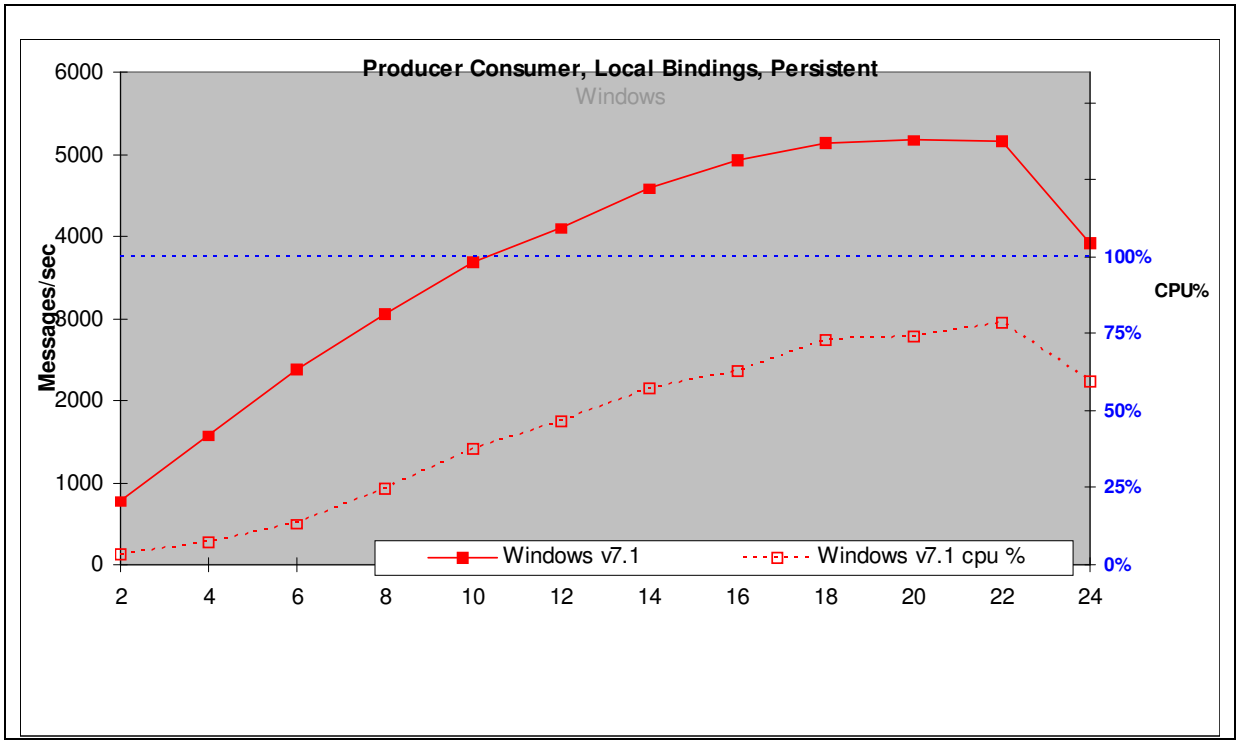

Figure 14 – Producer/Consumer, persistent messages, local queue manager, Windows

Test name: **PCLP**	**Apps**	**Messages Per second**	**CPU**
Linux64	**88**	**33286**	**34%**
AIX	**136**	**31270**	**89%**
Windows	**22**	**5162**	**79%**

Table 4 – Producer/Consumer, persistent messages, local queue manager

Each message producer creates 400 persistent messages per second and the system throughput increases as a straight diagonal line until the system capacity is reached. With 44 producers and 44 consumers (88 Applications) on Linux64, the expected throughput is 400*44*2=35200 whereas the measured throughput is 33286 messages per second.

2.3 Publish/Subscribe Single Publisher, Many Subscribers Scenario(1:N)

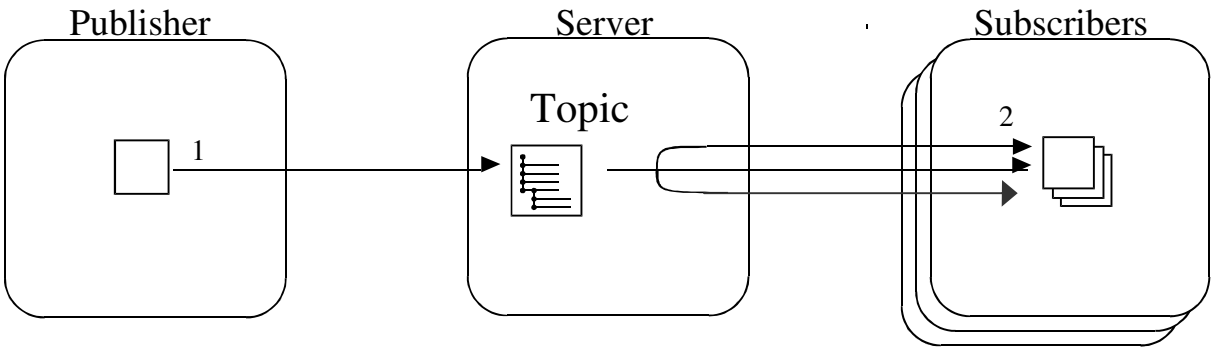

Figure 15 – Publish Subscribe 1:N

All subscribers used unique, managed subscriber queues. A single publisher publishes a message to the topic. Each subscriber then receives the message. This scenario uses asynchronous messaging; hence, there is no connection between the number of messages in the system and the number of publishers or subscribers. The Publisher publishes the next message without any 'think' time. Message count is the number of published messages plus those consumed by the subscribers.

2.3.1 Publish Subscribe 1:N, Non Persistent messages, local

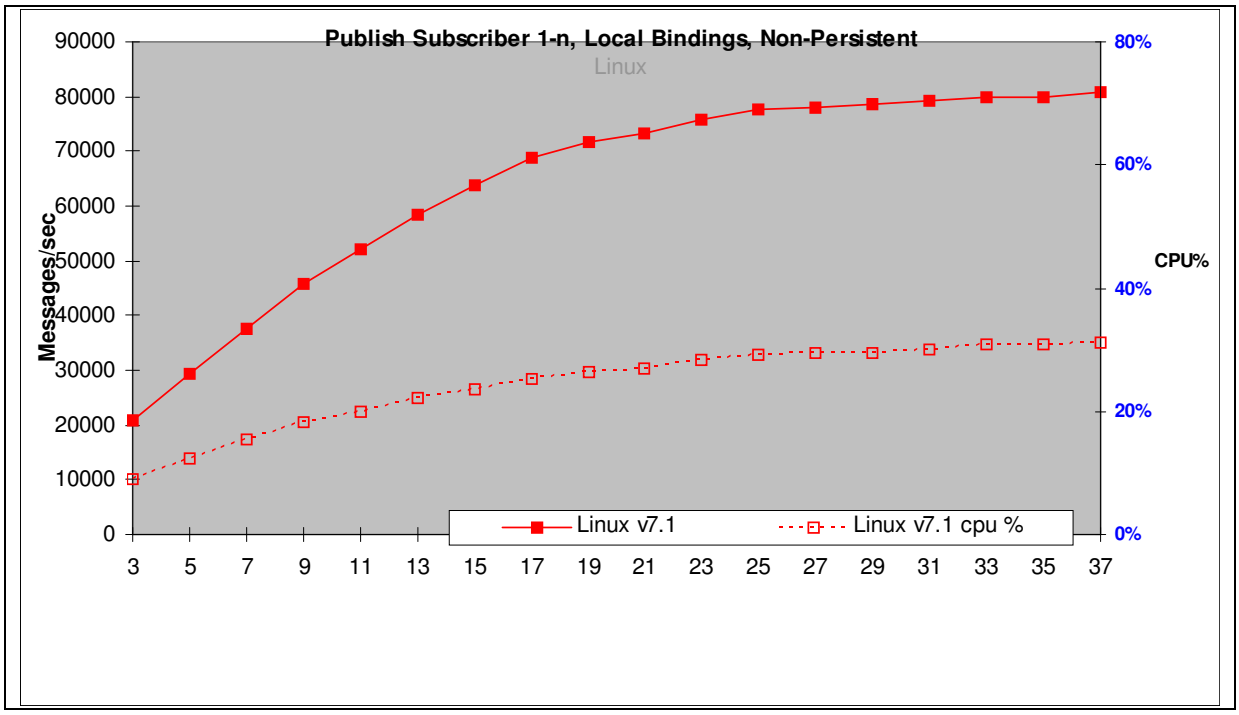

Figure 16 – Pub/Sub 1:N, non-persistent messages, local queue manager, Linux64

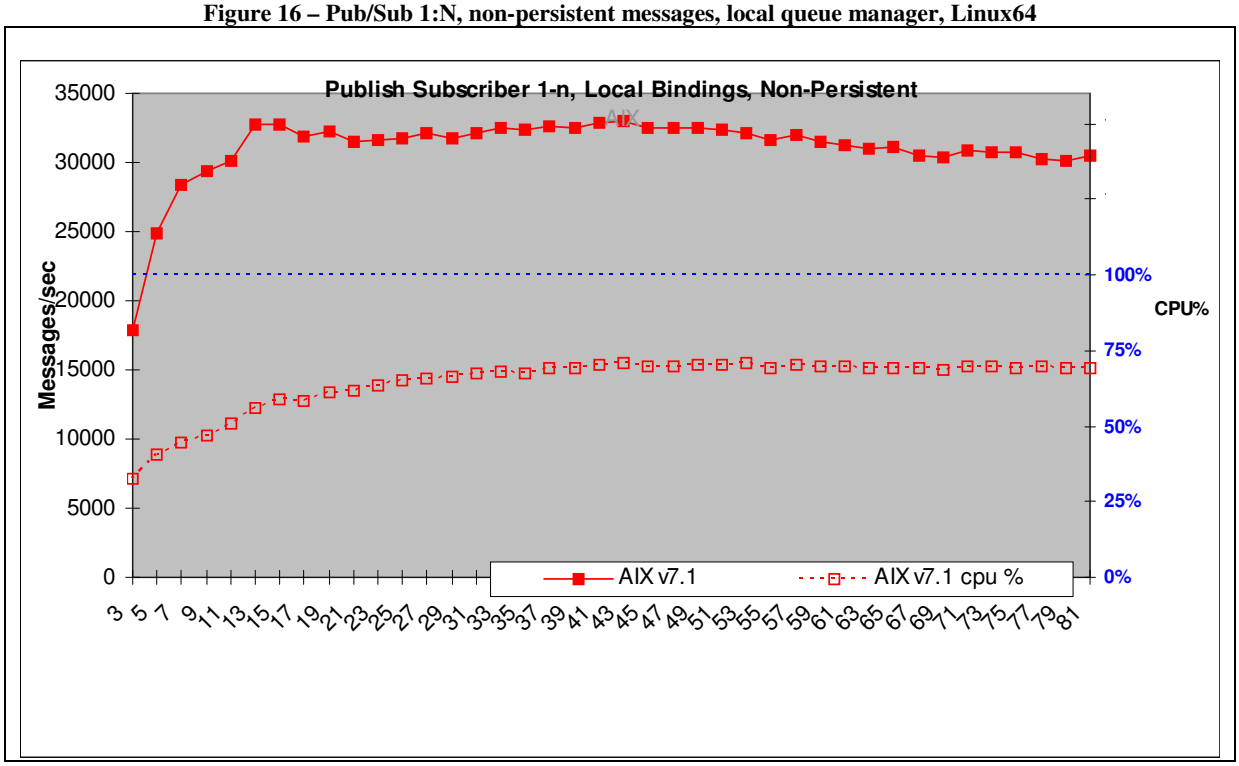

Figure 17 – Pub/Sub 1:N, non-persistent messages, local queue manager, AIX

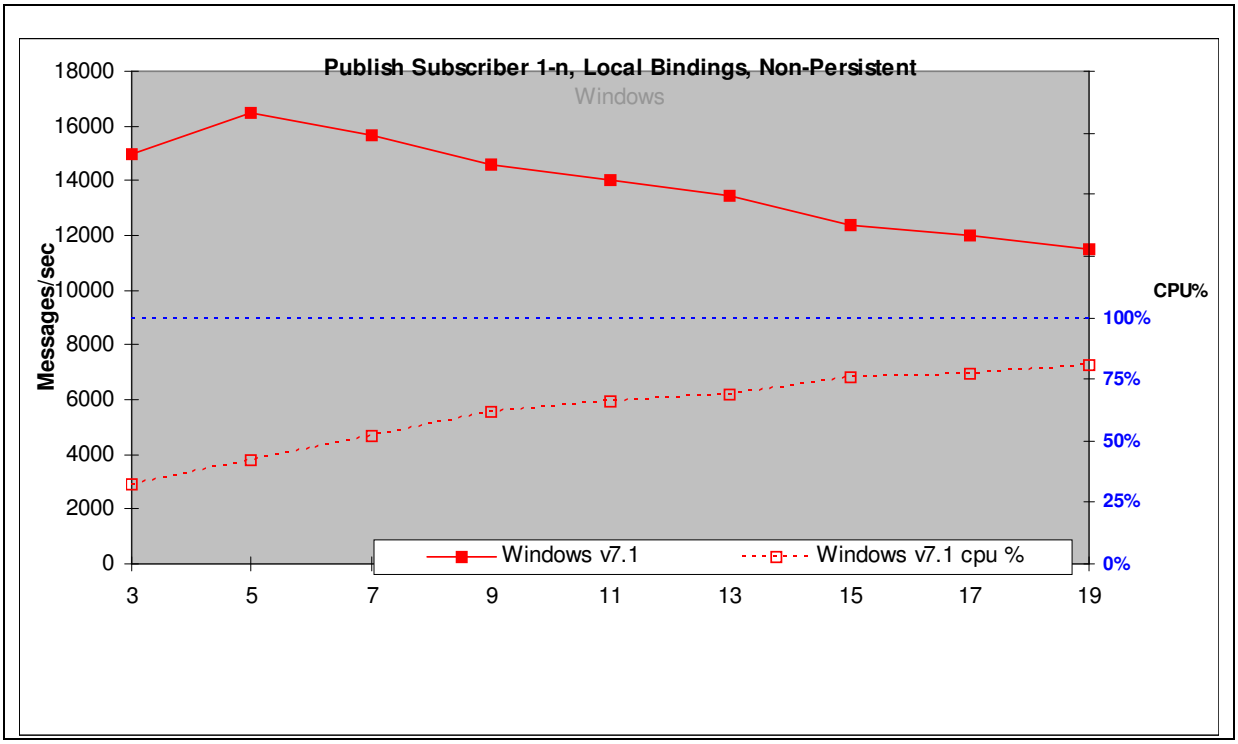

Figure 18 – Pub/Sub 1:N, non-persistent messages, local queue manager, Windows

Test name: **PS1NLN**	Apps	Messages Per second	Publications per second	CPU	Pubs per second With 2 subscribers Per publication
Linux64	**37**	**80787**	**2183**	**31%**	**6921**
AIX	**43**	**32998**	**767**	**43%**	**5955**
Windows	**5**	**16492**	**3298**	**42%**	**4991**

Table 5 – Publish Subscribe 1:N, non-Persistent messages, local queue manager

The publisher produces messages as fast as possible. Initially there are 2 subscribers and one publisher when 6921 publications per second can be achieved on Linux64. The response time for the publish command increases as the number of subscribers increase hence the system message rate plateaus above 36 subscribers. On Linux64 with 36 subscribers, the publisher creates 2183 messages per second, which are all consumed by the subscribers.

2.3.2 Publish Subscribe 1:N, Persistent messages, local

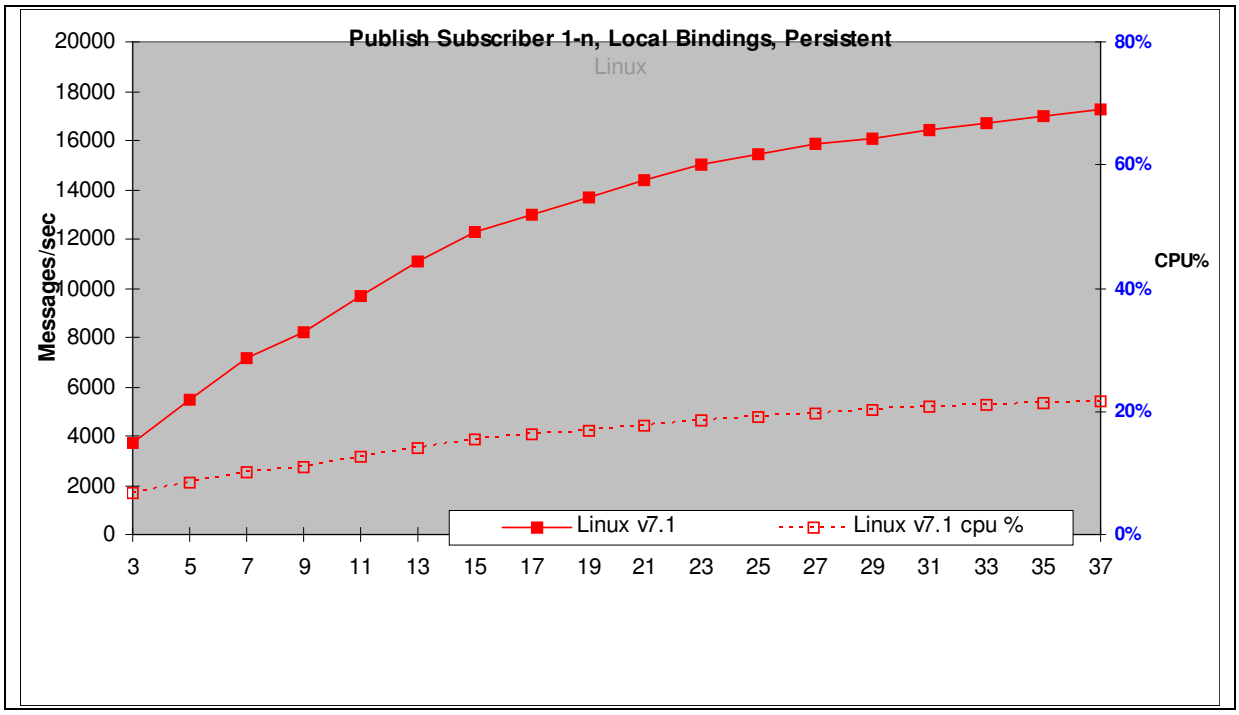

Figure 19 – Pub/Sub 1:N, Persistent messages, local queue manager, Linux64

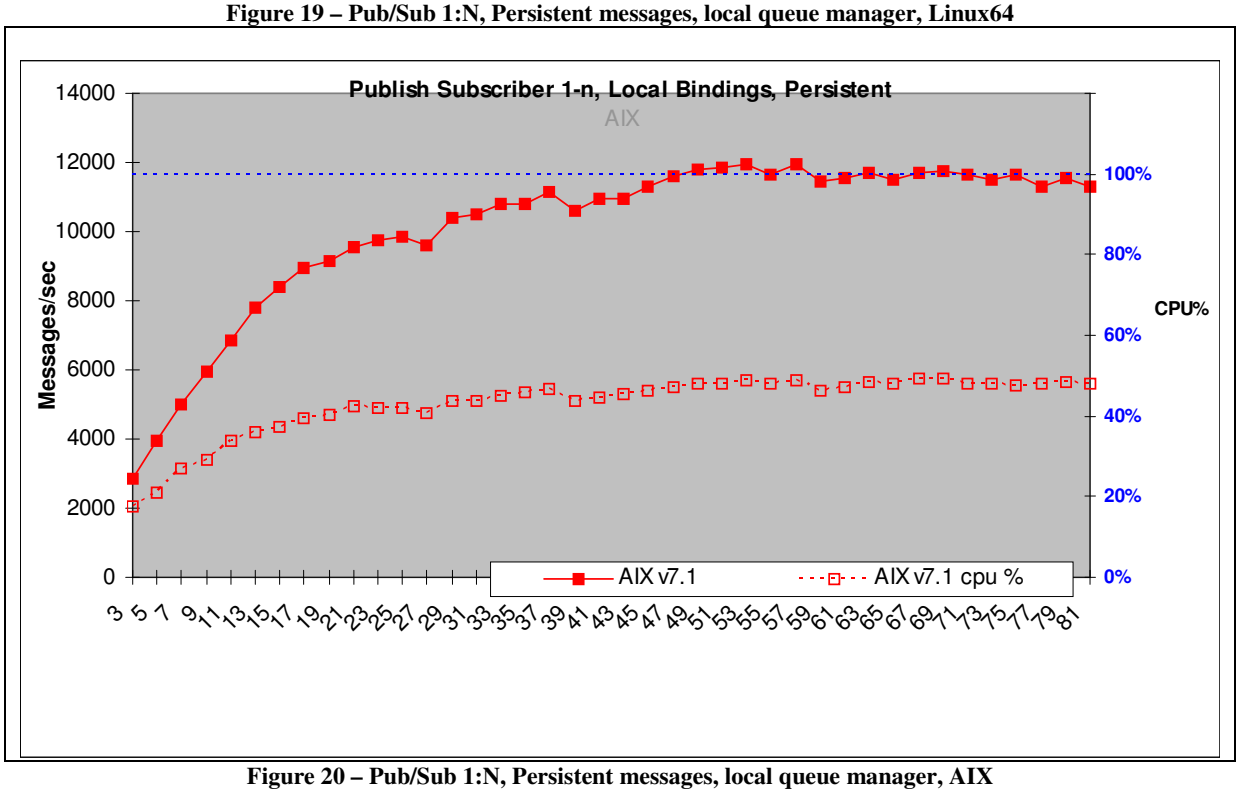

Figure 20 – Pub/Sub 1:N, Persistent messages, local queue manager, AIX

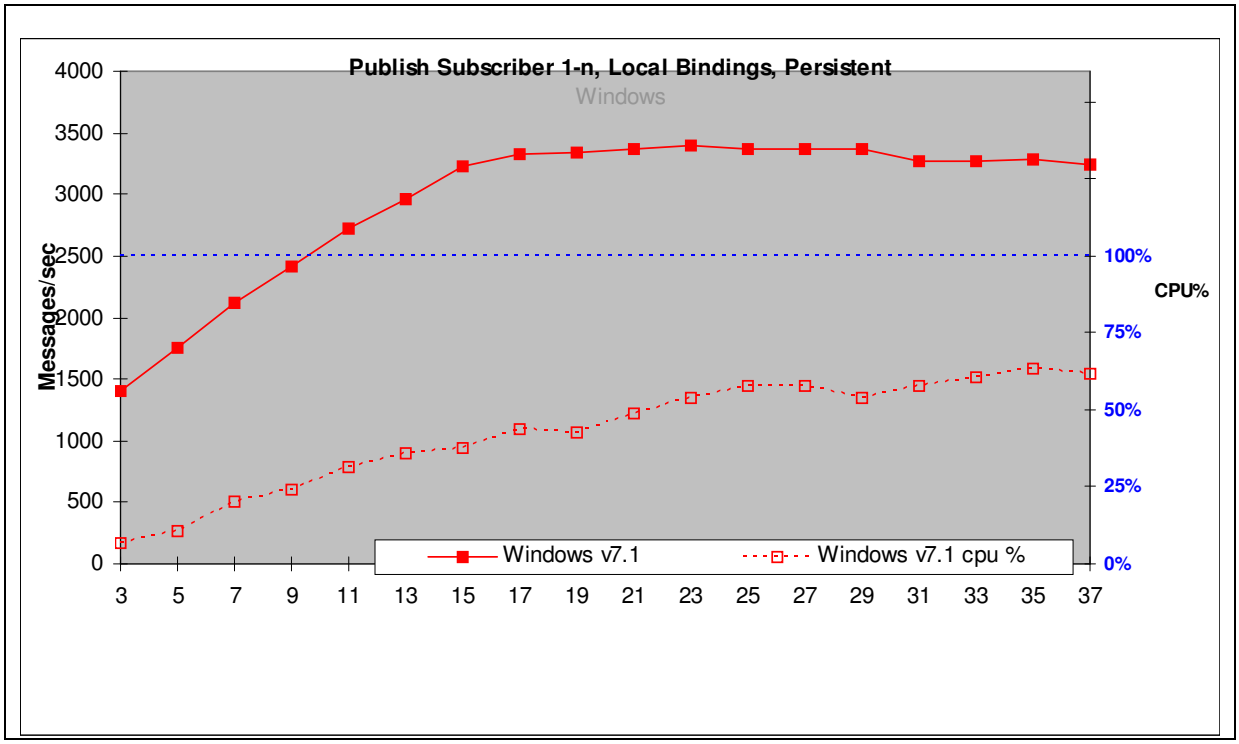

Figure 21 – Pub/Sub 1:N, Persistent messages, local queue manager, Windows

Test name: **PS1NLP**	Apps	Messages Per second	Publications Per second	CPU	Pubs per second With 2 subscribers Per publication
Linux64	**37**	**17285**	**467**	**21%**	**1247**
AIX	**53**	**11968**	**226**	**49%**	**952**
Windows	**23**	**3391**	**147**	**54%**	**468**

Table 6 – Publish Subscribe 1:N, Persistent messages, local queue manager

The publisher produces messages as fast as possible. Initially there are 2 subscribers and one publisher when 1247 publications per second can be achieved on Linux64. The response time for the publish command increases as the number of subscribers increase hence the system message rate plateaus above 36 subscribers. On Linux64 with 36 subscribers, the publisher creates 467 messages per second, which are all consumed by the subscribers.

2.4 Publish Subscribe multiple (Publisher, Topic, Subscriber) scenario

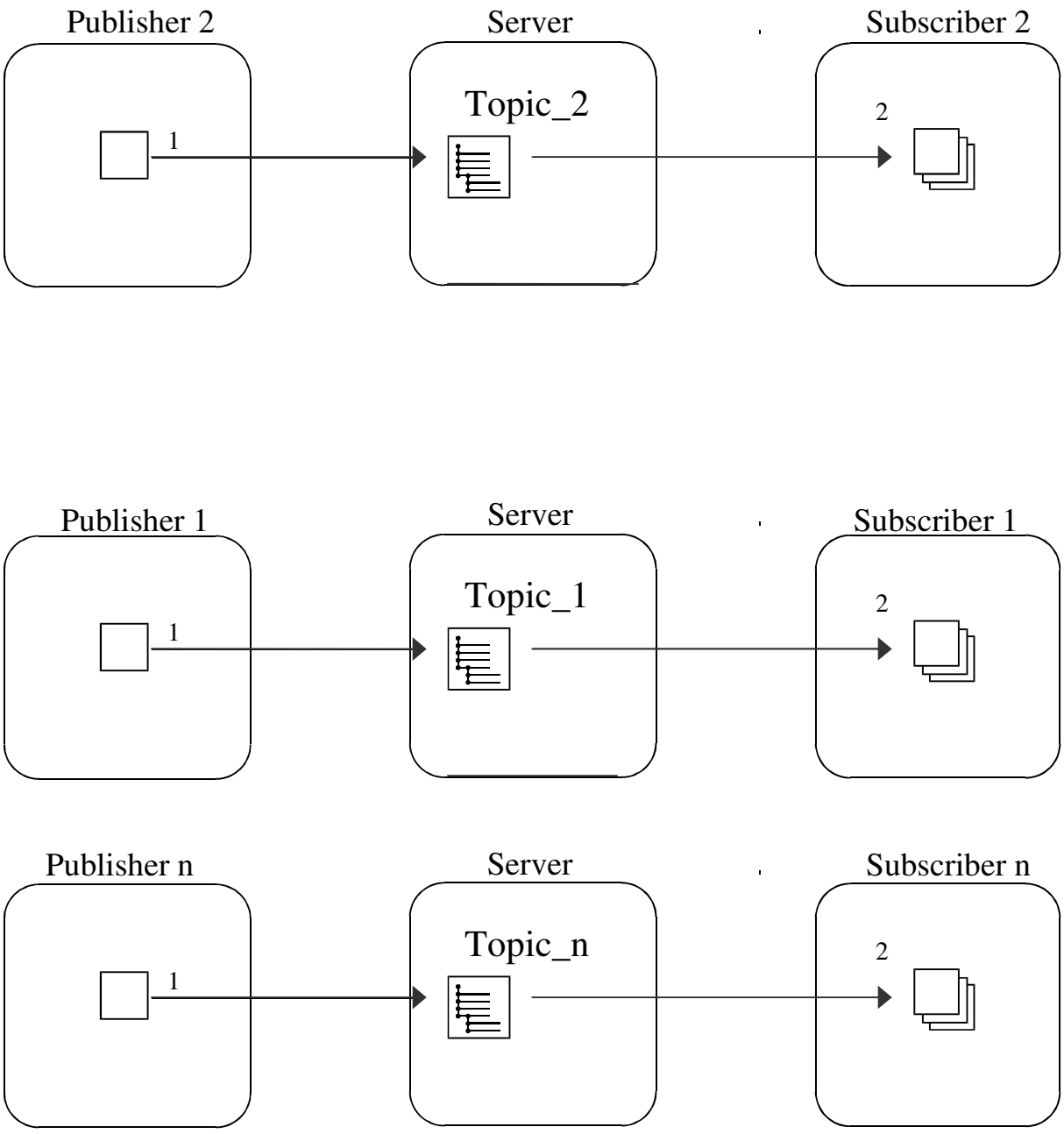

Figure 22 – Publish Subscribe

All subscribers used unique subscriber queues. Each publisher publishes a message to a specific topic. Only the single subscriber registered for that topic receives the message.
This scenario uses asynchronous messaging; hence, there is no connection between the number of messages in the system and the number of publishers or subscribers. The publisher publishes message at a predetermined rate which results in a gradually increasing workload as the number of (Publisher, Topic, Subscriber) triplets is increased. The message production rate is 1600 per second for non-persistent and 400 per second for persistent messages. Message count is the number of published messages plus those consumed by the subscribers.

2.4.1 Publish Subscribe (Multiple P/T/S), Non Persistent messages, local

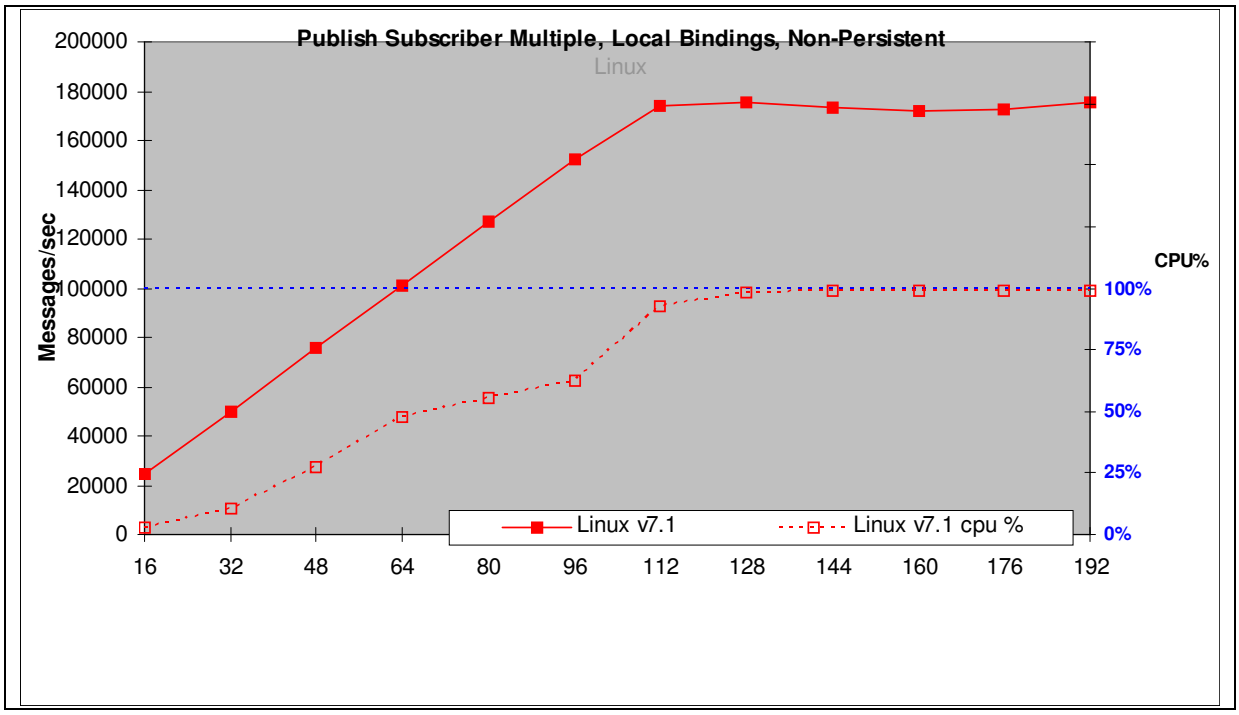

Figure 23 – Publish Subscribe Multiple, Non-persistent messages, local, Linux64

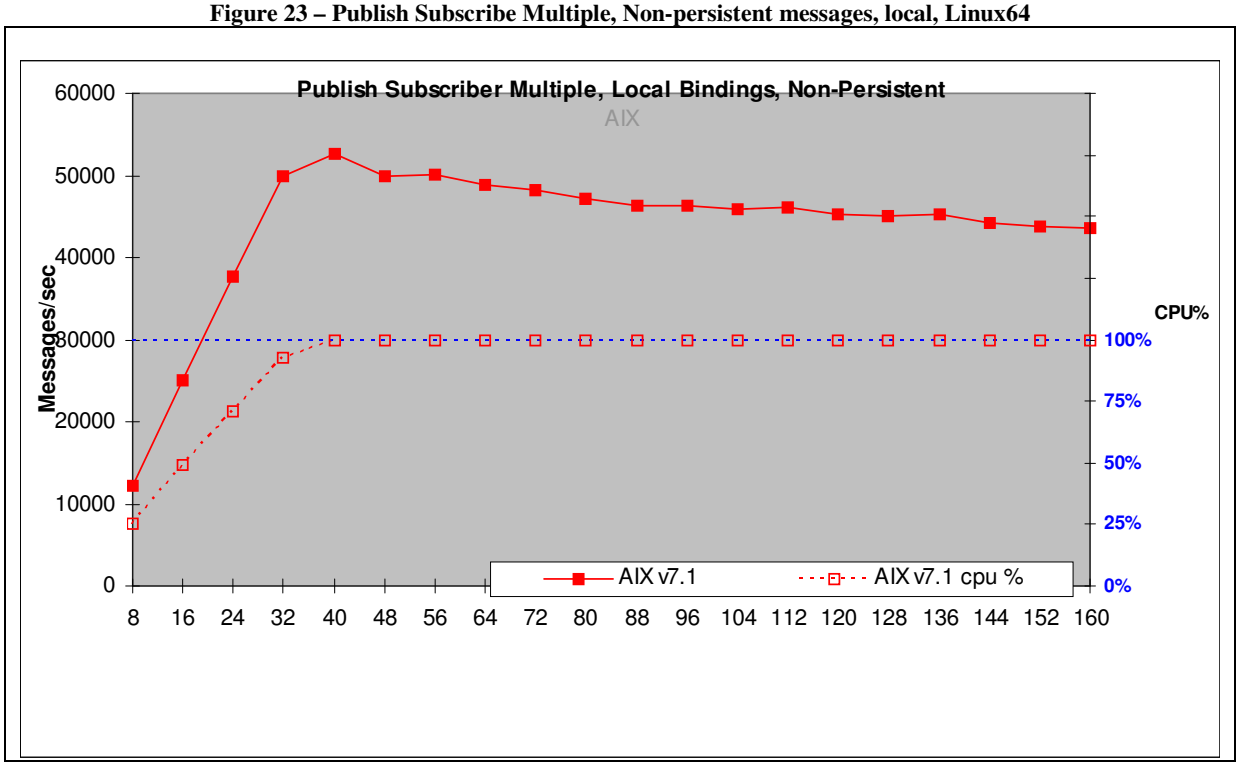

Figure 24 – Publish Subscribe Multiple, Non persistent messages, local, AIX

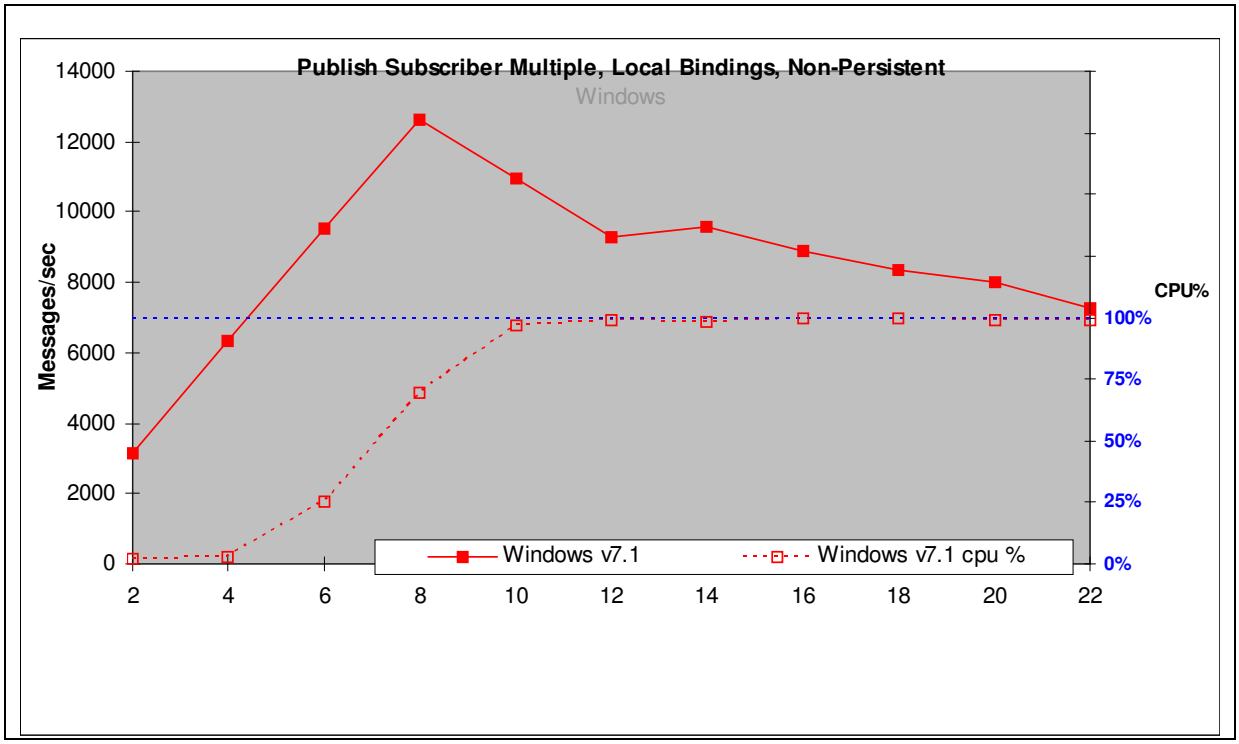

Figure 25 – Publish Subscribe Multiple, Non-persistent messages, local, Windows

Test name: **PSMLN**	**Apps**	**Messages Per second**	**CPU**
Linux64	**112**	**174099**	**92%**
AIX	**40**	**52608**	**99%**
Windows	**8**	**12627**	**69%**

Table 7 – Publish Subscribe Multiple, non Persistent messages, local queue manager

Each publisher creates 1600 non-persistent messages per second and the system throughput increases as a straight diagonal line until the system capacity is reached. With 56 producers and 56 consumers (112 Applications) on Linux64, the expected throughput is 1600*64*2=179200 whereas the measured throughput is 174099 messages per second.

2.4.2 Publish Subscribe (Multiple P/T/S), Persistent messages, local

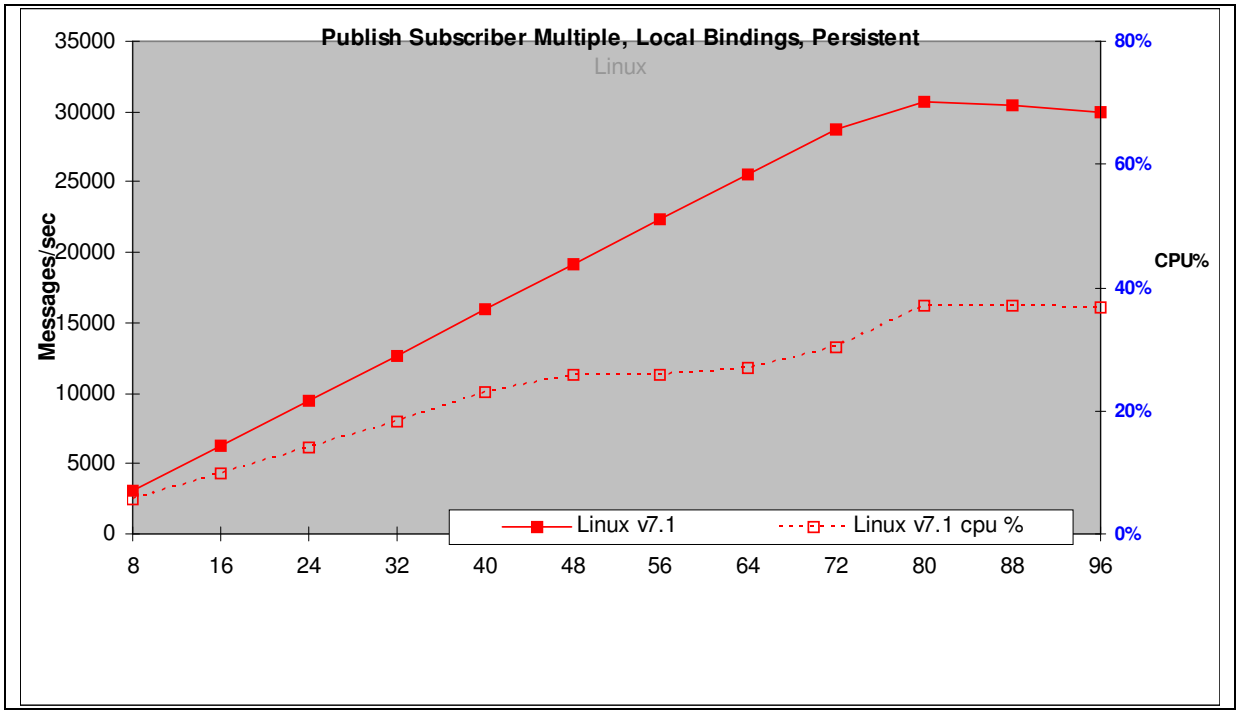

Figure 26 – Publish Subscribe Multiple, Persistent messages, local, Linux64

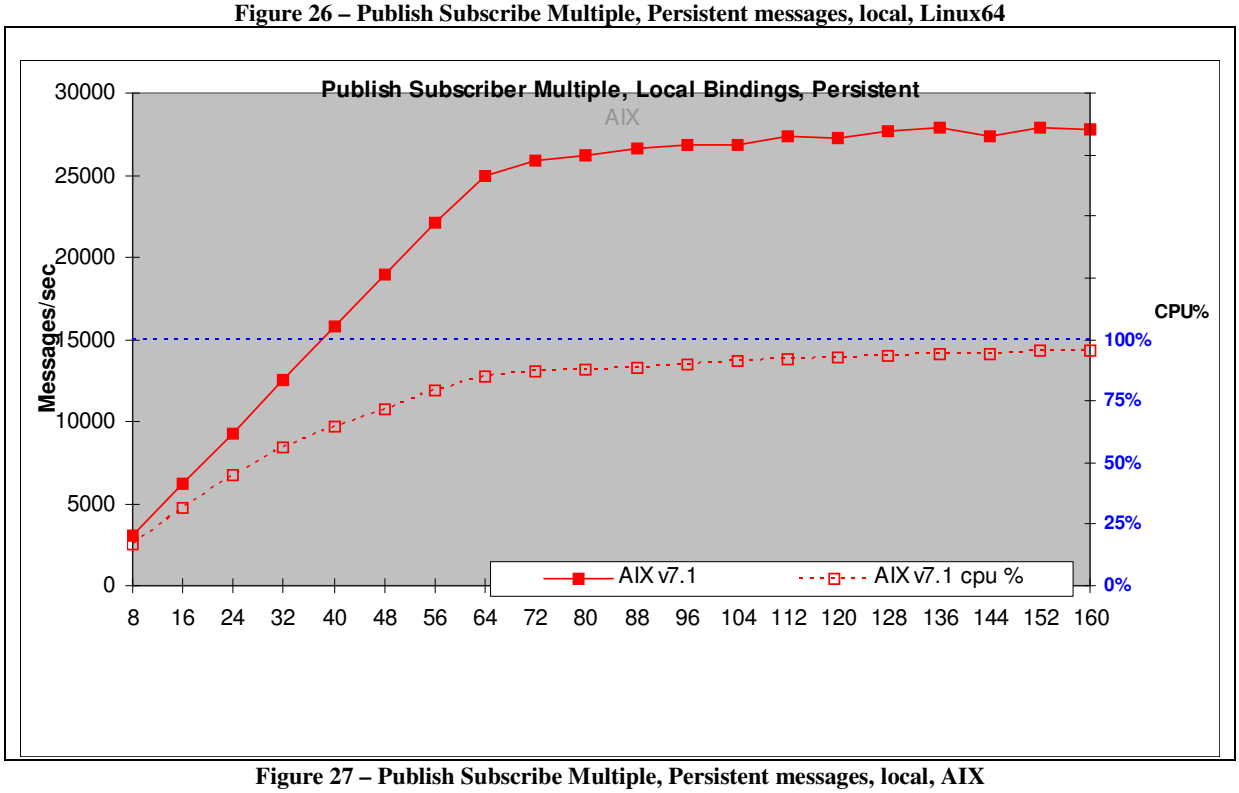

Figure 27 – Publish Subscribe Multiple, Persistent messages, local, AIX

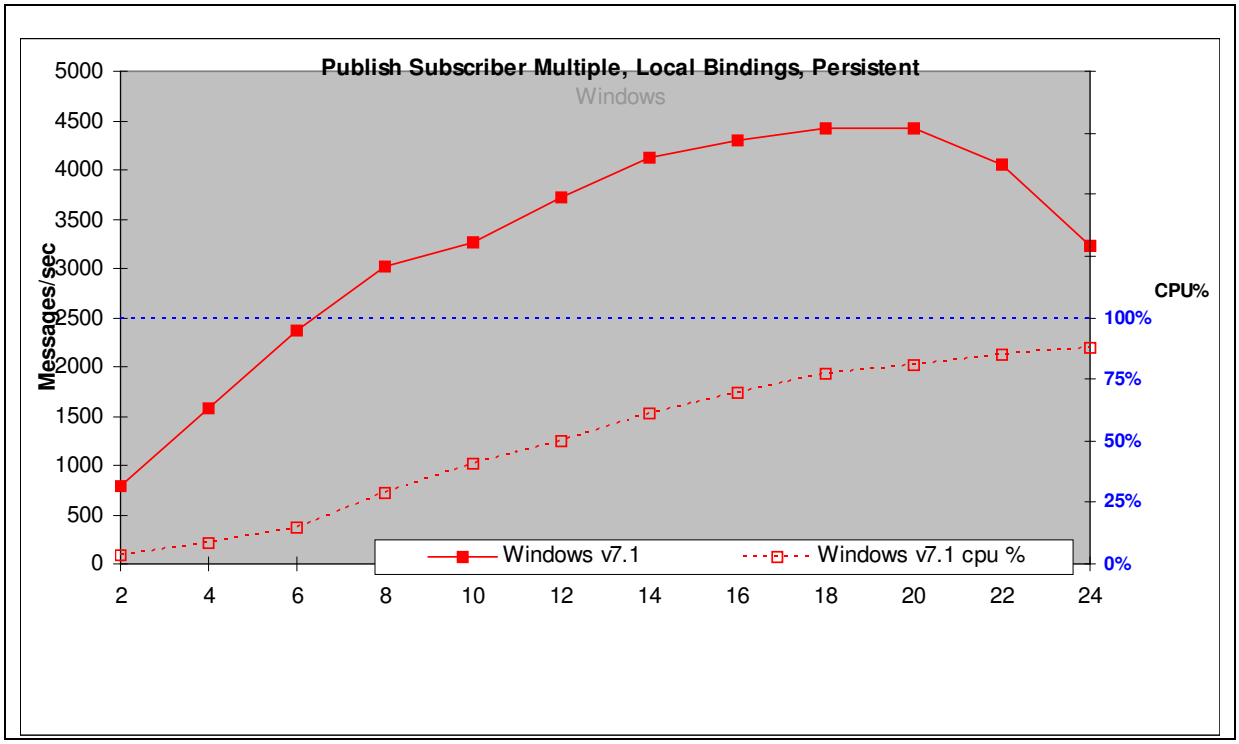

Figure 28 – Publish Subscribe Multiple, Persistent messages, local, Windows

Test name: **PSMLP**	**Apps**	**Messages Per second**	**CPU**
Linux64	**80**	**30648**	**37%**
AIX	**136**	**27866**	**94%**
Windows	**18**	**4422**	**77%**

Table 8 – Publish Subscribe Multiple, Persistent messages, local queue manager

Each message producer creates 400 messages per second and the system throughput increases as a straight diagonal line until the system capacity is reached. With 40 producers and 40 consumers (80 Applications) on Linux64, the expected throughput is 400*40*2=32000 whereas the measured throughput is 30648 messages per second.

3 Client Channels Test Scenario

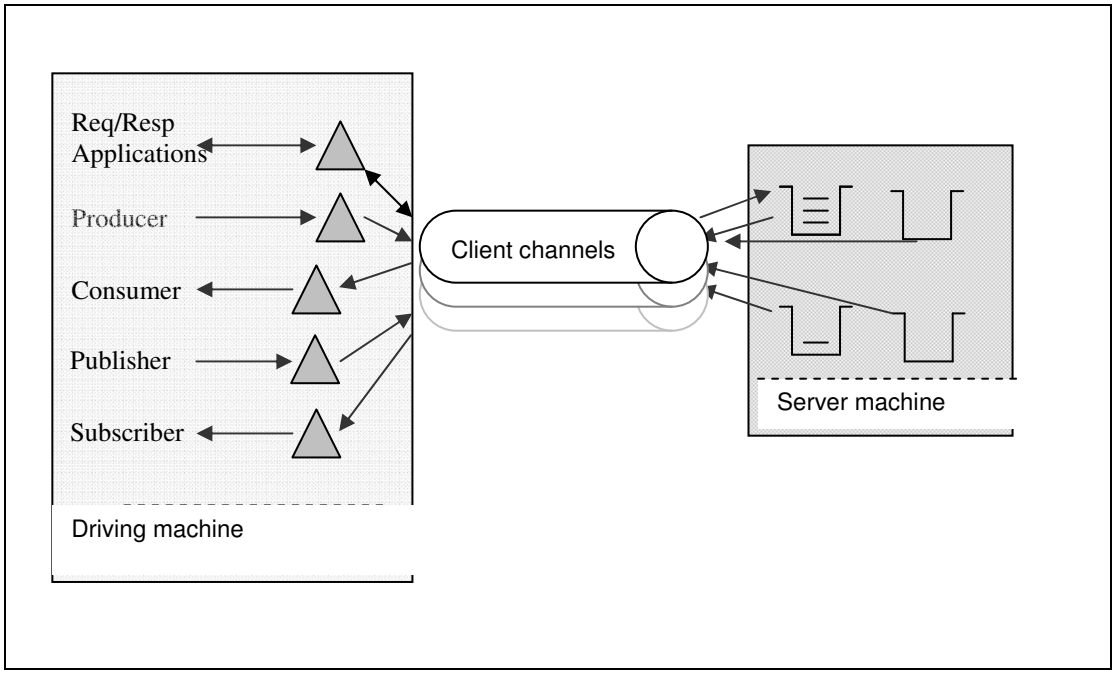

Figure 29 – MQI-client channels into a remote queue manager

The various message producers put a message (over a client channel), to the relevant queue on the server. The consumer application then waits indefinitely for messages to arrive on its input queue.

All of the JMS code is executed on the Client (driver) machine and the individual MQ verbs (Put, Get, Commit are sent to the server to drive the Queue Manager.

The Client Channel is set to 'MQIBindType = FASTPATH' . The major benefit is for non persistent messages because it eliminates the AGENT process (AMQZLAA) and reduces CPU cost. Environments using Channel exits should be aware that the exit code would run inside the Queue Manager.

From version 7.0.0, the default for client connections is to share an MQI channel. Each channel is defined with a default of 10 threads to run up to 10 client conversations per channel instance. In the test scenarios in this report all clients are busy concurrently and hence performance is improved by having a separate channel for each client. This is done by setting the SHARECNV channel attribute to one.

3.1 Requester-Responder Scenario

3.2 Requester-Responder Non-persistent Messages – Client

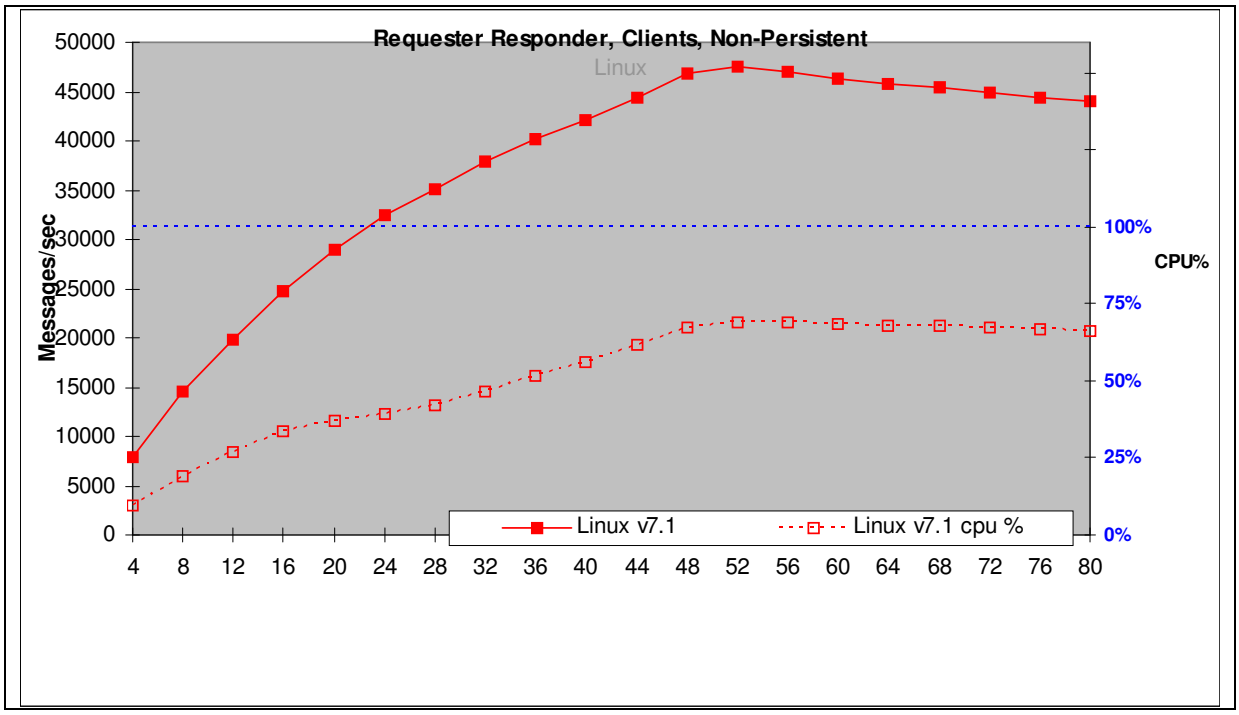

Figure 30 – Requester-Responder, non persistent, client, Linux64

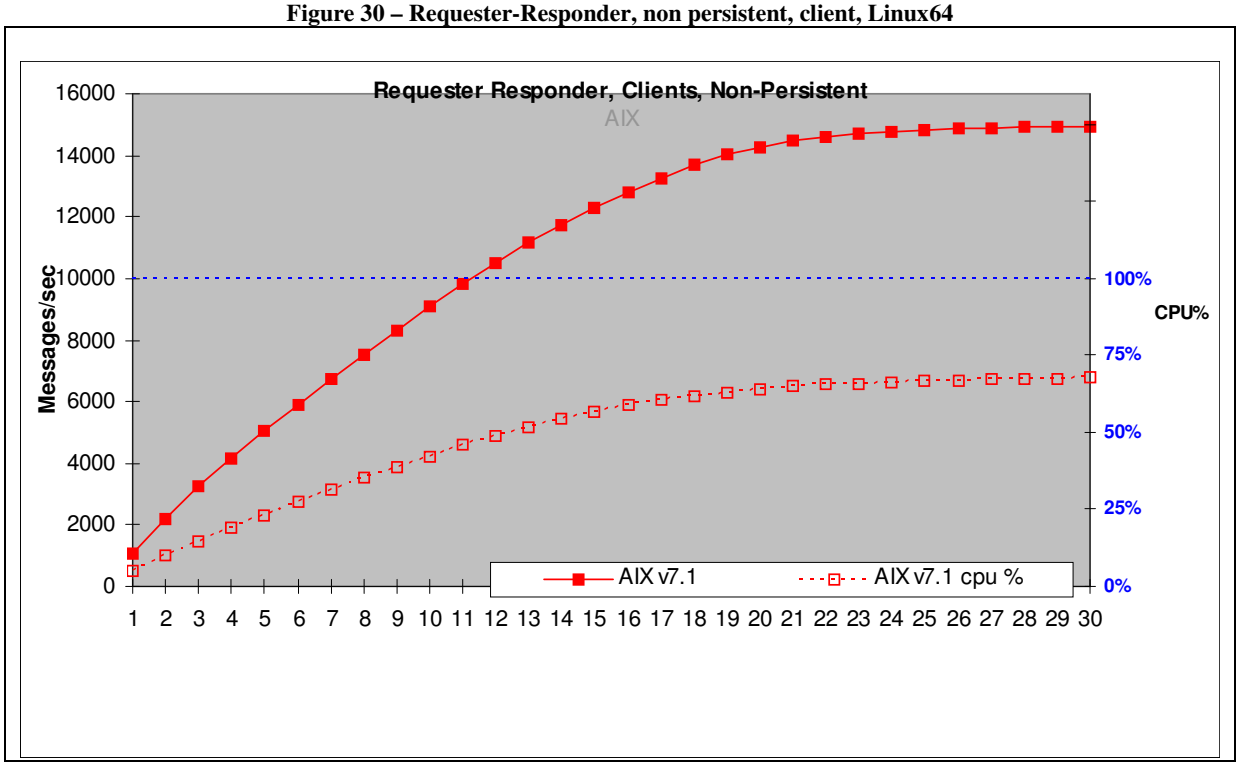

Figure 31 – Requester-Responder, non persistent, client, AIX

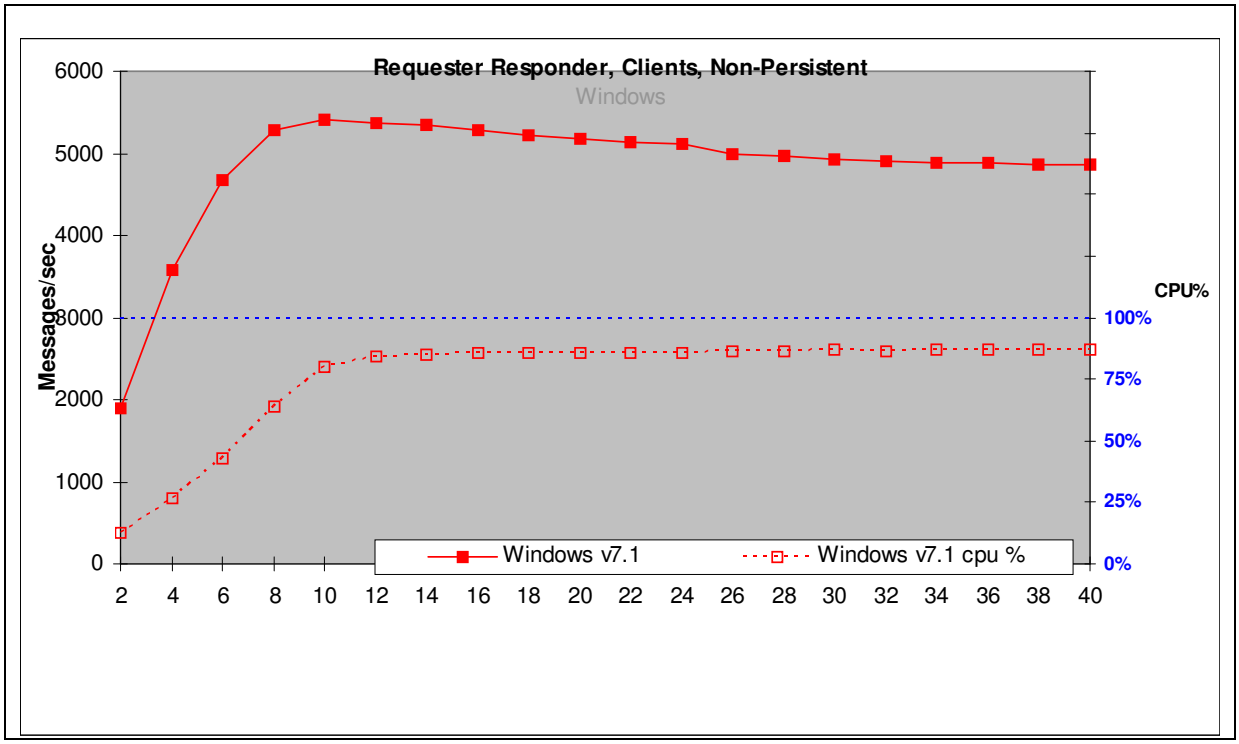

Figure 32 – Requester-Responder, non persistent, client, Windows

Test name: **RRQCN**	**Apps**	**Round Trips/sec**	**CPU**
Linux64	**52**	**47525**	**69%**
AIX	**30**	**14948**	**68%**
Windows	**10**	**5405**	**80%**

Table 9 – Requester-Responder, non-Persistent messages, Client connection

3.2.1 Requester-Responder Persistent Messages – Client

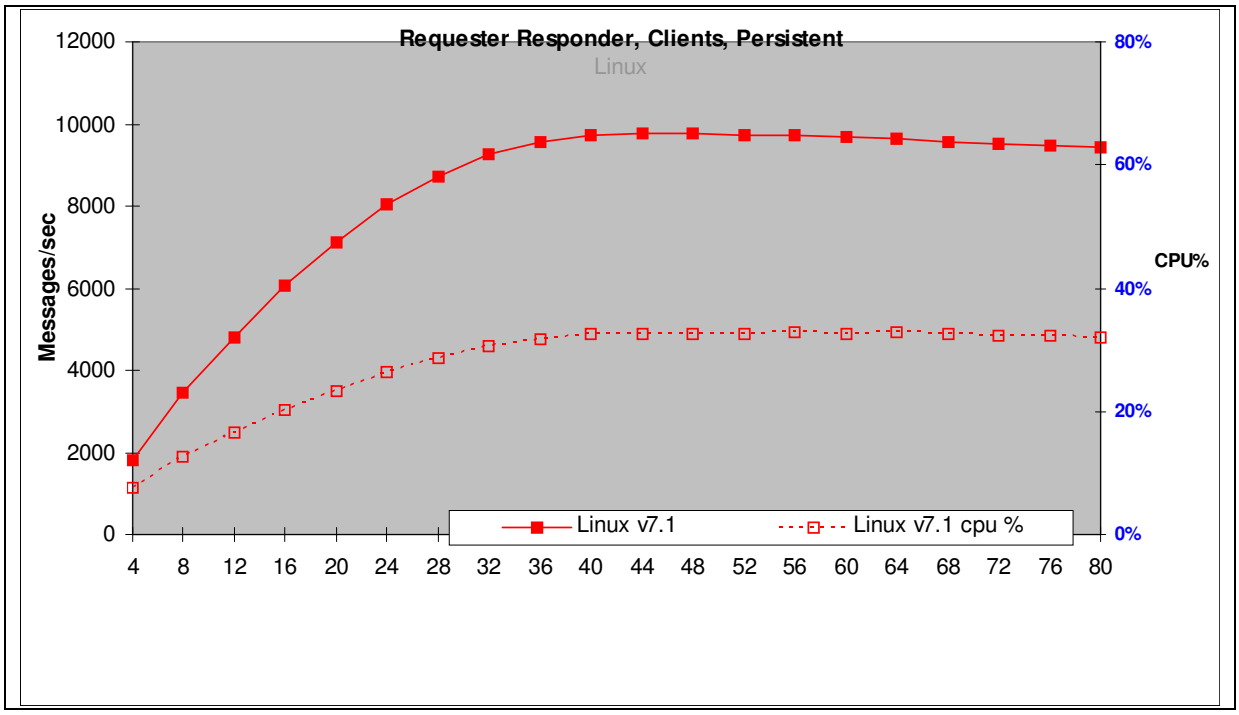

Figure 33 – Requester-Responder, persistent, client, Linux64

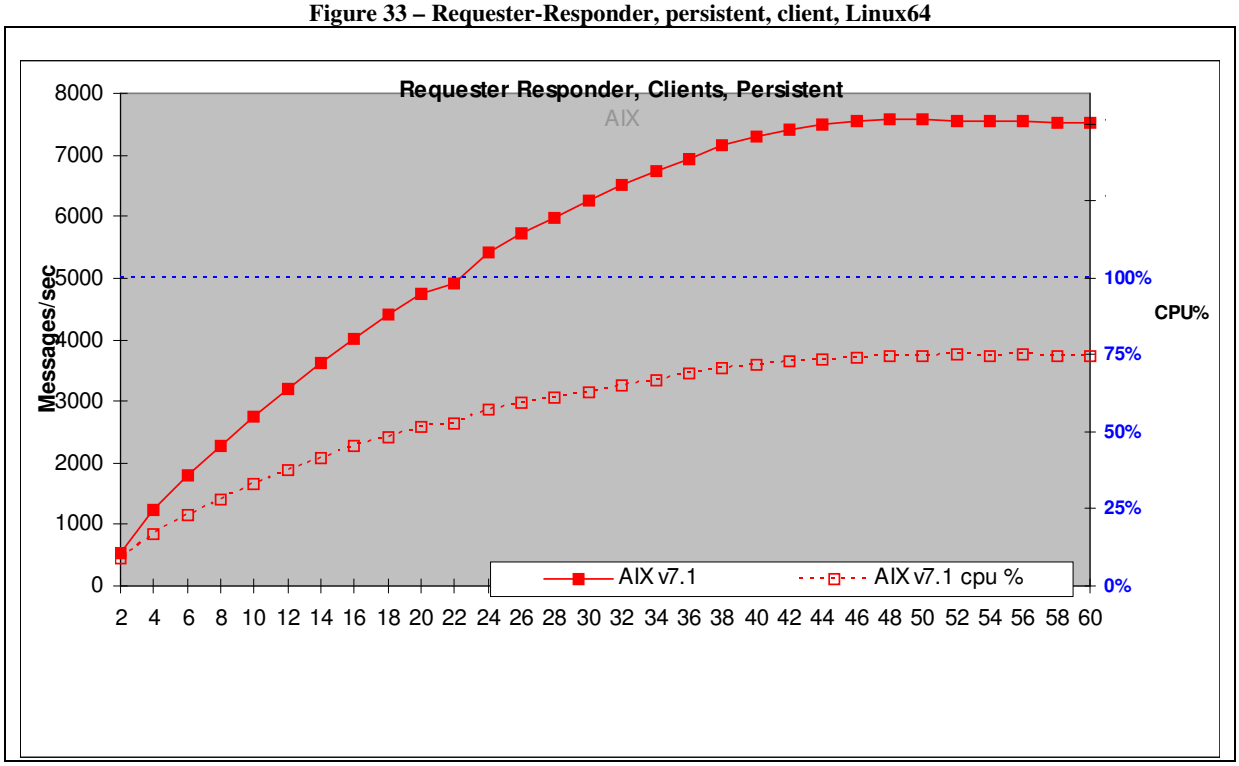

Figure 34 – Requester-Responder, persistent, client, AIX

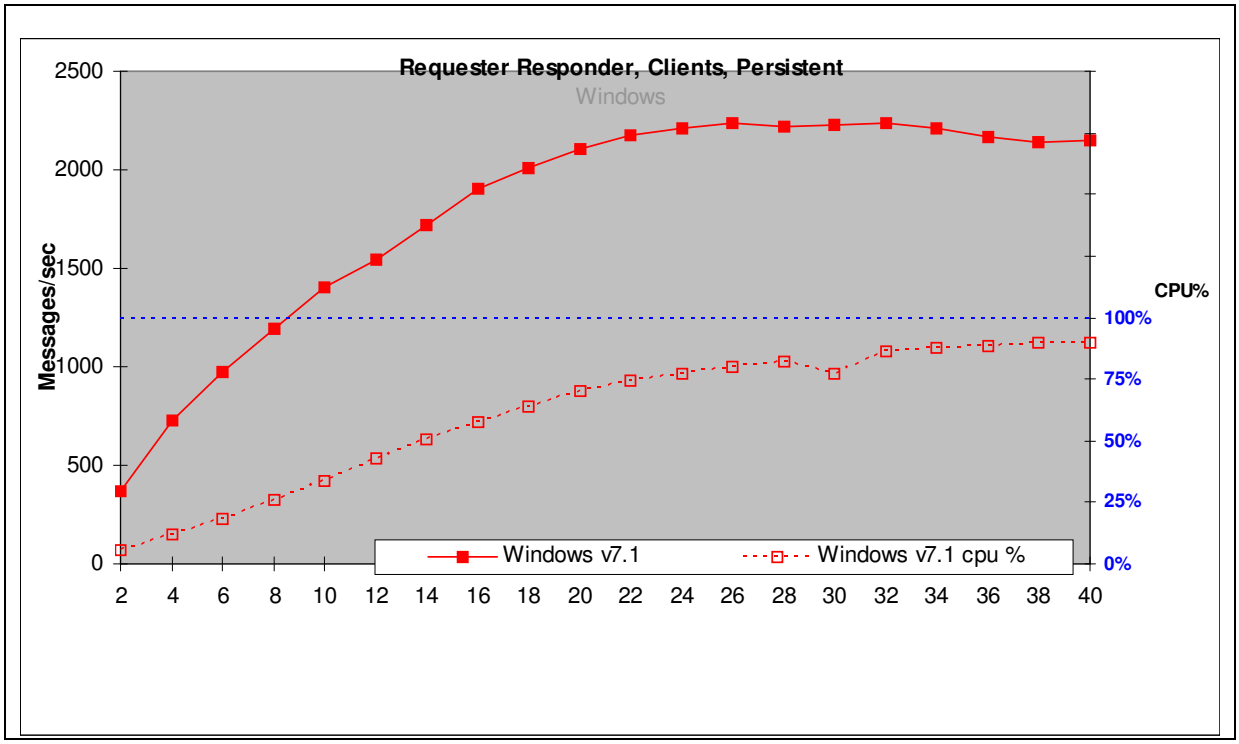

Figure 35 – Requester-Responder, persistent, client, Windows

Test name: **RRQCP**	**Apps**	**Round Trips/sec**	**Server CPU**
Linux64	**44**	**9763**	**33%**
AIX	**48**	**7587**	**75%**
Windows	**26**	**2238**	**80%**

Table 10 – Requester-Responder, Persistent messages, Client connection

.

3.3 Point to Point , Multiple (Producer, Consumer, Queue) Scenario

3.3.1 Producer Consumer, Non Persistent Messages, Client

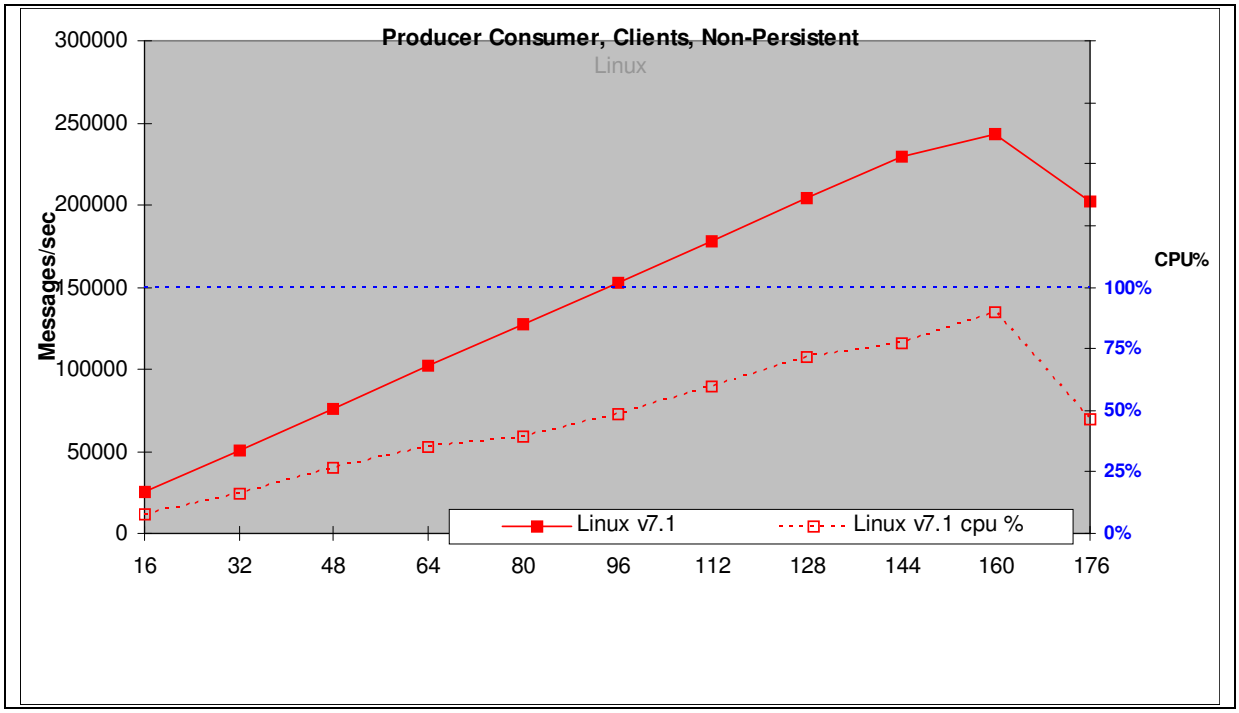

Figure 36 – Producer/Consumer, non persistent, client, Linux64

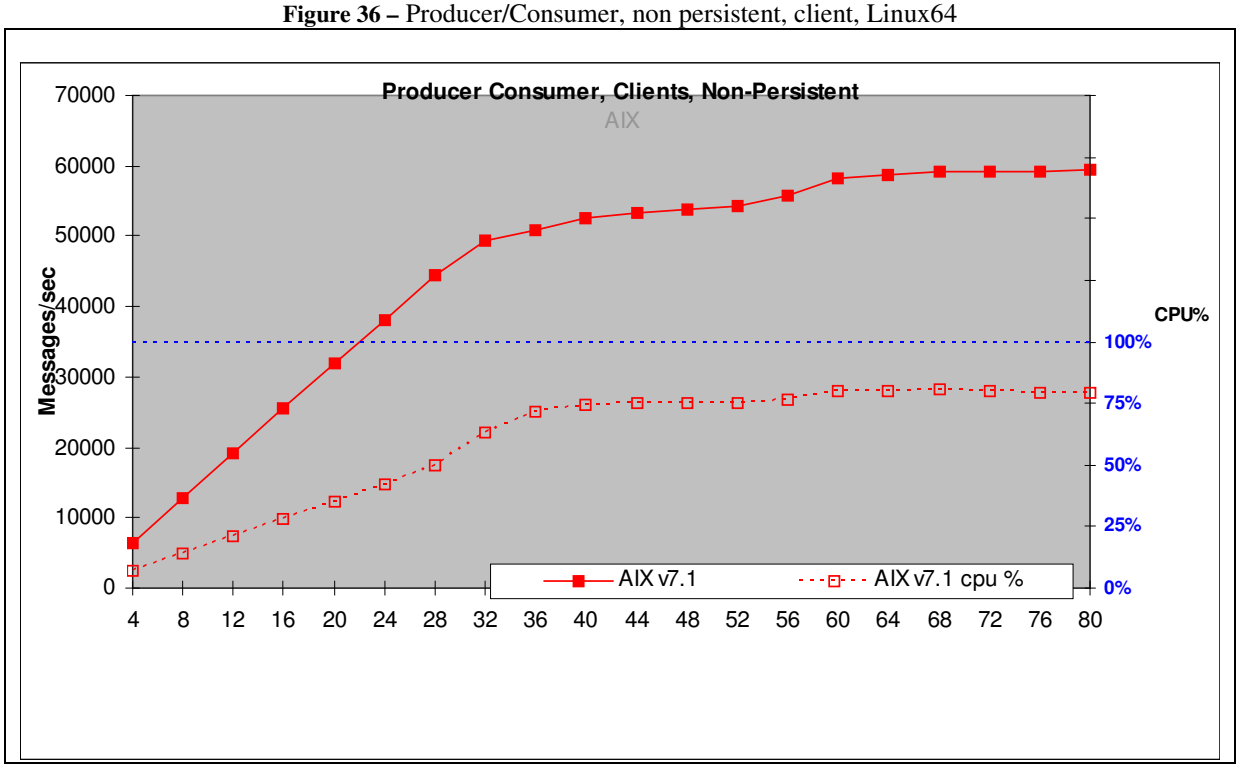

Figure 37 – Producer/Consumer, non persistent, client, AIX

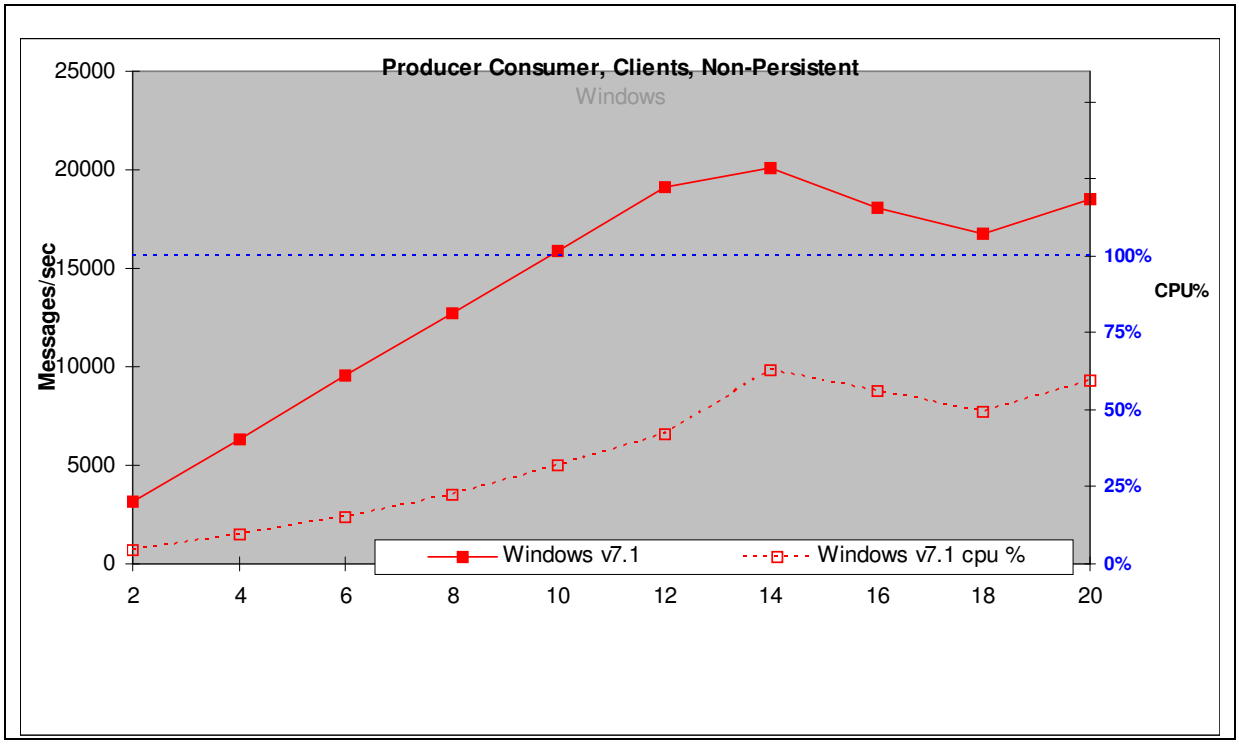

Figure 38 – **Producer/Consumer, non persistent, client**, Windows

Test name: **PCCN**	**Apps**	**Messages Per second**	**Server CPU**
Linux64	**128**	**203995**	**72%**
AIX	**80**	**59445**	**80%**
Windows	**14**	**20061**	**63%**

Table 11 – Producer/Consumer, non Persistent messages, Client connection

Each message producer creates 1600 non-persistent messages per second and the system throughput increases as a straight diagonal line until the system capacity is reached. With 64 producers and 64 consumers (128 Applications) on Linux64, the expected throughput is 1600*64*2=204800 whereas the measured throughput is 203995 messages per second.

3.3.2 Producer Consumer, Persistent Messages, Client

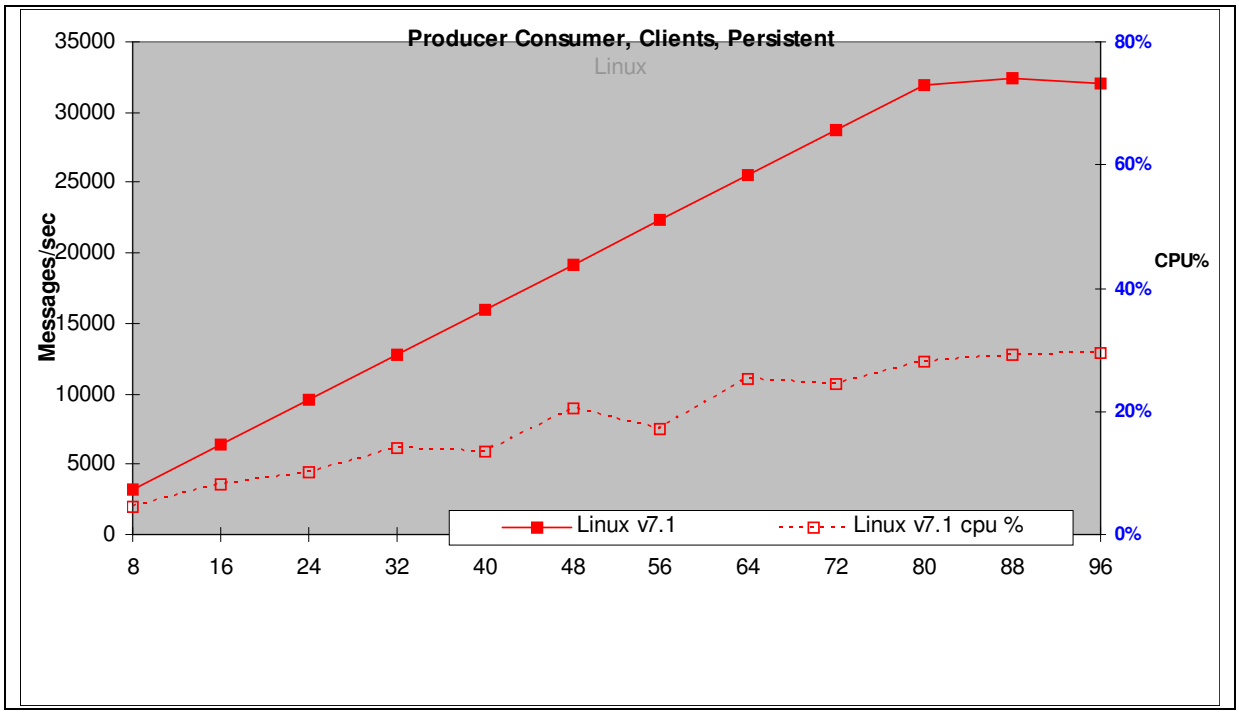

Figure 39 – producer/Consumer, persistent, client, Linux64

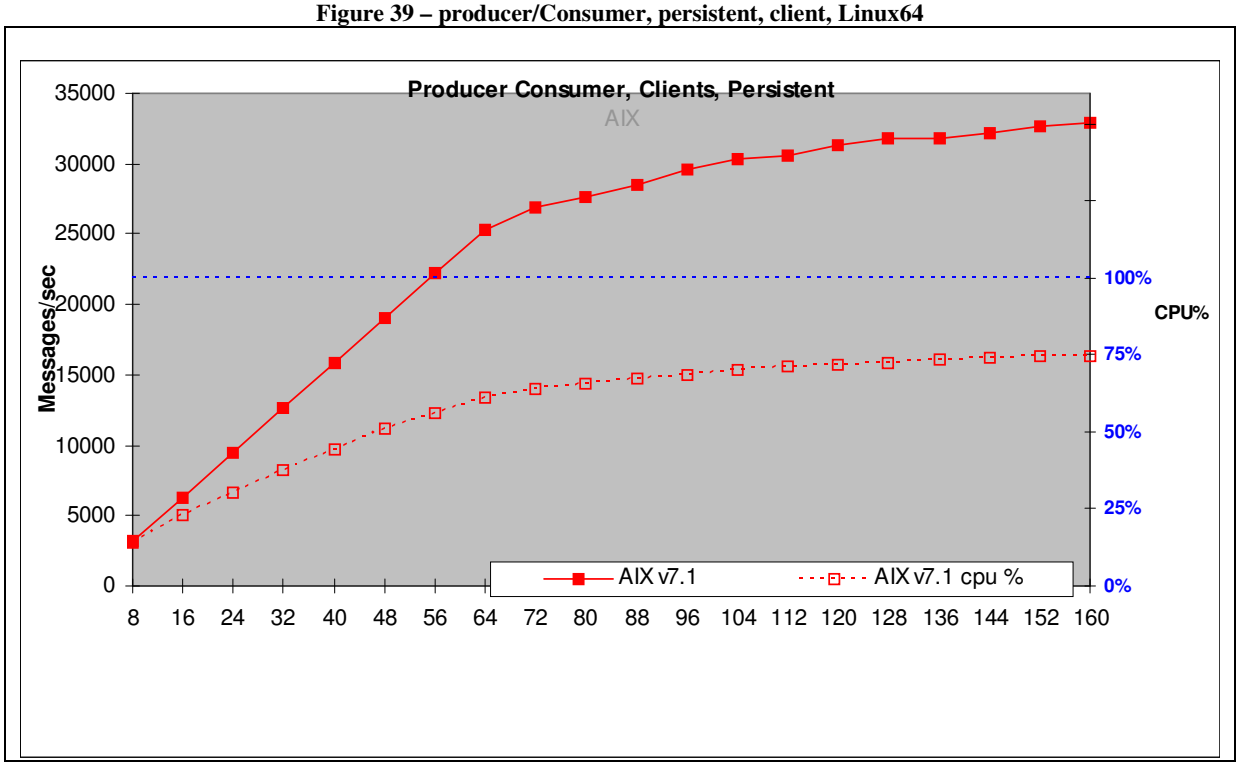

Figure 40 – producer/Consumer, persistent, client, AIX

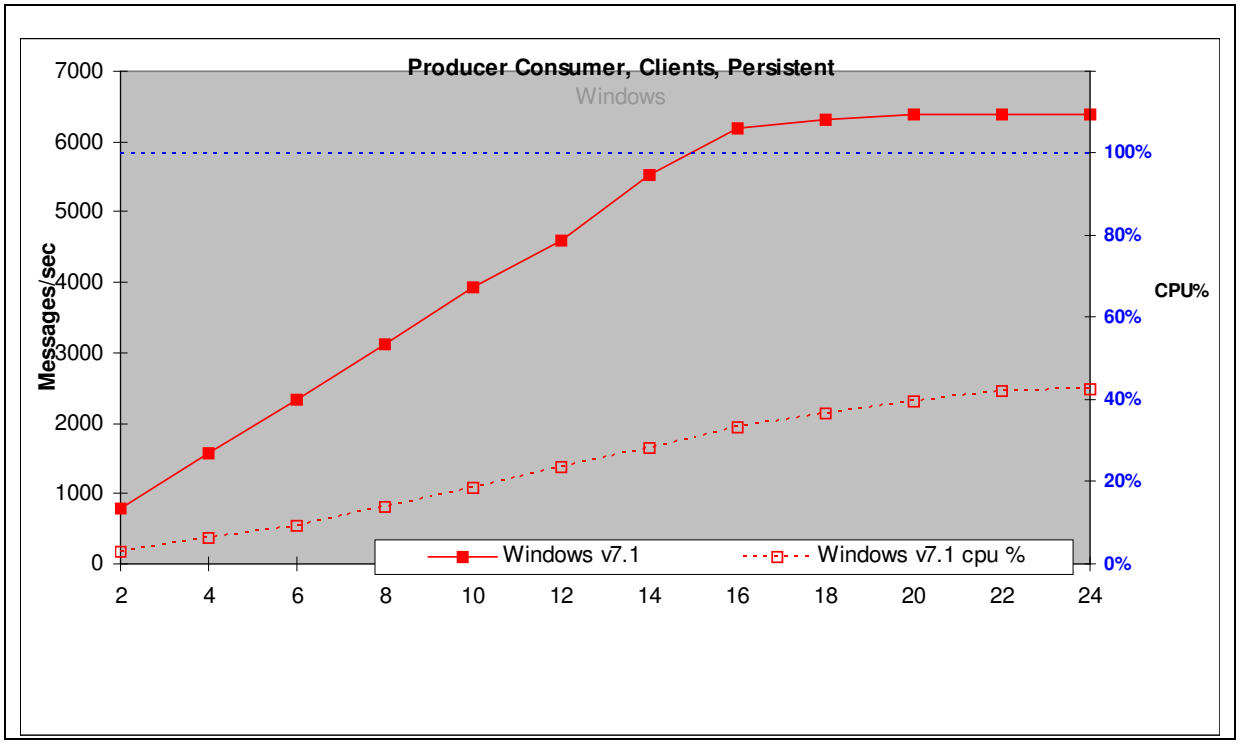

Figure 41 – producer/Consumer, persistent, client, Windows

Test name: **PCCP**	**Apps**	**Messages Per second**	**Server CPU**
Linux64	**80**	**31898**	**28%**
AIX	**160**	**32886**	**75%**
Windows	**20**	**6390**	**40%**

Table 12 – Producer/Consumer, Persistent messages, Client connection

Each message producer creates 400 messages per second and the system throughput increases as a straight diagonal line until the system capacity is reached. With 40 producers and 40 consumers (80 Applications) on Linux64, the expected throughput is 400*40*2=32000 whereas the measured throughput is 31898 messages per second.

3.4 Publish/Subscribe Single Publisher, Many Subscribers Scenario(1:N)

3.4.1 Publish Subscribe 1:N, Non Persistent messages, Client

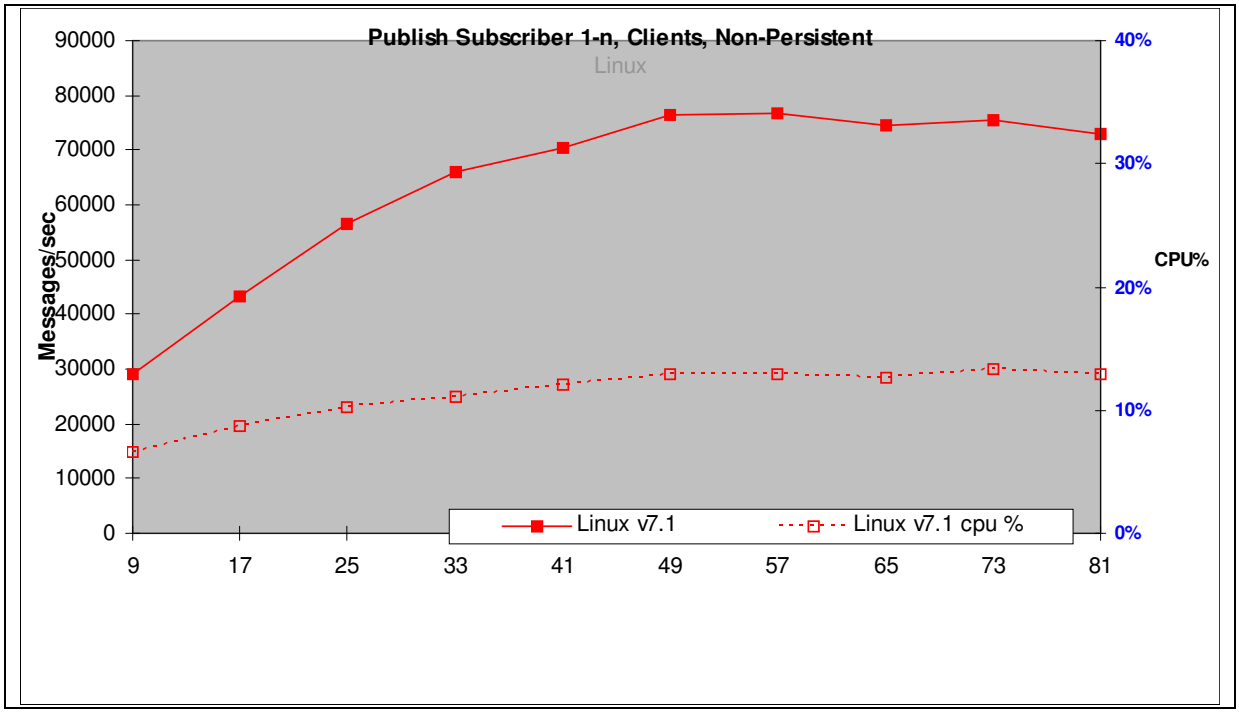

Figure 42 – Publish Subscribe 1:N, non-persistent, client, Linux64

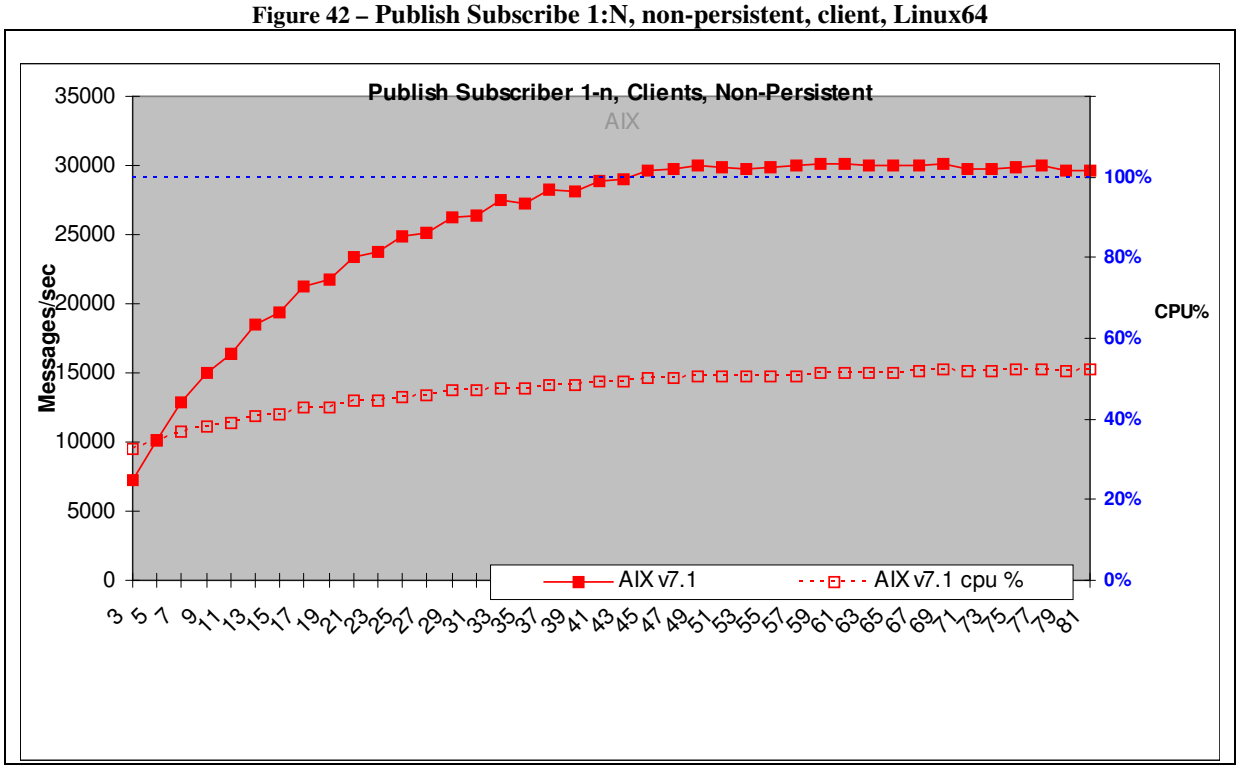

Figure 43 – Publish Subscribe 1:N, non-persistent, client, AIX

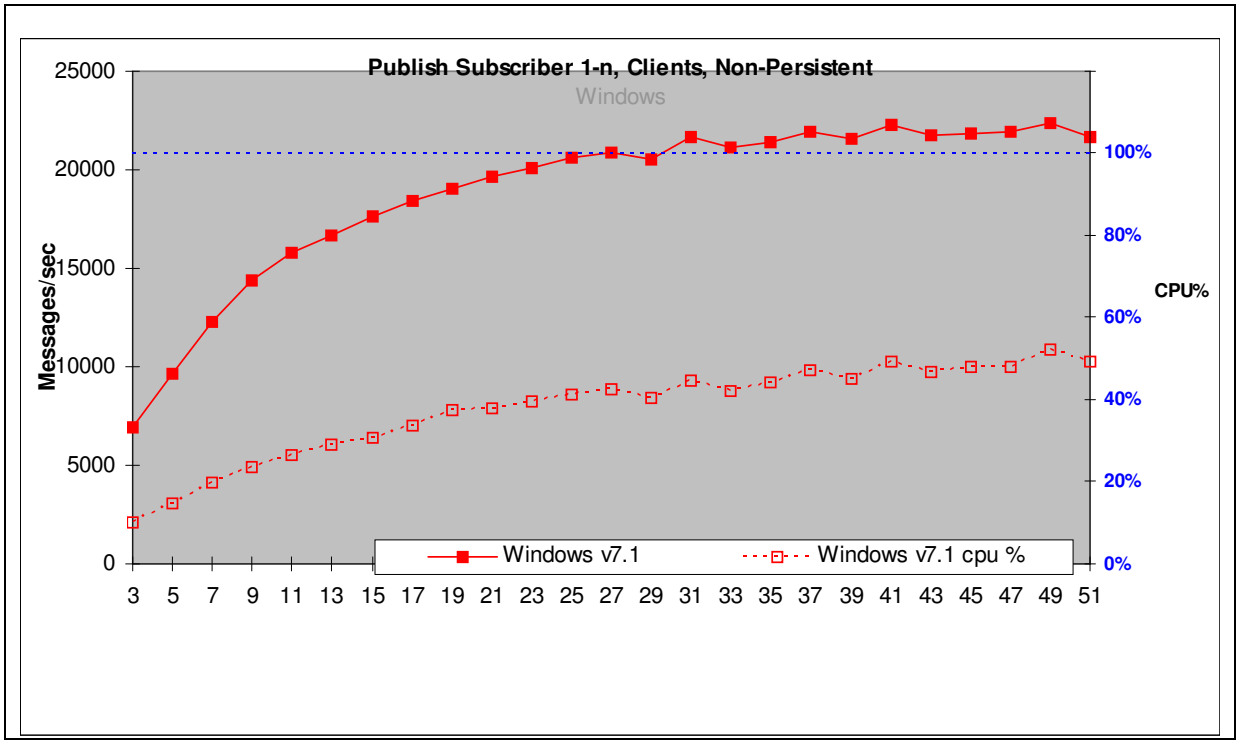

Figure 44 – Publish Subscribe 1:N, non-persistent, client, Windows

Test name: **PS1NCN**	Apps	Messages Per second	Publications per second	Server CPU	Pubs per second With 2 subscribers Per publication
Linux64	**57**	**76732**	**1346**	**13%**	**9632**
AIX	**59**	**30123**	**511**	**57%**	**2398**
Windows	**41**	**22238**	**542**	**49%**	**2307**

Table 13 – Publish/Subscribe 1:N, non-Persistent messages, Client connection

The publisher produces messages as fast as possible. Initially there are 2 subscribers and one publisher when 9632 publications per second can be achieved on Linux64. The response time for the publish command increases as the number of subscribers increase. On Linux64 with 56 subscribers, the publisher creates 1346 messages per second, which are all consumed by the subscribers.

3.4.2 Publish Subscribe 1:N, Persistent messages, Client

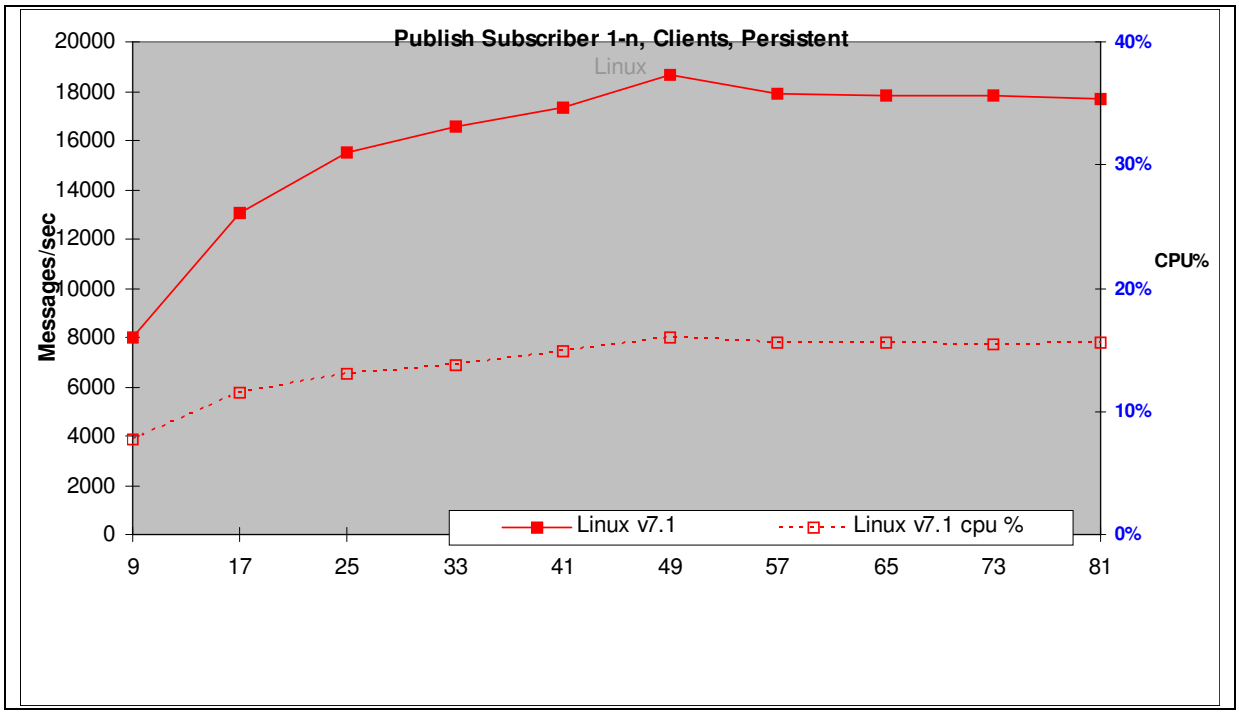

Figure 45 – Publish Subscribe 1:N, persistent, client, Linux64

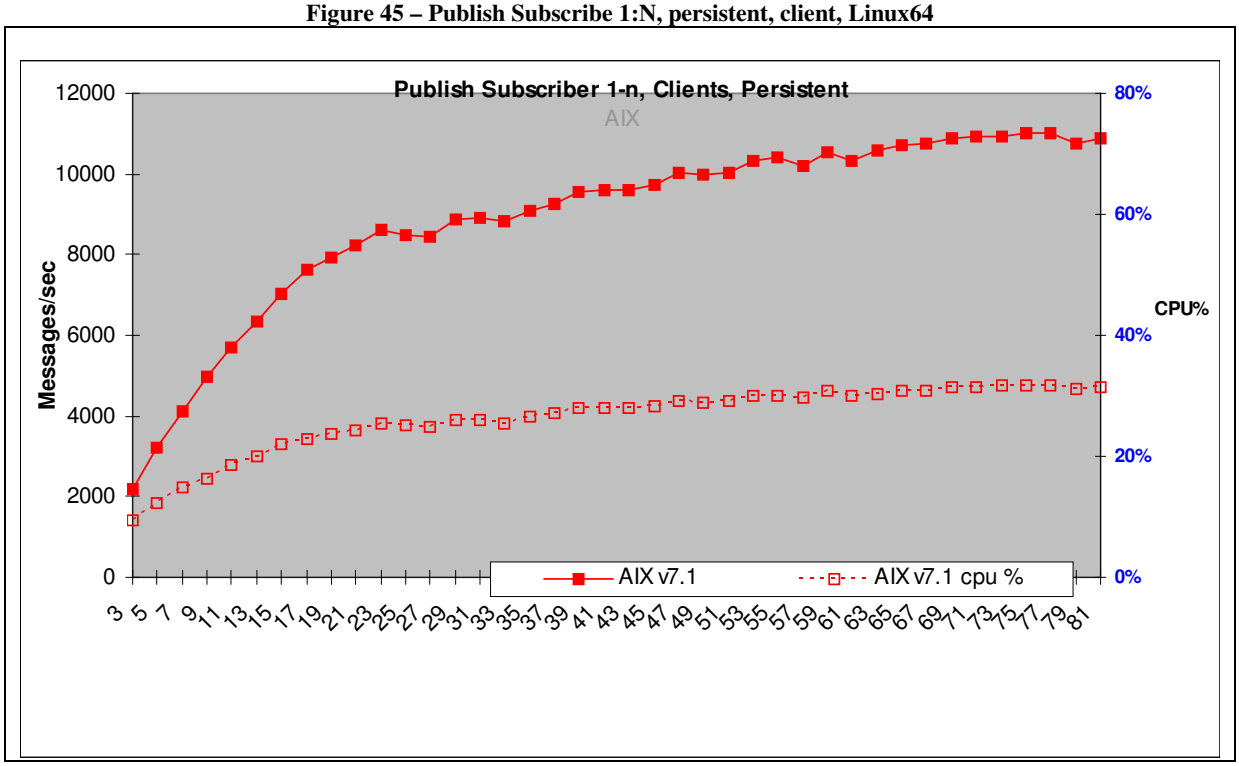

Figure 46 – Publish Subscribe 1:N, persistent, client, AIX

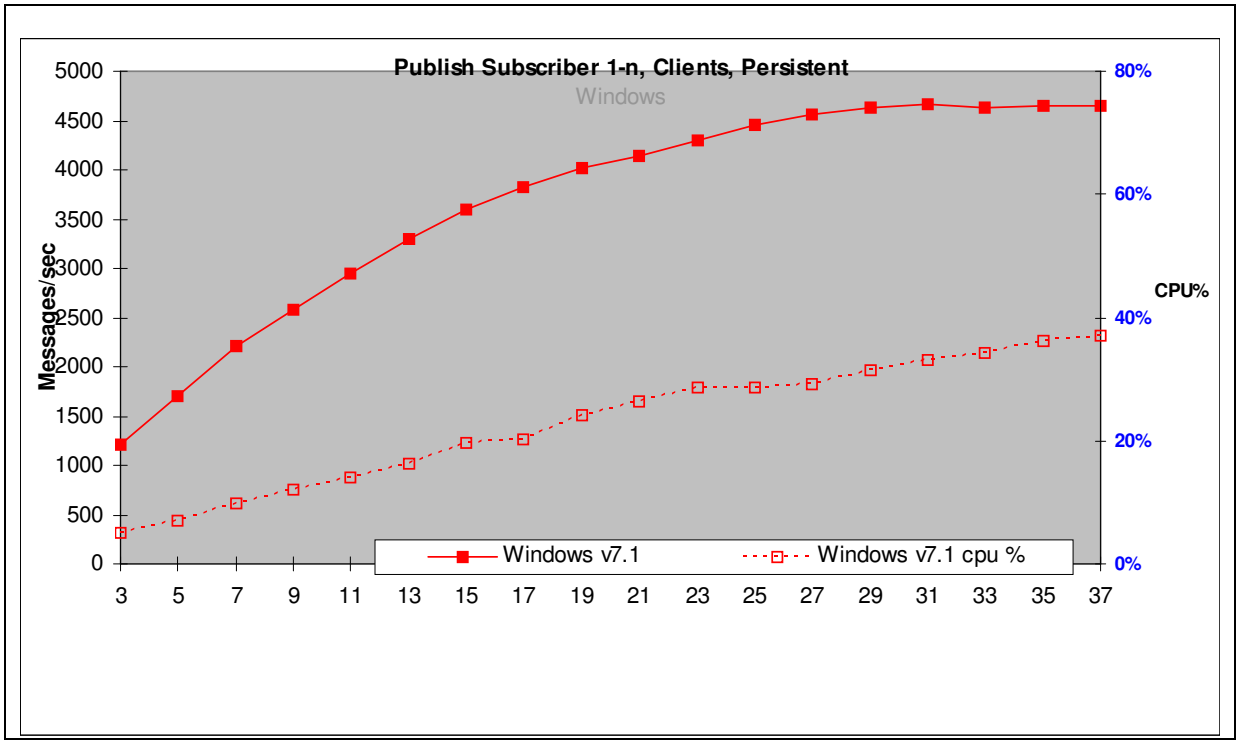

Figure 47 – Publish Subscribe 1:N, persistent, client, Windows

Test name: **PS1NCP**	**Apps**	**Messages Per second**	**Publications per second**	**Server CPU**	**Pubs per second With 2 subscribers Per publication**
Linux64	**49**	**18645**	**380**	**16%**	**2662**
AIX	**77**	**11005**	**143**	**32%**	**729**
Windows	**31**	**4659**	**150**	**33%**	**406**

Table 14 – Publish/Subscribe 1:N, Persistent messages, Client connection

The publisher produces messages as fast as possible. Initially there are 2 subscribers and one publisher when 2662 publications per second can be achieved on Linux64. The response time for the publish command increases as the number of subscribers increase. On Linux64 with 48 subscribers, the publisher creates 380 messages per second, which are all consumed by the subscribers

3.5 Publish Subscribe multiple (Publisher, Topic, Subscriber) scenario

3.5.1 Publish Subscribe (Multiple P/T/S), Non Persistent messages, Client

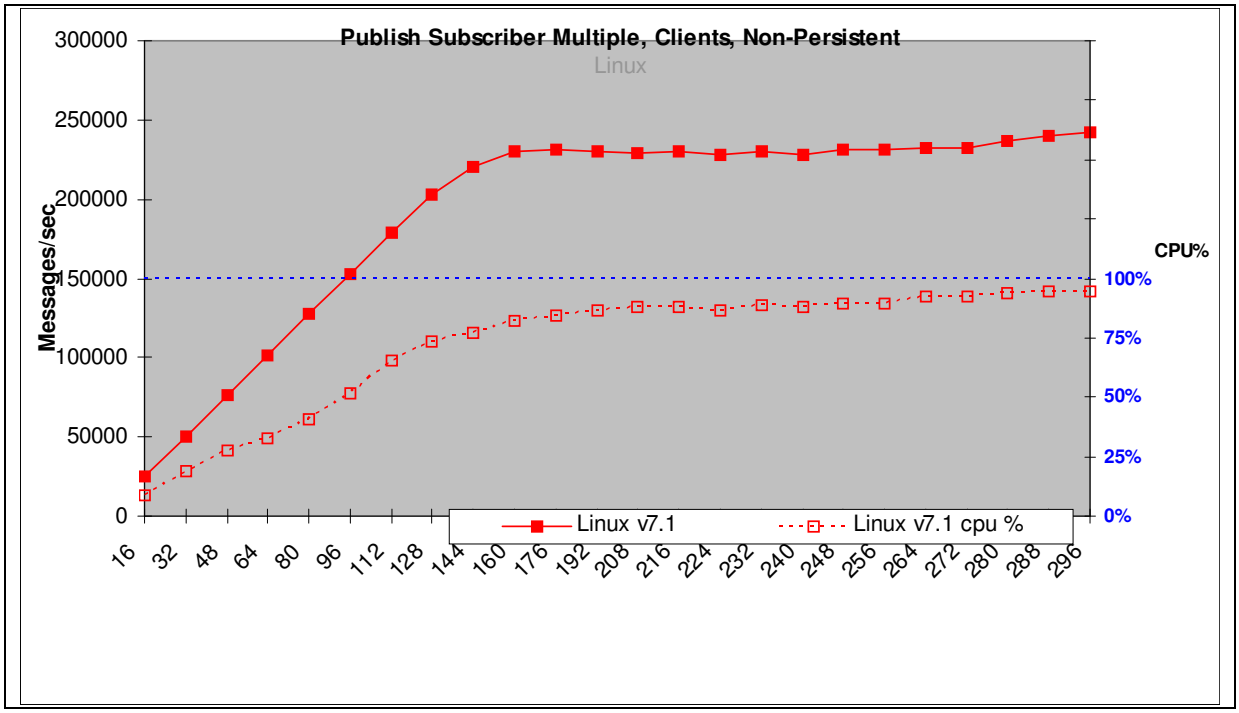

Figure 48 – Publish Subscribe Multiple, non persistent, client, Linux64

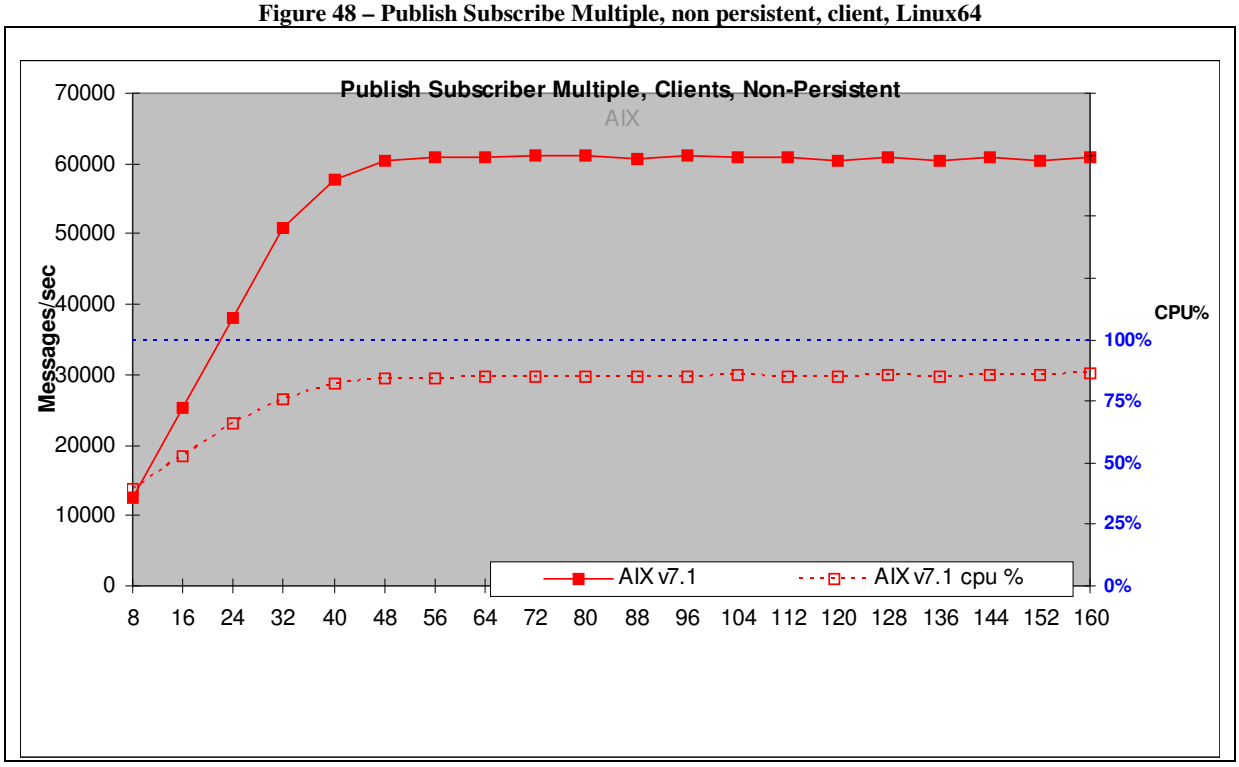

Figure 49 – Publish Subscribe Multiple, non persistent, client, AIX

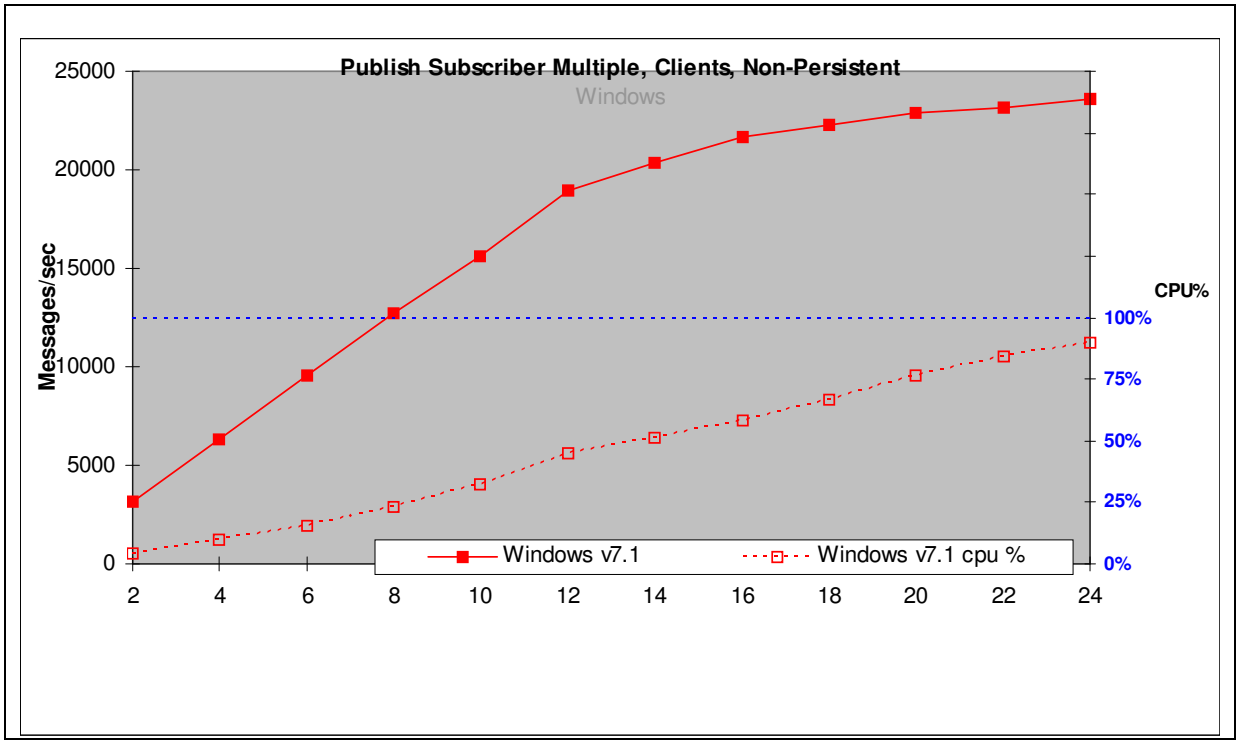

Figure 50 – Publish Subscribe Multiple, non persistent, client, Windows

Test name: **PSMCN**	**Apps**	**Messages Per second**	**Server CPU**
Linux64	**160**	**229872**	**82%**
AIX	**56**	**60824**	**84%**
Windows	**24**	**23598**	**90%**

Table 15 – Publish/Subscribe Multiple, non-Persistent messages, Client connection

Each publisher creates 1600 non-persistent messages per second and the system throughput increases as a straight diagonal line until the system capacity is reached. With 80 producers and 80 consumers (160 Applications) on Linux64, the expected throughput is 1600*80*2=256000 whereas the measured throughput is 229872 messages per second

3.5.2 Publish Subscribe (Multiple P/T/S), Persistent messages, Client

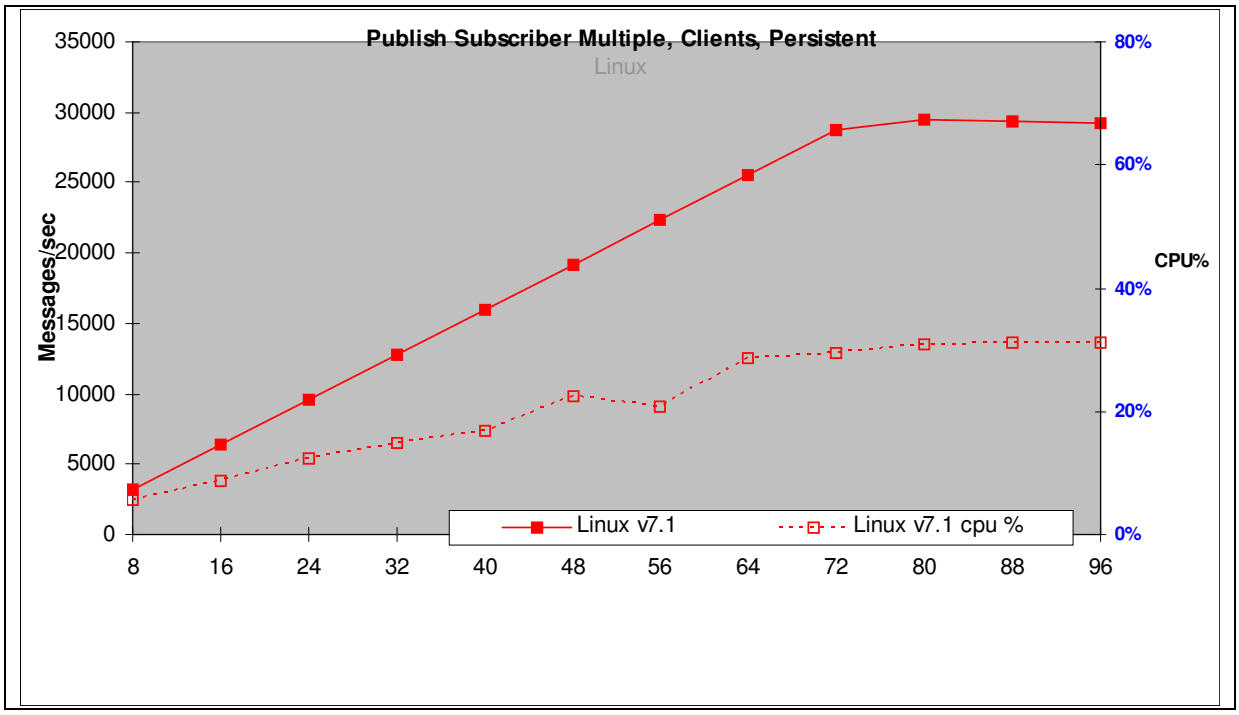

Figure 51 – Publish Subscribe multiple, persistent, client, Linux64

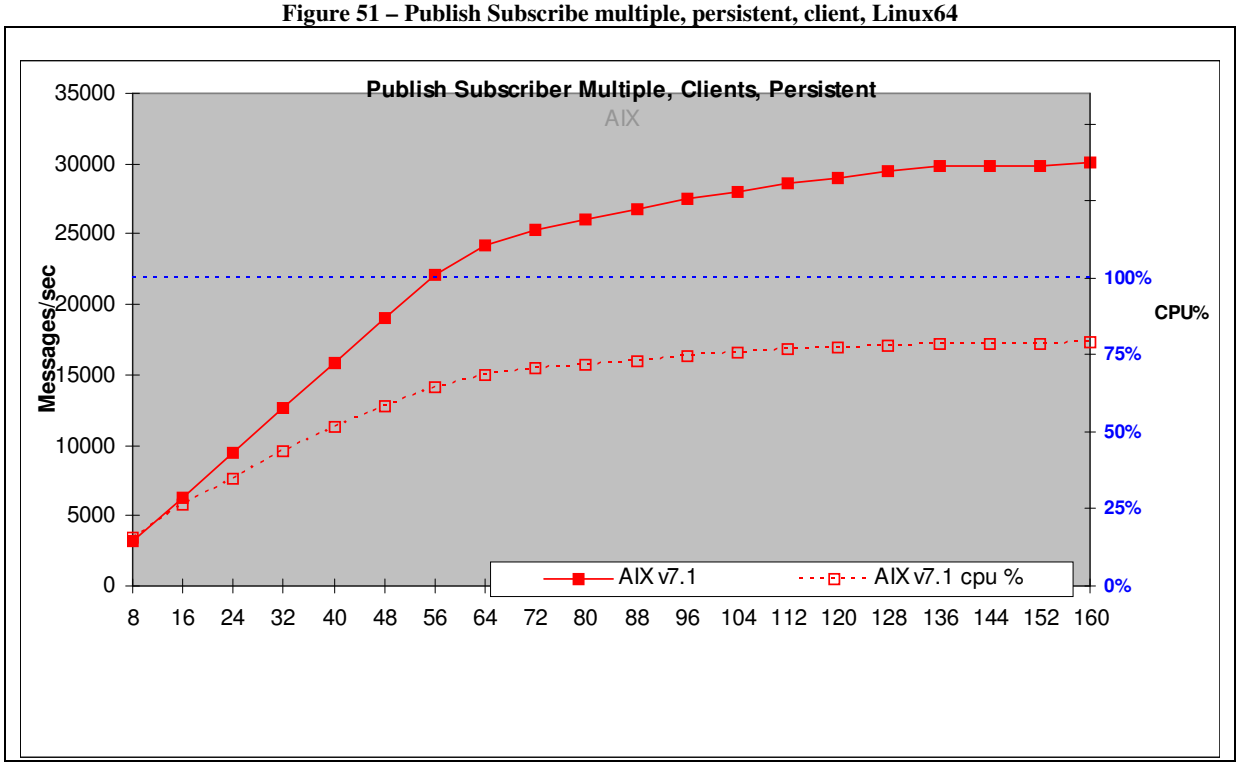

Figure 52 – Publish Subscribe multiple, persistent, client, AIX

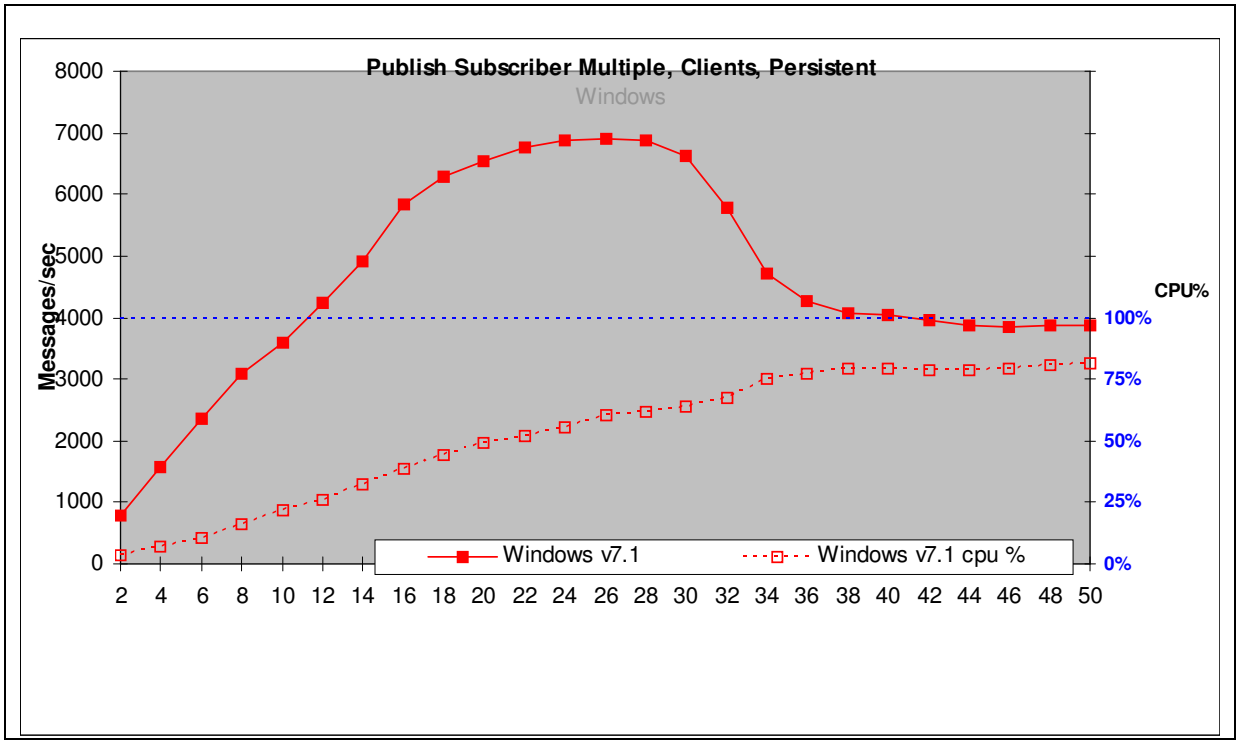

Figure 53 – Publish Subscribe multiple, persistent, client, Windows

Test name: **PSMCP**	**Apps**	**Messages Per second**	**Server CPU**
Linux64	**80**	**29491**	**31%**
AIX	**160**	**30073**	**79%**
Windows	**26**	**6894**	**60%**

Table 16 – Publish/Subscribe Multiple, Persistent messages, Client connection

Each message producer creates 400 messages per second and the system throughput increases as a straight diagonal line until the system capacity is reached. With 40 producers and 40 consumers (80 Applications) on Linux64, the expected throughput is 400*40*2=32000 whereas the measured throughput is 29491 messages per second.

4 z/OS – Client mode

This chapter shows the throughput achieved when the z/OS Queue Manager is used with the benchmarks run in Client mode using either Private or Shared queues. Private queues are local to the Queue Manager, whereas Shared queues can be shared between Queue Managers using the z/OS Coupling Facility. Client mode means the JMS applications are running on Linux64 with a client channel to the z/OS server.

4.1 Requester-Responder Scenario

4.1.1 Requester-Responder Non-Persistent Messages

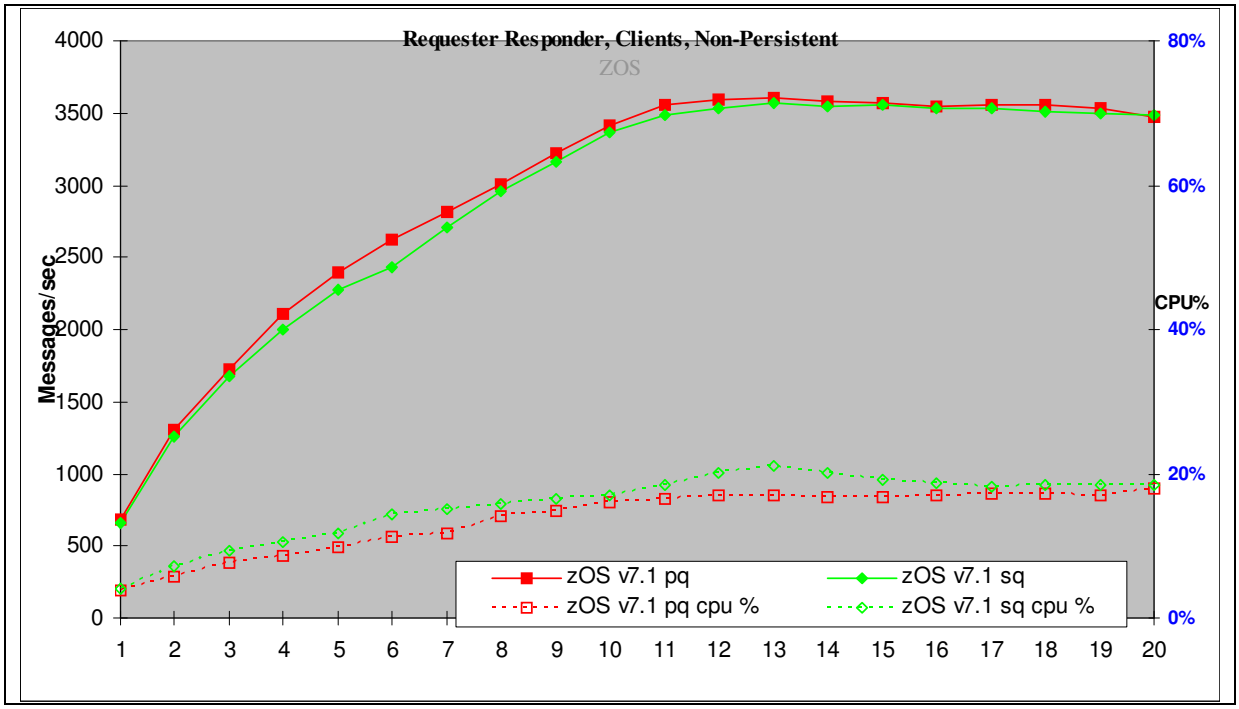

Figure 54 – Requester-Responder, non-persistent, client, z/OS

Test name: **RRQCN**	**Apps**	**Round Trips/sec**	**CPU**
Private	**12**	**3557**	**16%**
Shared	**13**	**3547**	**19%**

Table 17 – Requester-Responder, non-persistent messages

4.1.2 Requester-Responder Persistent Messages

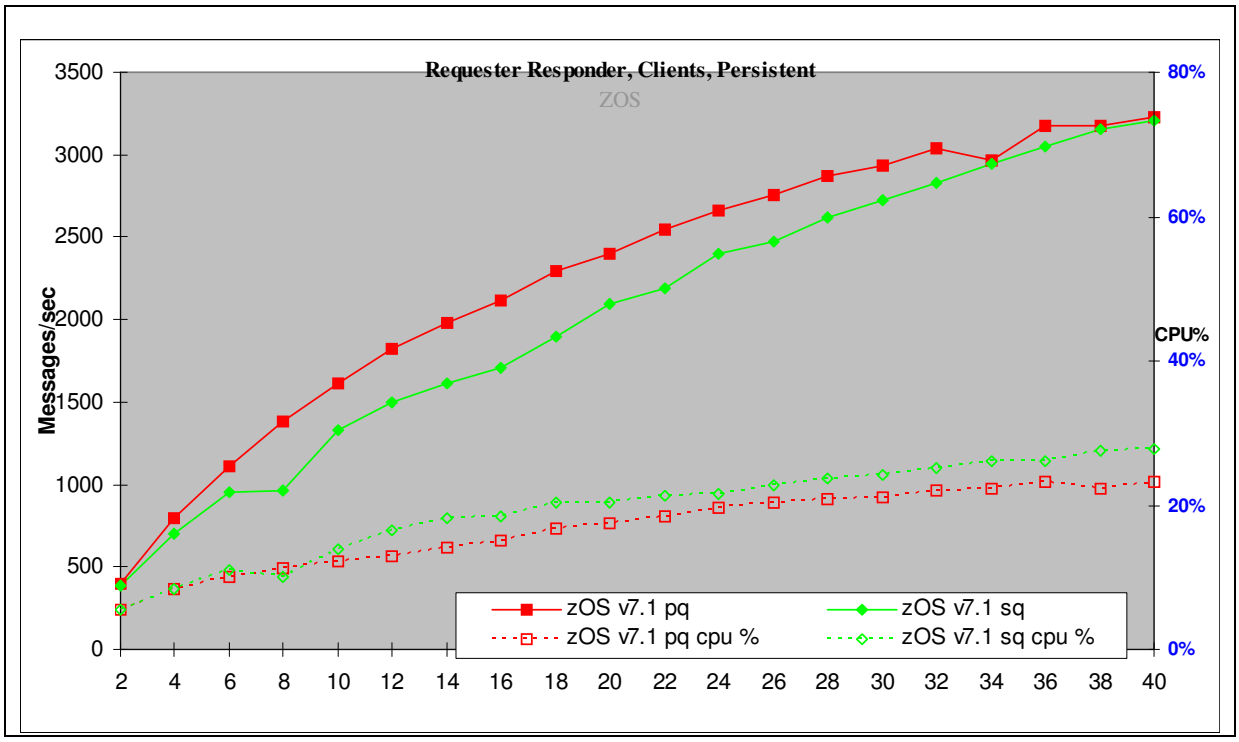

Figure 55 – Requester-Responder, persistent, client, z/OS

Test name: **RRQCP**	**Apps**	**Round Trips/sec**	**Server CPU**
Private	**38**	**3243**	**24%**
Shared	**40**	**3079**	**30%**

Table 18 – Requester-Responder, Persistent messages

With Persistent messages, more applications in parallel allow the logger to work more efficiently.

4.2 Publish/Subscribe Single Publisher, Many Subscribers Scenario (1:N)

4.2.1 Publish Subscribe 1:N, Non Persistent messages

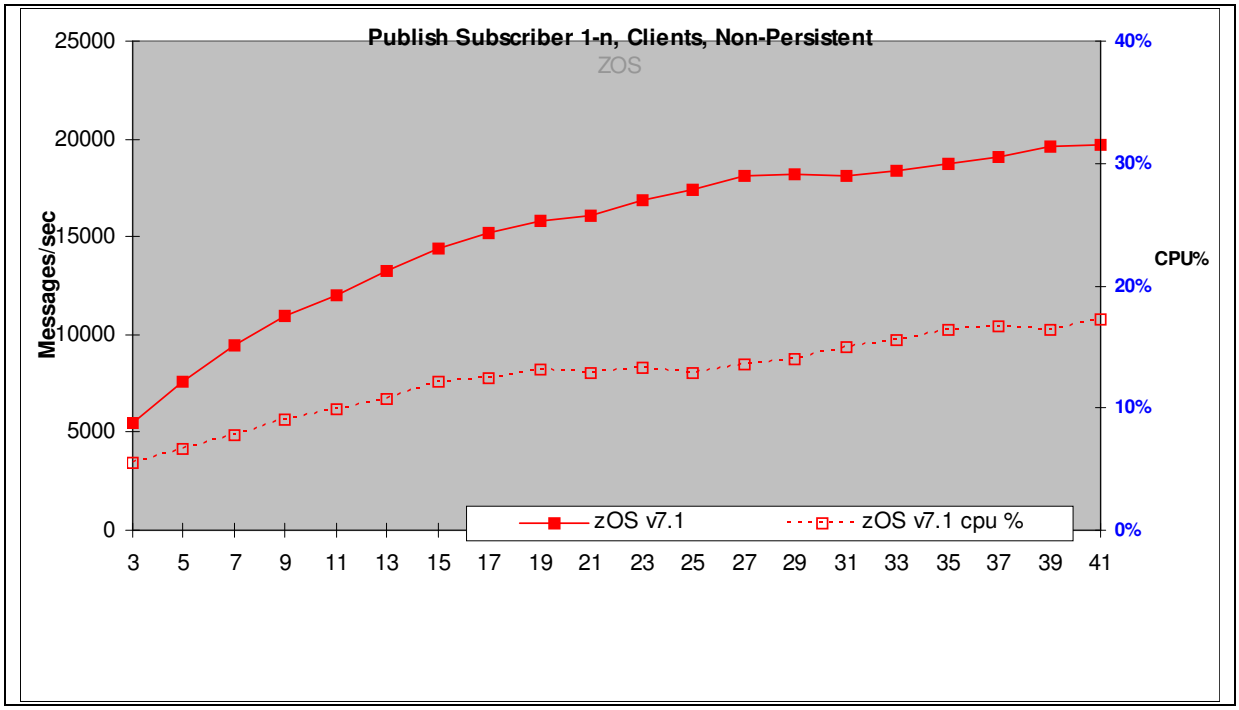

Figure 56 – Publish Subscribe 1:N, non-persistent z/OS

Test name: **PS1NCN**	**Apps**	**Messages Per second**	**Publications per second**	**Server CPU**	**Pubs per second With 2 subscribers Per publication**
Client	**41**	**19322**	**471**	**17%**	**1774**

Table 19 – Publish/Subscribe 1:N, non-Persistent messages

The publisher produces messages as fast as possible. Initially there are 2 subscribers and one publisher when 1774 publications per second can be achieved on z/OS. The response time for the publish command increases as the number of subscribers increase. On z/OS with 40 subscribers, the publisher creates 471 messages per second, which are all consumed by the subscribers.

4.2.2 Publish Subscribe 1:N, Persistent messages

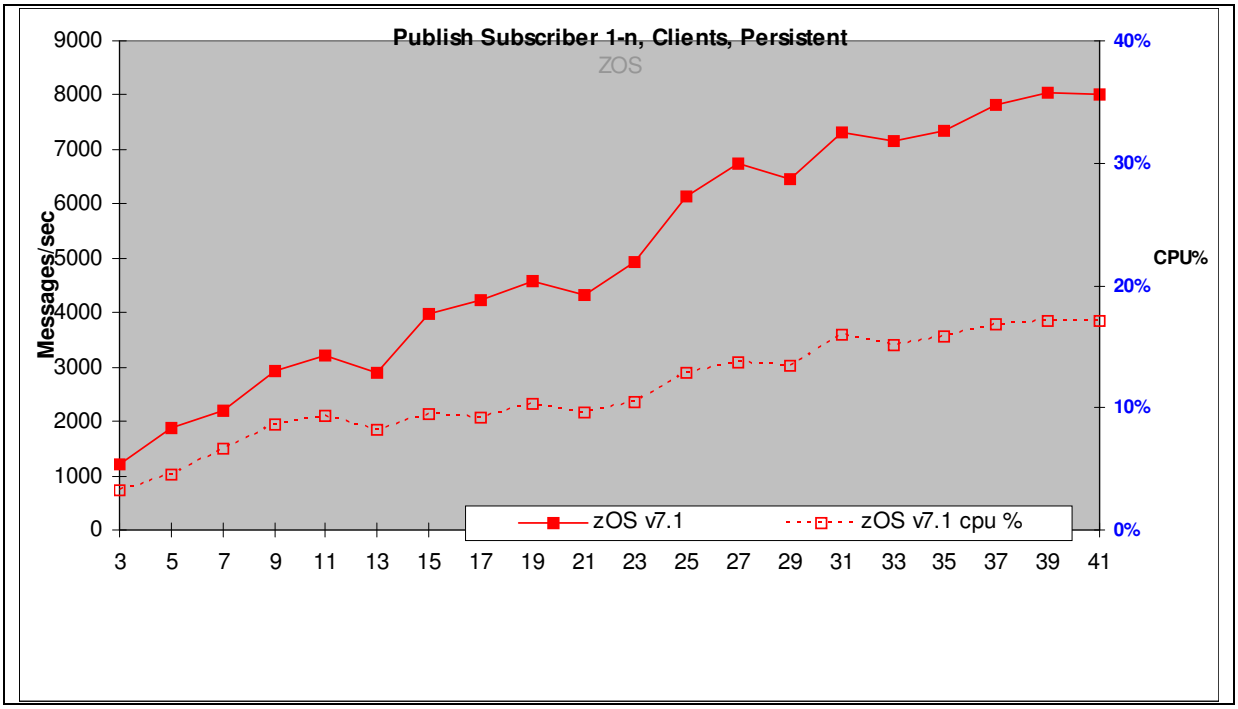

Figure 57 – Publish Subscribe 1:N, persistent z/OS

Test name: **PS1NZP**	Apps	Messages Per second	Publications per second	Server CPU	Pubs per second With 2 subscribers Per publication
Client	**41**	**9128**	**223**	**22%**	**591**

Table 20 – Publish/Subscribe 1:N, Persistent messages

The publisher produces messages as fast as possible. Initially there are 2 subscribers and one publisher when 591 Publications per second can be achieved on z/OS. The response time for the publish command increases as the number of subscribers increase. On z/OS with 40 subscribers, the publisher creates 223 messages per second, which are all consumed by the subscribers

4.3 Put/Get with 4 Queues Scenario

In this scenario, there are 4 Queues. Each Put/Get application is allocated a particular Queue.
The application puts a message to the queue, and stores the message identifier returned in the message descriptor. It then gets the message from the queue using the message identifier. Only one message per Put/Get application exists at any point in time. This scenario uses synchronous messaging.

4.3.1 Put/Get Non-Persistent messages

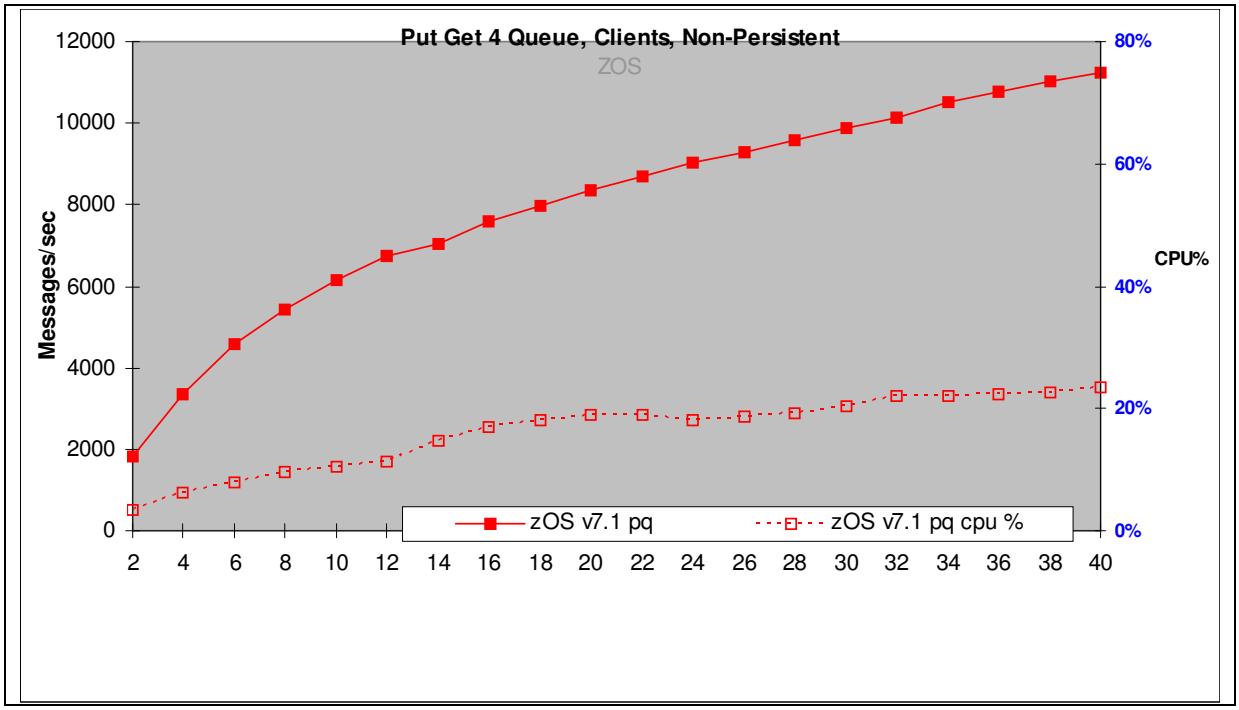

Figure 58 – Put/Get, non-persistent 2K z/OS

Test name: **JPG4QCN**	**Apps**	**Messages /sec**	**CPU**
Private	**40**	**9878**	**21%**
Shared	**40**	**10049**	**21%**

Table 21 – Put/Get, 2K non-persistent messages

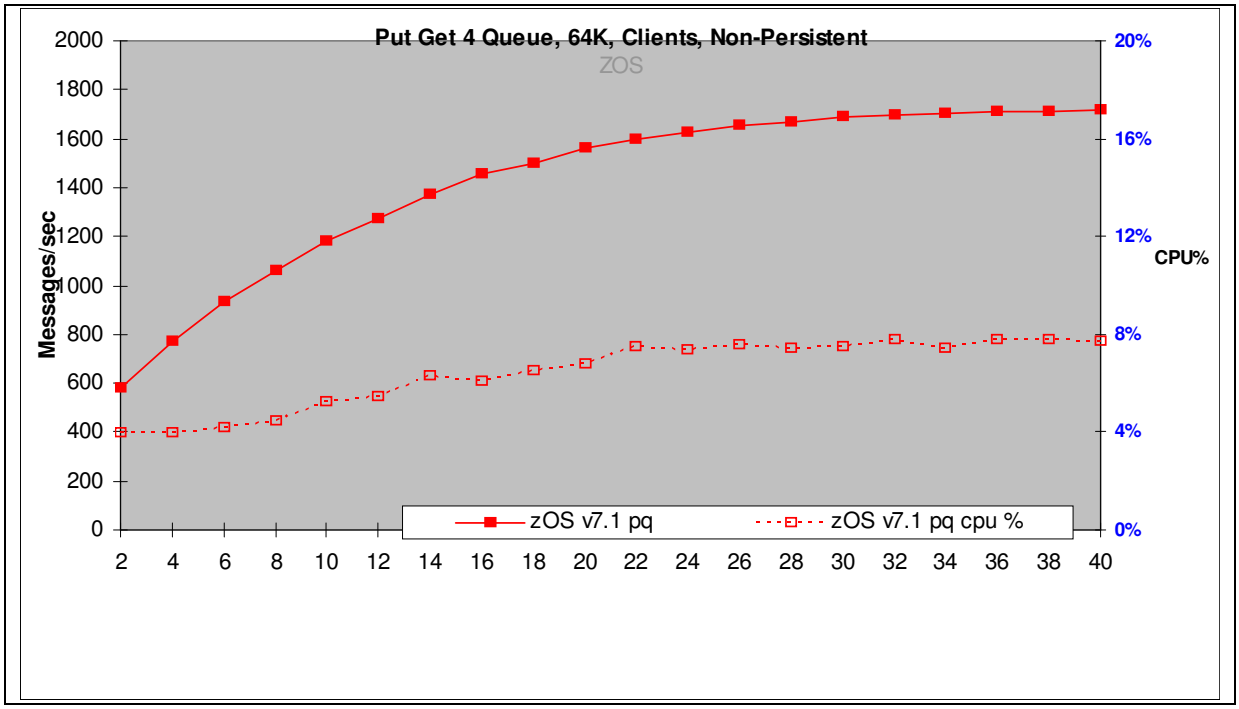

Figure 59 – Put/Get, non-persistent 64K z/OS

Test name: **JPG4QCN**	**Apps**	**Messages /sec**	**CPU**
Private	**40**	**1660**	**8%**
Shared	**40**	**1691**	**8%**

Table 22 – Put/Get, 64K non-persistent messages

4.3.2 Put/Get Persistent messages

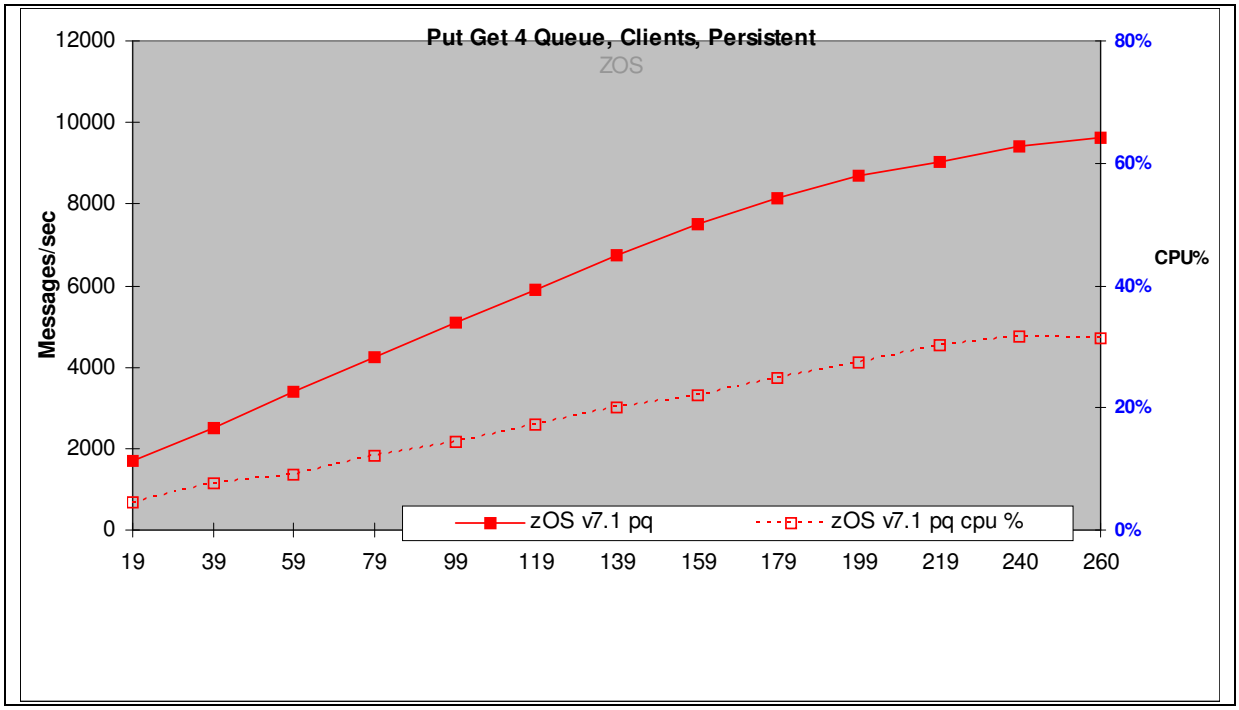

Figure 60 – Put/Get, persistent 2K z/OS

Test name: **JPG4QCP**	**Apps**	**Messages /sec**	**CPU**
Private	**200**	**8749**	**35%**
Shared	**200**	**8790**	**31%**

Table 23 – Put/Get, 2K persistent messages

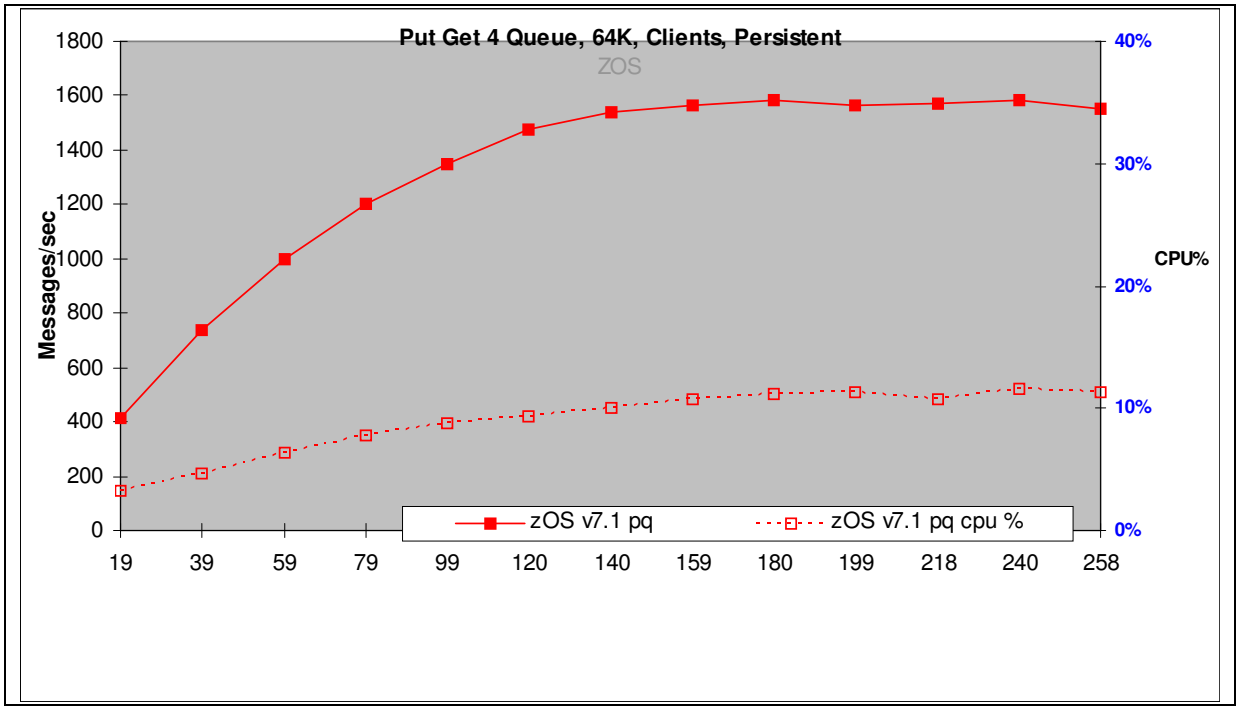

Figure 61 – Put/Get, persistent 64K z/OS

Test name: **JPG4QCN**	**Apps**	**Messages /sec**	**CPU**
Private	**100**	**1650**	**14%**
Shared	**100**	**1612**	**12%**

Table 24 – Put/Get, 64K persistent messages

5 Large Messages

The effect of message size on throughput was investigated on the Linux64 platform using the Requester-Responder Scenario described in Section 2.1 using Client Channels.

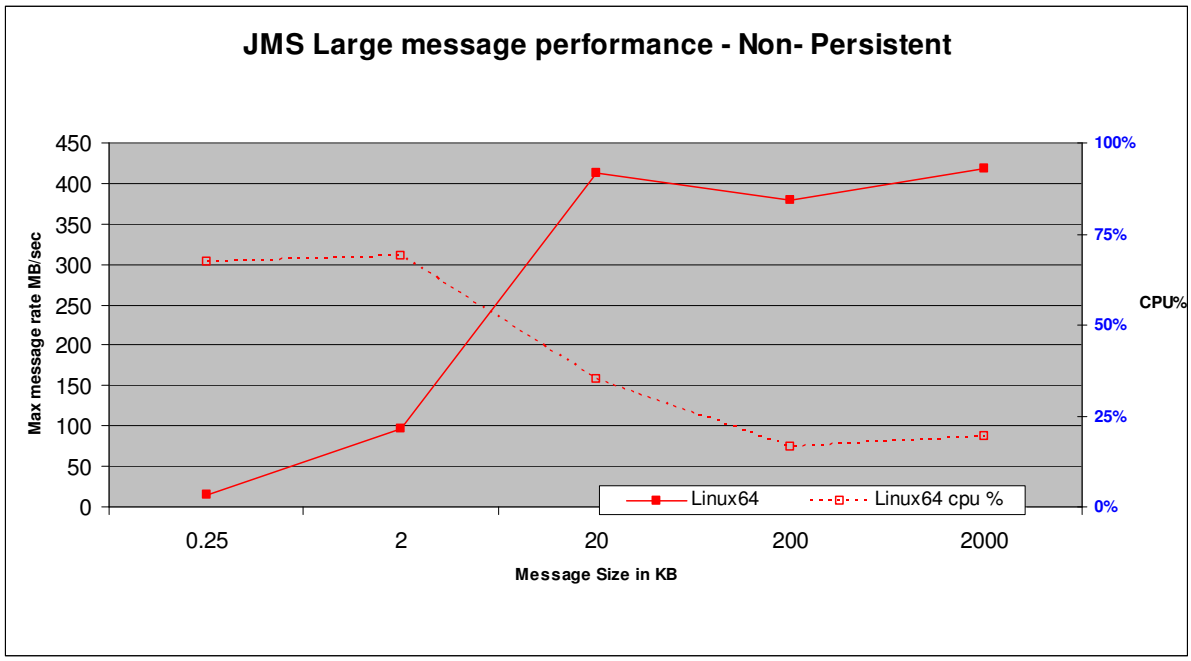

Figure 62 – Large Message Performance - non-persistent

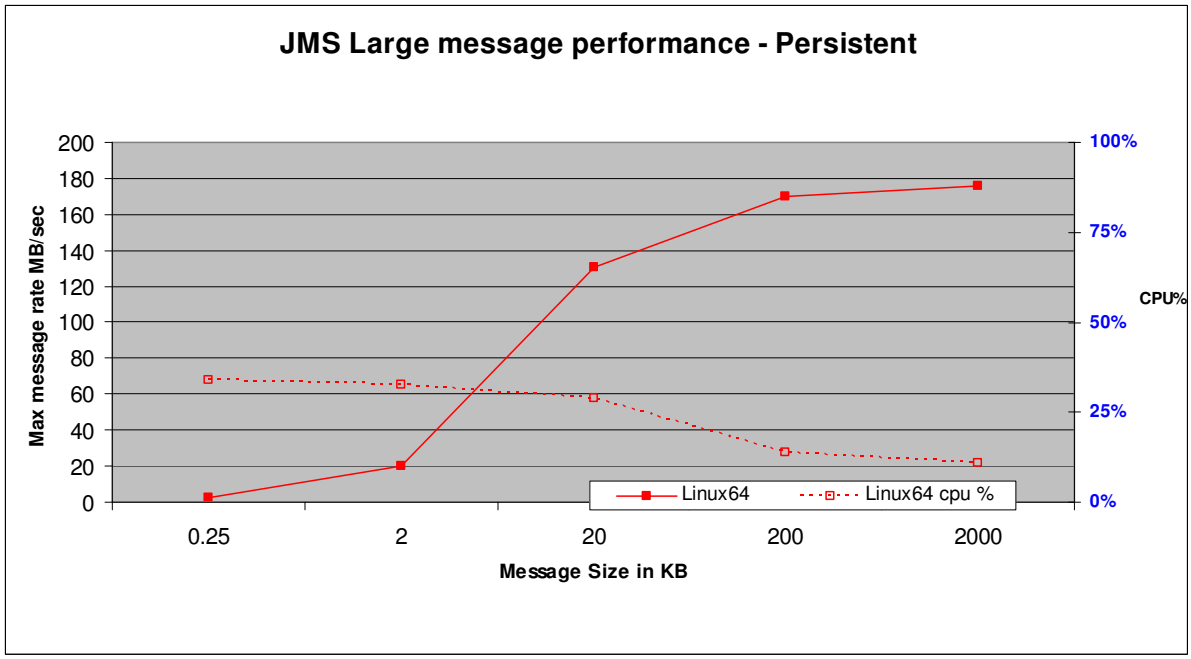

Figure 63 – Large Message Performance -persistent

Figures **Figure 62** and **Figure 63** show the maximum sustainable message rate relative to message size for a range of different message sizes from 256 bytes to 2MB.
The reported result at each message size is the maximum throughput achievable without queuing, multiplied by the message size to give a throughput rate in MB/sec.

6 Performance Enhancements inV7.1

There have been a number of improvements in the locking and threading code in V7.1 and in logging of persistent messages. The throughput increase which results from these changes is most significant on multi-core machines such as the 2 x 6-core hyper threaded machine used for Linux64 testing.

6.1 Throughput improvements using a single queue

The Requester-Responder scenario described in Section 2.1 of this report uses a single Request queue for all Requesters and a single Reply queue for all Responders. Reduction in contention between the clients putting messages to, and getting messages from, these two queues results in a significant improvement in throughput as can be seen in the graph below.

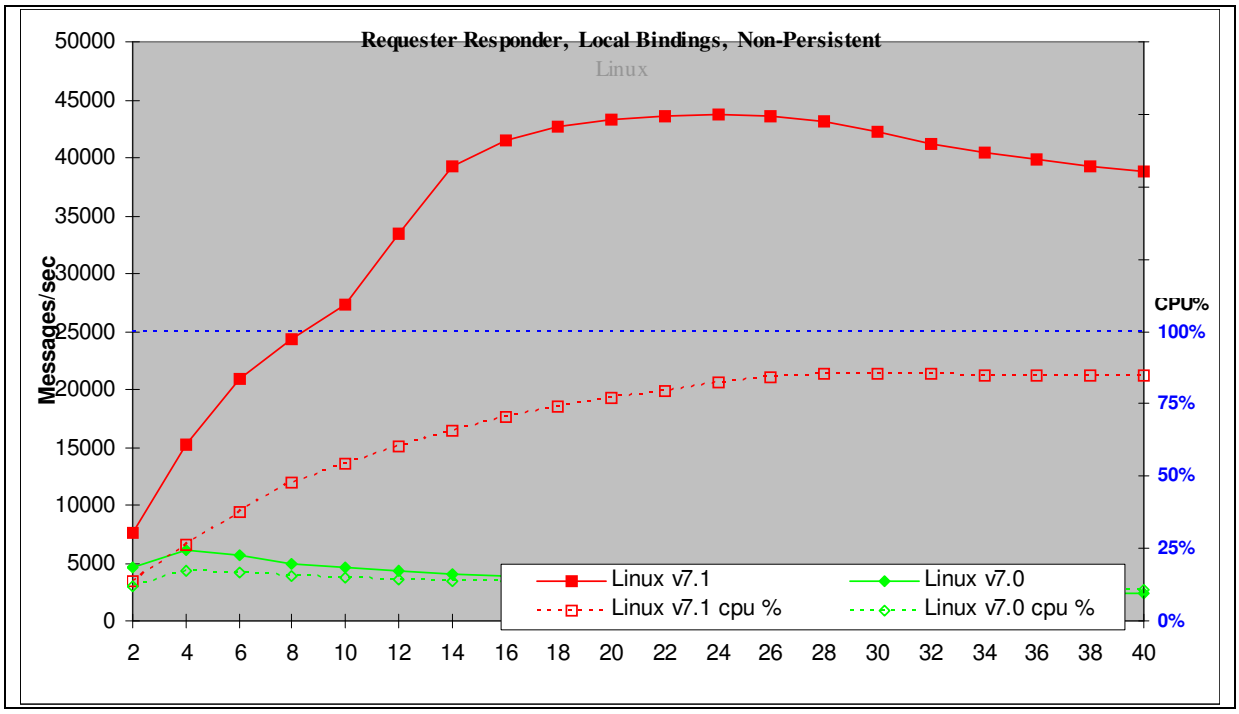

Figure 64 – Comparison V7.1 to V7.0 Requester/Responder, non-persistent, local queue manager, Linux64

The number of clients that can simultaneously access the queues without degrading performance has increased from 4 to 24 and the percentage of the available CPU that can be exploited by this scenario has increased from 17% to 80% on Linux64. The maximum throughput increased by 607% on Linux64, 189% on AIX and 45% on Windows.

6.2 Throughput improvements using multiple queues

The Point-to-Point and Pub-Sub 'Multiple' test scenarios described in Sections 2.2 and 2.5 of this report use separate queues for each pair of Producers and Consumers. The throughput improvement in these tests is not as marked as it is in the single queue case above but is still significant.

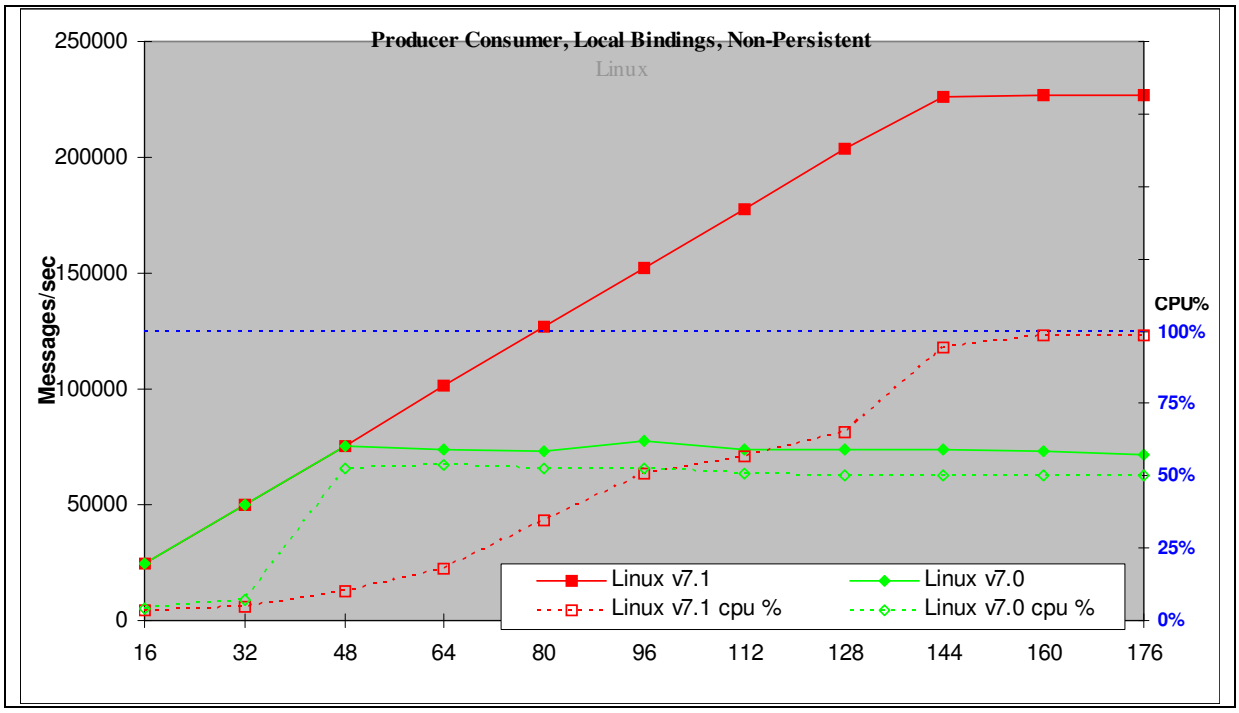

Figure 65 – Comparison V7.1 to V7.0 Producer/Consumer, non persistent, local queue manager, Linux64

The percentage of the available CPU that can be exploited by this scenario has increased from 54% to 98% on Linux64. The maximum throughput increased by 200% on Linux64, 7% on AIX and was unchanged on Windows.

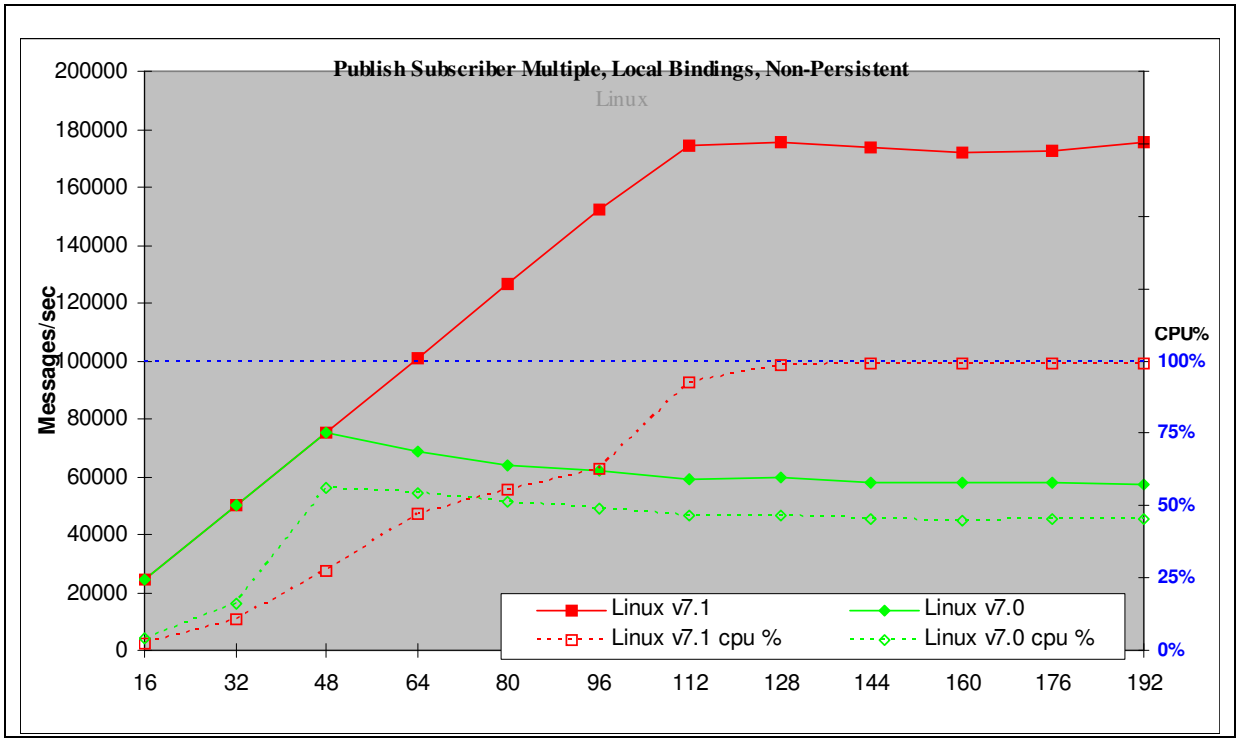

Figure 66 – Comparison V7.1 to V7.0 Publish Subscribe Multiple, non-persistent, local queue manager, Linux64

The percentage of the available CPU that can be exploited by this scenario has increased from 56% to 98% on Linux64. The maximum throughput increased by 130% on Linux64, 10% on AIX and was unchanged on Windows.

6.3 Throughput improvements for persistent tests

Throughput improvements were also seen for scenarios using persistent messages. This is due to improvements in the code that handles message logging. The example below shows the improvement in the Requester-Responder scenario on Linux64.

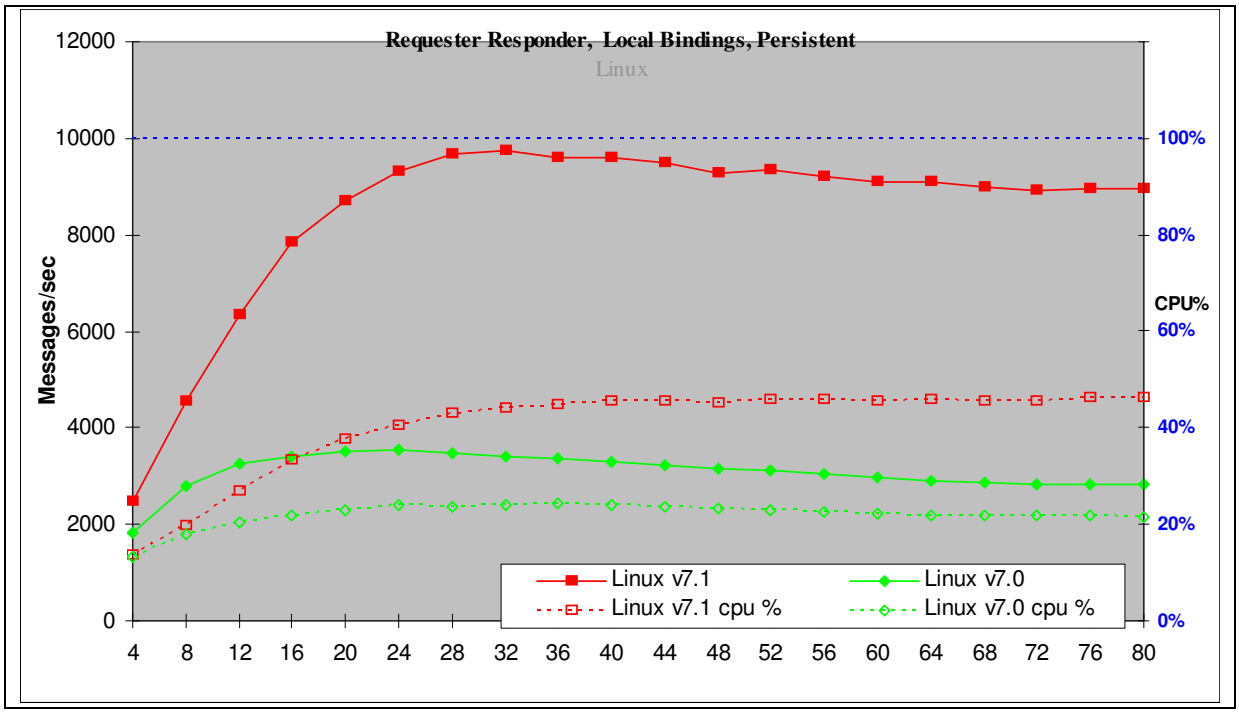

Figure 67 – Comparison V7.1 to V7.0 Requester/Responder, persistent, local queue manager, Linux64

The percentage of the available CPU that can be exploited by this scenario has increased from 23% to 45% on Linux64. The maximum throughput increased by 175% on Linux64, 160% on AIX and 57% on Windows.

6.4 Throughput using single Publisher

The Publish/Subscribe Single Publisher scenario described in Section 2.3 of this report uses a single Publisher to publish messages to multiple Subscribers. The throughput of this scenario is limited by the Publisher and hence is unchanged within measurement accuracy from V7.0.

7 Tuning/programming guidelines

7.1 Tuning the queue manager

Performance reports with tuning information for WebSphere MQ v7.1 on all supported operating systems can be found on the IBM SupportPac webpage at the following URL:
http://www.ibm.com/software/integration/wmq/support/

The main tuning actions taken for the tests in Chapter 2 and 3 of this report were:

- Log / LogBufferPages = 4096 (size of memory used to build log I/O records)
- Log / LogFilePages = 16348 (size of Log disk file extent)
- Log / LogPrimaryFiles = 16 (number of disks extents in log cycle)
- LogWriteIntegrity=TripleWrite

- Channels / MQIBindType = FASTPATH (channels are an extension to QM address space)
- Channels / SHARECNV = 1
- TuningParameters / DefaultQBufferSize = 1MB (use 1MB of main memory per Q to hold non persistent messages before spilling to the file system
- TuningParameters / DefaultPQBufferSize = 1MB (use 1MB of main memory per Q to hold persistent messages)

7.2 Shared Conversations

Clients producing or consuming a small number of messages per second can usefully share the TCP socket with other threads in the same process. For MQ v7.1 the default is for 10 applications to share a channel and hence a TCP socket. Benchmarks that produce or consume multiple hundreds of messages per second will bottleneck on the shared socket and should use a single socket per application by setting SHARECNV=1.

7.3 Avoiding running in Migration/Compatibility Mode

An MQ JMS 7.1 client can connect to V7.1, V6 and V7 Queue Managers. When connected to a V6 Queue Manager a less optimised codepath is used. This facilitates migration from V6 to V7 but should not be considered as a long-term solution if performance is important.
It is also possible to connect a V7.1 JMS client to a V7.1 Queue Manager in migration mode by setting WMQ_PROVIDER_VERSION to "6.0.0.0" on the ConnectionFactory, but for best performance the default V7.1 value should be used.

7.4 Tuning the heap size for Java

During operation, current garbage collectors (GC) will normally interrupt the execution of all other threads in a JVM to some extent. The level of interruption depends on the amount and the type of work the GC is doing. This is largely dependant on how the memory is being used by the application and the GC settings currently in operation.
JMS has characteristics such that fixed memory requirements are low but transient memory requirements can be high, depending on message size and application design. Without tuning, or with incorrect tuning, the automatic garbage collection policies of Java can adversely affect messaging performance.
The most common GC settings are:

- `-Xms` Minimum heap size.
- `-Xmx` Maximum heap size.
- `-verbose:gc` Display garbage collection events.

As an example, the following line fixes the heap size at 512MB and enables verbose garbage collection.

```
java -Xms512M -Xmx512M -verbose:gc
```

Recommendations

- Use `-verbose:gc` to monitor the frequency of your application's garbage collection under different loads and adjust the minimum and maximum heap sizes accordingly.
- A garbage collection interval of less than one second is detrimental to performance. A sensible minimum GC interval is 1-2 seconds, but consideration should also be given to the GC pause time.
- If the machine has sufficient memory then setting –Xms equal to –Xmx will allocate the specified maximum heap size at jvm startup. This avoids any costs involved in dynamically resizing the heap.
- GC implementations offer a variety of GC policies including concurrent and generational modes and you should consult your JVM documentation to determine the best option for your workload, then experiment with -verbose:gc to tune the settings.

7.5 JVM Warmup

- JVMs employ sophisticated Just-In-Time (JIT) compilers to optimise the executable code. These JITs can continue to recompile selected java methods for many minutes or even hours after the jvm has initialised. Full performance may not be achieved until this is completed, and indeed the cost of

compilation can slow down performance in the early stages of execution. In most cases the default JIT settings will give best overall performance but in situations where a faster startup is desirable the JIT activity can be reduced at the expense of absolute performance.

- Performance Measurements on JMS workloads should only be done after a warmup period to ensure JIT activity has largely completed. You may need to experiment to find this point.

7.6 Use of Correlation Identifiers

- Selecting against *correlationId* or *messageId* follows an optimised path through WebSphere MQ 7.1 and the selection occurs on the server-side (in the queue manager). This gives better performance than when using arbitrary JMS selectors.
- Use of the provider-specific "ID:" tag is applicable to these two fields only and is of practical use only with correlation identifiers.
- To use the optimised path, the *correlationId* must be prefixed with "ID:" and must be formatted correctly as 24 bytes represented as a hex-string (of 48 characters). Failure to adhere to this means the selection will revert to expensive client-side methods.

  Example:
  ```
  Session.createConsumer(
      destination,
      "JMSCorrelationID='ID:574d51373053616d706c65436f7272656c6174696f6e4944'");
  ```
  In this case, the hexadecimal represents a 24-byte ASCII string "WMQ70SampleCorrelationID"
- The safest way of generating a correct identifier is to use *JMSMessage.setJMSCorrelationIDAsBytes*. This allows the formatted version to be returned by *getJMSCorrelationID*. The number of bytes input should not be more than 24 or the identifier will be truncated.

  Example:
  ```
  Message.setJMSCorrelationIDAsBytes( "WMQ70SampleCorrelationID".getBytes("UTF8") );
  Session.createConsumer(
      destination,
      "JMSCorrelationID='" + message.getJMSCorrelationID() + "'" );
  ```
- A change to the *correlationId* (or indeed any selector) that you are matching against requires opening a new MessageConsumer and discarding the old one. This is an expensive operation if it is done for every message that is processed since it involves closing and re-opening the underlying queue. For this reason you should consider generating your own *correlationId* for each **client** rather than the common design pattern of using the *messageId* of a sent message as the *correlationId* of its reply. Another alternative is to use a temporary queue per client.

7.7 Other Programming Recommendations

- Use Non-Persistent, Non-Transactional messages whenever possible.
- Take performance into account when choosing which message type to use. The relative performance of the different JMS message types running a typical workload his as follows (fastest first)
  1. JmsTextMessage
  2. JmsBytesMessage (typically 5% slower than JmsTextMessage for a 2k message size)
  3. JmsObjectMessage (+10%)
  4. JmsStreamMessage (+15%)
  5. JmsMapMessage (+20%)
- If your application uses both transactional and non-transactional messages, consider creating separate transactional and non-transactional sessions for the different message types.
- Always call the *close()* method on JMS connection and session objects when they are no longer needed. This releases the underlying resource handle. This is especially important for publish-subscribe, where clients need to deregister from their subscriptions. Closing the objects allows the

queue manager to release the corresponding resources in a timely fashion; failure to do so can affect the capacity of the queue manager for large numbers of applications or threads.

- Do not lose references to connection and session objects (e.g. after registering an asynchronous listener) as this precludes being able to call their *close()* methods.
- To ensure an application or internal object will always tidy up correctly, including if it should fail, these *close()* calls should be made in the *final* part of a try-catch-finally control structure.
- Do not create sender or receiver objects regularly if you can reuse them instead. This avoids releasing then re-acquiring the same queue manager resource.
- Always call delete() on temporary queues and topics when they are no longer needed. Otherwise, they will not be deleted until the connection is closed. For long running applications this will cause performance and administration problems.

7.8 JMS Persistence

Several JMS settings control the effective QoS of a JMS client's communication. The delivery and acknowledgement modes indicate how many times a given message can be delivered to an application: at-most-once or once-and-only-once. The customer solution relies on a certain level of resilience from the messaging provider.

If the messages are carrying 'inquiry' questions and answers, then it is likely that speed is far more important than resilience, so the architects can make this trade-off and use non-persistent messages.

JMS delivery mode

The JMS API supports two delivery modes to specify what should happen to messages if the JMS provider fails.

The PERSISTENT delivery mode, which is the default, instructs the JMS provider that a message should not be lost in transit in case of a JMS provider failure. A message sent with this delivery mode is logged to stable storage when it is sent. Only a hard media failure should cause a PERSISTENT message to be lost. PERSISTENT has the caveat that it does not cover message destruction due to message expiration (which would be considered a normal event), or loss due to "resource restrictions" (which the JMS specification does not define further). PERSISTENT messages should not be lost during a controlled restart of a JMS provider but there are no guarantees of protection across an unexpected failure.

The NON_PERSISTENT delivery mode does not require the JMS provider to store the message or otherwise guarantee that it is not lost if the provider fails or is restarted - in fact NON_PERSISTENT messages should NOT be kept across a restart of a JMS provider.

JMS acknowledgement mode

The JMS API also supports the ACKNOWLEDGE_MODE property that controls message duplication on non-persistent messaging.

Auto acknowledgement (default) means messages will not be delivered more than once

DUPS_OK acknowledgement means messages may be delivered more than once in certain circumstances and the client application must be prepared to deal with seeing the same message twice.

Client acknowledgement leaves control of this feature entirely to the user.

WebSphere MQ Quality-of-Service

The JMS definition of persistence allows considerable scope for different quality of service (QoS).

WebSphere MQ provides QoS that have been appreciated by customers over the last 18 years. Many of the installation defaults provide robustness and small memory footprint rather than maximising good performance.

WebSphere MQ has traditionally provided two QoS: persistent and non-persistent. The WebSphere MQ definitions are similar, but not identical to the JMS requirements. In particular,

WebSphere MQ does not discard non-persistent messages while the queue manager is running, even in the event of a memory buffer shortage.

WebSphere MQ provides a persistence and transaction integrity, above and beyond the specification of JMS, which has been industry-proven for a decade.

These QoS are usually paired together with transactionality. If messages are persistent it is expected, though not required, that they should be transactional and if they are non-persistent, they should be non-transactional. Messages carrying 'valuables' should normally be persistent and transactional since that eliminates most causes of failure. The application and system designer needs to consider the levels of resilience and recovery needed in different places, and the complexity needed in each component - the application, the messaging provider, a database, and so on. Using persistent, transactional messaging can remove a lot of complexity from application code.

Non-persistent messages are discarded by WebSphere MQ in the event of a queue manager restart but otherwise are not lost. The discarding of non-persistent messages can be altered on an individual queue basis by specifying NPMCLASS=HIGH which tells the Queue manager to preserve non-persistent messages when a controlled shutdown and restart of the queue manager is undertaken. During a failure (hardware or software) messages may have been lost and because there is no message log, we cannot rebuild the queue with integrity. These uncontrolled failures are outside of the JMS definition of persistent messages. Consequently, because MQ does not discard non-persistent messages during resource shortages, MQ non-persistent messages qualify as JMS Persistent messages when they are stored on a queue marked with NPMCLASS=HIGH. This code fragment shows how Persistence is taken from the JMS destination/queue using the WMQ_PER_NPHIGH string, which tells WebSphere MQ that it should treat messages sent to that destination as JMS PERSISTENT messages, but that it can use its knowledge of the underlying Websphere MQ queue configuration to optimise performance by using WebSphere MQ non-persistent messaging where possible

```
// Create a connection factory
    JmsFactoryFactory ff = JmsFactoryFactory.getInstance(WMQConstants.WMQ_PROVIDER);
    JmsConnectionFactory cf = ff.createConnectionFactory();
//Add some connection factory configuration here to tell the application how to connect to WebSphere MQ
    connection = cf.createConnection();
    session = connection.createSession(false, Session.AUTO_ACKNOWLEDGE);
    destination = (JmsDestination) session.createQueue("queue:///Q2");
    destination.setIntProperty(WMQConstants.WMQ_PERSISTENCE, WMQConstants.WMQ_PER_NPHIGH);
```

WebSphere MQ's use of transactional recovery logs in combination with secondary storage of queues results in resilience against individual failures can be used for high availability and disaster recovery scenarios.

The JMS definition of a persistent message is not precise so application solutions must decide how much dependence is put on the message provider.

- Does the message have to survive if various resource shortages are encountered on the journey?
- Does the message survive if various application, software, or hardware failures are encountered on the journey?
- Greater reliability inevitably means lower run-time performance because of the extra work needed to provide the information needed during recovery.

8 Machine and Test Configurations

The JMS applications used to generate the performance data in this report are:

- Co-located on the server. (Local Measurements)
- Located on Linux clients communicating with the server. (Client measurements)

Clients are hosted by up to 4 Linux driver machines of varying powers. The driver machines are connected to the AIX and Windows machines over a 1Gb Ethernet LAN. The Linux64 machine is connected to its driver machines by a 10Gb LAN because it is a more powerful machine and the additional network bandwidth is required to enable the server to be driven to maximum CPU.

Due to the differences between the hardware used for each operating system, it is not possible to compare throughput across operating systems.

8.1 Linux64

An xSeries 5660 2 x 6-core hyperthreaded 2.80GHz Intel Xeon with 32GB of RAM was used as the Linux64 machine under test.

Linux64 Redhat 5.5 (kernel 2.6.18) with MQ Log and Queues on SAN disks on DS8700

8.2 AIX

A pSeries P5 8 CPU SMT-enabled with 16GB of RAM was used as the AIX device under test.

AIX 6.1.0.0 TL05 SP2 with MQ Log and Queues on SAN disks on DS8700

8.3 Windows

An xSeries 350 4 CPU hyperthreaded 2.80GHz Intel Xeon with 3.4GB of RAM was used as the Windows 2003 device under test.

Windows 2003 with MQ Log and Queues on 2 * local cache disks

8.4 SAN disk subsystem

The machines under test are connected to a SAN via a dedicated SVC. The SVC provides a transparent buffer between the server and SAN that will smooth any fluctuations in the response of the SAN due to external workloads. The server machines are connected via a fibre channel trunk to an 8Gb Brocade DCX director. The speed of each server is dictated by the server's HBA (typically 2Gb). 5GB generic LUNs are provisioned via SVC. The SVC is a 2145-8G4 which connects to the DCX at 4Gb. The SAN storage is provided by an IBM DS8700 which is connected to the DCX at 4Gb.

8.5 z/OS

CPU: 16-way LPAR on a 2817-779 (z196)
CPUs were dedicated
Its capacity is similar to that of a 2817-716.
DASD: DS8800's with dedicated links.

z/OS 1.12

Note that since the z/OS measurements were done using a 1Gb LAN the network was the bottleneck in many of the tests and hence the same or very similar throughput could be achieved using only 3 CPUs at lower cost per transaction.

Appendix JMS Performance Harness Commands

This appendix lists the commands used to drive the IBM Performance Harness for JMS for all the scenarios in the report. The examples below show the commands used for Client tests using non-persistent messaging. To change the commands to use Local mode change the –jt mqc flag to –jt mqb and remove the –jh myServerName as the default is local host. To change the commands to use Persistent messaging add the – pp and –tx flags.

Requester/Responder

java -ms128M -mx128M JMSPerfHarness -su -wt 10000 -nt 1 -ss 10 -sc BasicStats -ms 2048 -rl 0 -tc jms.r11.Requestor -co -to 30 -d QUEUE -mt text -jp 1420 -jc SYSTEM.DEF.SVRCONN -jh myServerName -jb myQueueManager -jt mqc -pc WebSphereMQ -jq SYSTEM.BROKER.DEFAULT.STREAM -ja 100

java -ms128M -mx128M JMSPerfHarness -su -wt 10000 -nt 2 -ss 0 -sc BasicStats -rl 0 -id 1 -tc jms.r11.Responder -cr -to 30 -d QUEUE -jp 1420 -jc SYSTEM.DEF.SVRCONN -jh myServerName -jb myQueueManager -jt mqc -pc WebSphereMQ

Publish/Subscribe Single Publisher, Many Subscribers

java -ms128M -mx128M JMSPerfHarness -su -wt 10000 -nt 1 -ss 10 -sc BasicStats -ms 2048 -rl 0 -tc jms.r11.Publisher -d TOPIC -mt text -db 1 -dx 1 -jp 1420 -jc SYSTEM.DEF.SVRCONN -jh myServerName -jb myQueueManager -jt mqc -pc WebSphereMQ -jq SYSTEM.BROKER.DEFAULT.STREAM -ja 100

java -ms128M -mx128M JMSPerfHarness -su -wt 10000 -nt 1 -ss 10 -sc BasicStats -rl 0 -id 1 -tc jms.r11.Subscriber -d TOPIC -db 1 -dx 1 -jp 1420 -jc SYSTEM.DEF.SVRCONN -jh myServerName -jb myQueueManager -jt mqc -pc WebSphereMQ

Publish Subscribe multiple

java -ms128M -mx128M JMSPerfHarness -su -wt 10000 -nt 2 -ss 10 -sc BasicStats -ms 2048 -rl 0 -rt 1600 -tc jms.r11.Publisher -d QUEUE -mt text -db 3 -jp 1420 -jc SYSTEM.DEF.SVRCONN -jh myServerName -jb myQueueManager -jt mqc -pc WebSphereMQ -jq SYSTEM.BROKER.DEFAULT.STREAM -ja 100

java -ms128M -mx128M JMSPerfHarness -su -wt 10000 -nt 2 -ss 10 -sc BasicStats -rl 0 -id 2 -tc jms.r11.Subscriber -d QUEUE -db 1 -jp 1420 -jc SYSTEM.DEF.SVRCONN -jh myServerName -jb myQueueManager -jt mqc -pc WebSphereMQ –ju

Point to Point multiple

java -ms128M -mx128M JMSPerfHarness -su -wt 10000 -nt 1 -ss 10 -sc BasicStats -ms 2048 -rl 0 -rt 1600 -tc jms.r11.Sender -d QUEUE -mt text -db 1 -jp 1420 -jc SYSTEM.DEF.SVRCONN -jh myServerName -jb myQueueManager -jt mqc -pc WebSphereMQ -jq SYSTEM.BROKER.DEFAULT.STREAM -ja 100

java -ms128M -mx128M JMSPerfHarness -su -wt 10000 -nt 1 -ss 10 -sc BasicStats -rl 0 -id 2 -tc jms.r11.Receiver -d QUEUE -db 1 -jp 1420 -jc SYSTEM.DEF.SVRCONN -jh myServerName -jb myQueueManager -jt mqc -pc WebSphereMQ -ju

Put/Get

java -ms128M -mx128M JMSPerfHarness -su -wt 10000 -nt 2 -ss 10 -sc BasicStats -ms 2048 -rl 0 -tc jms.r11.PutGet -d QUEUE -mt text -db 1 -dx 4 -dn 1 -jp 2201 –jc SYSTEM.DEF.SVRCONN -jh myServerName -jb myQueueManage -jt mqc -pc WebSphereMQ -jq SYSTEM.BROKER.DEFAULT.STREAM -ja 100

Details of Flags

-tc test definition class (jms.r11.PutGet, jms.r11.Publisher, jms.r11.Subscriber. jms.r11.Requester, jms.r11.Responder, jms.r11.Sender, jms.r11.Reciever)

-nt number of worker threads
-wt worker thread start timeout

-ms message size in bytes
-mt message type (text, bytes, stream, map, object, empty, ebcdic)

-rl run length in seconds (0 disables the timer and runs forever)
-rt desired rate in operations/sec

-d destination
-db multi destination numeric base
-dn multi desination numeric range
-dx multi destination numeric maximum

Examples of use of these options:
-d QUEUE
All threads operate on a destination named QUEUE
-d MYTOPIC -dn 3
destinations are distributed round-robin in the order MYTOPIC1..MYTOPIC3
-d MYTOPIC -db 6 -dn 3
destinations are distributed round-robin in the order MYTOPIC6..MYTOPIC8
-d MYTOPIC -dx 6 -dn 3
destinations are distributed round-robin in the order MYTOPIC4..MYTOPIC6

-jh provider host machine
-jp port of the provider host machine
-jb WMQ Queue Manager to connect to
-jc WMQ Channel to connect to
-jt transport , set to mqc for client bindings and mqb for local bindings
-jq publish queue
-ja publish acknowledgement interval
-ju use unique queue per subscriber

-pc provider class (WebSphereMQ)

-pp Use Persistent messages
-tx Transactionality (default is false)

-co use correlation id on requests (this sets the JMSCorrelationID)
-cr copy Request message to response (used on the Responder only)

-to polling interval when receiving synchronous messages

-su display final summary
-ss statistics reporting period
-sc statistics module to use
-id process identifier (if set this is displayed in the statistics reporting)