MQSeries Integrator for AIX V1.1 - Maximizing persistent message throughput

Version 1.0

7th February, 2000

Michael A. Perreault
IBM AIM Services
Middletown, RI
USA

Alan Hollingshead
IBM United Kingdom Laboratories
Hursley Park
Hursley
UK

Take Note!

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

First Edition, February 2000
This edition applies to Version 1.0 of *MQSeries Integrator for AIX V1.1 - Maximizing persistent message throughput* and to all subsequent releases and modifications unless otherwise indicated in new editions.

Table of Contents

Notices

This report is intended to help understand the performance characteristics of MQSeries Integrator for AIX V1.1, and the ability of the solution to scale. The information is not intended as the specification of any programming interfaces that are provided by MQSeries or MQSeries Integrator.

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 as is. The use of this information and the implementation of any of the techniques is the responsibility of the reader. Much depends on the ability of the reader 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:

- AIX
- DB/2
- IBM
- MQSeries
- MQSeries Integrator
- RS6000

Acknowledgments

The following individuals contributed in varied ways to developing the presented information.

Thomas S. Cloud
Matthew Courtois
Alicia Harvey
Alan Hollingshead
John Jones
Neil Kolban
Lyle Larson
Dave Moule
Michael A. Perreault
Bruce Spencer
Raymond St. Germain

(Alphabetically Listed)

Preface

This document outlines aspects of the MQSeries interface and MQSeries Integrator (MQSI) performance within the AIX environment as described within. It is intended to help persons who are investigating IBM MQSeries and MQSI implementations for positioning them within their installation's needs.

NOTE: The information in this publication is supplemental to, and not intended as the replacement of, any product documentation. This information was developed in conjunction with use of the environment specified, and is limited in application to those specific hardware and software products and levels.

The Audience

This SupportPac is designed for people who:

- Want to explore the performance considerations of MQSI for AIX V1.1 with MQSeries V5.0
- Will be performance testing MQSI for AIX V1.1 in order to provide Technical Support
- Will be designing and developing implementations that use MQSI for AIX V1.1

Users should have a general awareness of the RS6000, AIX, MQSeries, MQSI, and be familiar with the Persistent/Non-Persistent Message concept, as well as Unit of Work operation to get the best out of this SupportPac.

What is in this SupportPac

- A set of graphs showing the MQSI performance aspects in a variety of application configurations and implemented on various RS6000 configurations
- Interpretation of these graphs, and the implications for application design

Note:

The information in this SupportPac is **not** aimed at Capacity Planners. You should consult your IBM representative for information on this sort of sizing.

Introduction

Many companies are facing an environment where enterprise application integration has become increasingly important to their information technology operations. Additionally, the changing business environment requires a migration beyond Intra-Company integration, and often calls for Inter-Company integration. The use of MQSeries, MQSeries Integrator and MQSeries Workflow can bring many benefits to an Intra-Company integration, as they enable asynchronous processing and provide cross-environment facilities. This SupportPac provides information on the performance of the MQSeries Integrator as it transforms real-world transaction data within an RS6000/AIX environment.

Objective

> The Objective of this testing was to determine the application configuration and tuning that would maximize the throughput for *Persistent MQSeries messages* using MQSeries for AIX V5.0 CSD6 with the minimum of MQSI V1.1 processing.

The hardware configuration(s) were provided by the IBM Dallas RS6000 Benchmark Center, and were not specifically tuned nor optimized toward the MQSI application. Instead, a default installation of the software stack was utilized, and configuration and tuning of the application was utilized to achieve the best throughput performance in the installed environment.

Consequently, the results may not directly apply to your environment, but they should give some guidance on how the solution is likely to behave. You should verify these results in your own installation.

Test Environment

All of the tests were conducted on a single RS6000 machine with a common system software stack as described in the Physical Environment section of this document.

The *preliminary* tests were conducted to determine the optimum application configuration for the fully configured test machine, where 'optimum' was defined as the greatest message per second throughput rate. This involved varying the number of:

- Queue Managers
- MQSeries Input Queues
- MQSI Rules Engine Daemons

Once the optimum application configuration was found, the second set of tests were run to fine tune the applications to prevent them flooding the queues.

Subsequently, *final* tests were conducted where the hardware was reconfigured, reducing the available processors and memory, to a point where the performance degraded to an unacceptable level. These modifications are described in detail.

Test Model

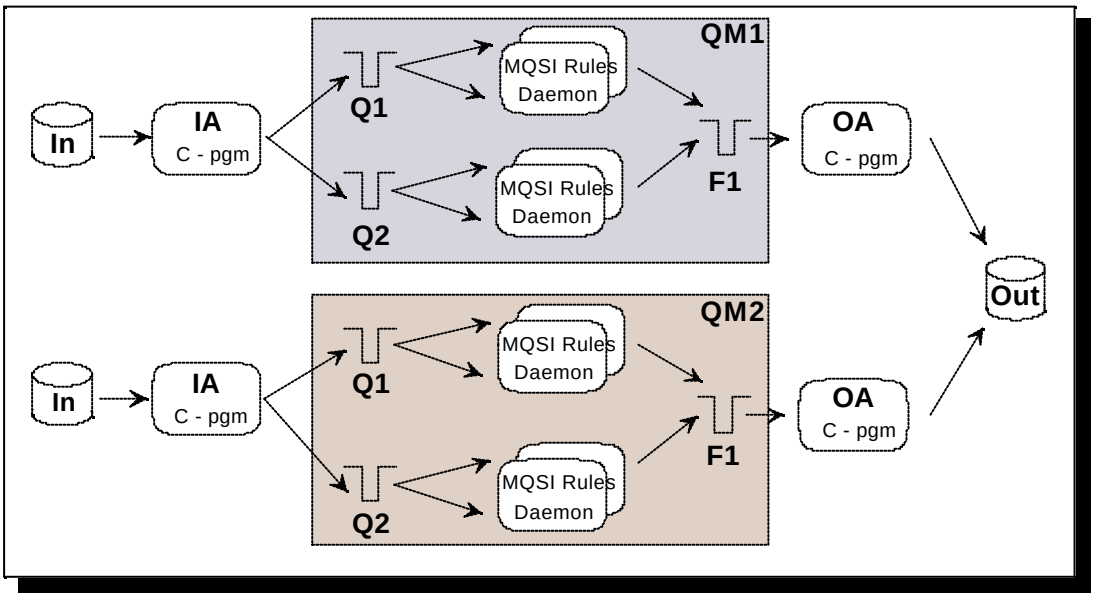


Figure 1 - Preliminary Test Model Diagram

Figure 1 illustrates the model that was used as a starting point for the testing. Each input file contained approximately one million variable length records, ranging in size from 115 - 485 bytes in length. The output file contained 226 byte, fixed-length records, each of which consisted of data fields that were transcribed from fields in the input file records. See **Messages** in Appendix A for more details.

The test model was processed as follows:

- The Input Adapter (IA) would connect to the appropriate queue manager (passed as parameter 1) and open the number of queues (parameter 2).
- Read a message from the input file (In).
- Alternately put a persistent message to an input queue (Q1 and Q2) in a round-robin fashion under syncpoint control (parameter 3 specified the number of records per commit).
- Start looping until reaching end-of-file.

- One or more instances of the MQSI rule engine would read the input queue, with the resulting output being written to a single MQSeries queue (F1) per Queue Manager (QM1 and QM2).

- The Output Adapter (OA) would read the output queue under syncpoint, and write the message to the output file (Out).
- A commit was issued after the appropriate number of messages, as passed in a program parameter.

The following MQSI environment variables were set:

- LogLevel was set to 3 in the .mpf file to log only fatal errors
- The environment variable NN_ALERT=OFF was set to prevent formatting errors from being logged

The settings of the above MQSI parameters should be carefully considered if used in a production environment and are only recommended for well-tested applications.

Hypothesis

Before testing began, it could have been assumed that:

- The more queue managers added, the greater the throughput
- That disk input/output would be a bottleneck
- Continually increasing the number of MQSI Rules Daemons per input queue would improve performance
- Increasing the number of “Rule Engines/Input Queue” groups per queue manager would improve performance

Physical Environment

Preliminary Environment

Hardware:

RS6000 Enterprise Server	
Model:	S80
Processor:	450 MHz, 64-bit RS64 III
L2 Cache:	8MB/proc
Architecture:	22way SMP
Memory:	64 gig
Disk:	3 internal SCSI drives - 9 gig each
	7133D40 SSA drawer of sixteen 9.1 Gig drives
	Looped into 4 loops with 4 drives each
	Two 32meg 6225 SSA adapters with fast write cache

Software:

Operating System:	AIX V4.3
Subsystem:	MQSeries for AIX V5.0 CSD 6
Application:	MQSI V1.1
Database:	DB/2 V5.2
Compiler:	IBM C Set ++ version 3

Each MQSeries queue manager had two dedicated SSA disks with fast/write cache enabled to instruct the disks to use the cache on the SSA adapters. One SSA disk was for the logfile logging the persistent messages, whilst the other was for the queues belonging to that queue manager.

Final Testing Environments

The above hardware environment was altered during the final testing phase to support the configurations in the table below:

		Amount of Physical Memory			
		64	32	16	8
Number of Processors	22	✓	✓	✓	✓
	12	✓	✓	✓	✓
	6	✓	✓	✓	✓

Figure 2 - Hardware Configurations

All other hardware and above mentioned software remained constant throughout the testing.

Test Cases

Preliminary Test Cases

The described test cases were executed on the fully configured machine (22 processors, 64 gig memory, etc...) to determine which application configuration would perform the best. The initial tests created a throughput of 100,000 MQSeries messages that were filtered from the 112,000 messages read by the input adapters. Once an optimum application configuration was found, one million, and then all two million input records were read by the input adapters (tests C2_250 and C2_500 respectively).

Scenario ID	Input Files	Total Input Records	Total MQSeries Messages	QMgr Count	MQSI Input Queues / Input Adapter	MQSI Rules Daemons / MQSI Input Queue	Relative Performance msg/sec
B2	2	112K	100K	2	2	2	351
B3	2	112K	100K	2	2	3	309
B4	2	112K	100K	2	3	3	259
B5	2	112K	100K	2	3	2	309
C2	4	112K	100K	4	2	2	555
C4	4	112K	100K	4	3	3	483
D2	6	112K	100K	6	2	2	472
D4 Note 1	6	112K	100K	6	3	3	438
C2_250	4	1.12 million	1 million	4	2	2	536
C2_500	4	2 million	1.71 million	4	2	2	520

Figure 3 - Preliminary Test Cases

The C2_500 test case represented the volume for our final objective. Two basic concepts of gaining throughput were explored:

- Expand the depth of processing, within each Queue Manager, by the addition of Input Queues and MQSI rule engines per Queue

- Expand the breadth of the system through additional Queue Managers, and therefore, additional Queues and MQSI Rule Engines

Note 1: Test Case D4 encountered an error during processing. The MQSI rule engines in QM6 failed to successfully start. The results that are presented are based on the performance of the five active Queue Managers. Due to time constraints and hardware availability, the test was not restarted.

Results

The best performance was offered by the configuration C2. This configuration would be drawn as in Figure 4, with an implementation utilizing: 4 Input Files, 4 Input Adapters, 4 Queue Managers, 2 Input Queues per Queue Manager, 2 MQSI Rules Engines per Input Queue, 1 Output Queue and Output Adapter per Queue Manager.

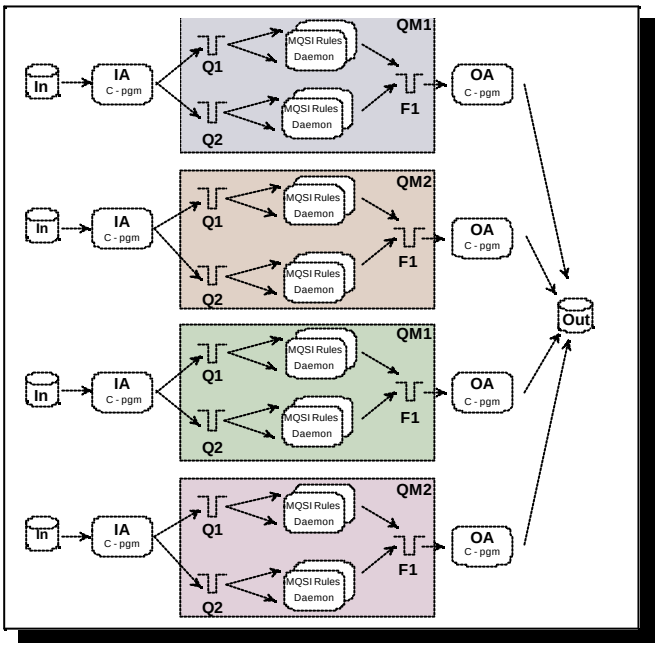


Figure 4 - C2 Configuration

The observed results are as follows:

		Observations					
		Msg. / Sec	AVG. CPU (Total)	IO Wait	Paging	AVG. File Utilization	Peak File Utilization
Test Case	B2	351	17.2	2.6	0	0	31.5
	B3	309	16.6	2.6	0	17.4	28.8
	B4	259	16.6	1.8	0	18.7	24.4
	B5	309	16.6	2.1	0	22.6	27.2
	C2	555	36.6	3.6	0	22.0	27.6
	C4	483	35.5	3.0	0	18.7	23.6
	D2	472	56.1	2.6	0	12.7	20.0
	D4 Note 1	438	45.9	2.8	0	16.8	26.4
	C2_250	536	35.2	3.9	0	22.7	29.2
	C2_500	520	34.4	4.2	0	24.1	34.0

Figure 5 - Preliminary Results

It became evident that there was a practical limit to the scalability of this solution, though not constrained by CPU, Memory nor Disk. In each of the test cases, the scenario was executed repetitively, and data captured by *vmstat* and *iostat* was used to determine if the scenario had exhausted any system resource (see Appendix A).

In no case did we determine that system resources were constraining the test, though the ability of the solution to scale is limited by some form of application contention.

Previously conducted testing indicated an increase in Messages/Second for a given test scenario, when the number of MQSI rule engines was increased from one to two per input queue and when adding an additional queue manager. The initial test cases for this study would begin with two queue managers, each configured with two input queues and two rules engines per input queue. (See Figure 1)

The initial series of test cases B2-B5 deepened the processing within the two queue manager configuration. In these test cases the effects of additional input queues and MQSI rule engines was tested. From B2 to B3 performance decreased when the number of MQSI rule engines were increased from two per input queue to three. A further decrease was noted in B4 when a third input queue with three additional rule engines were added. Test case B5 retained the three input queues, but reduced the number of rule engines to two per queue, and performance increased to a level identical to B2.

Was Our Hypothesis True?

Interestingly, the outcome of this test was that we received the best performance with a configuration of two input queues per queue manager, with two rules engines per input queue.

Following the B series of tests, the breadth of the configuration was increased by adding additional MQSeries queue managers. The initial test case was C2, with four queue managers utilizing two input queues and 2 rules engines per queue. This yielded a 58% increase in message throughput over the B2 test case. This configuration was extended further with the addition of two queue managers in test case D2. In this scenario performance degraded, yielding a loss of 15% in message throughput compared to C2. This same pattern of performance behavior was evident when comparing B4, C4 and D4.

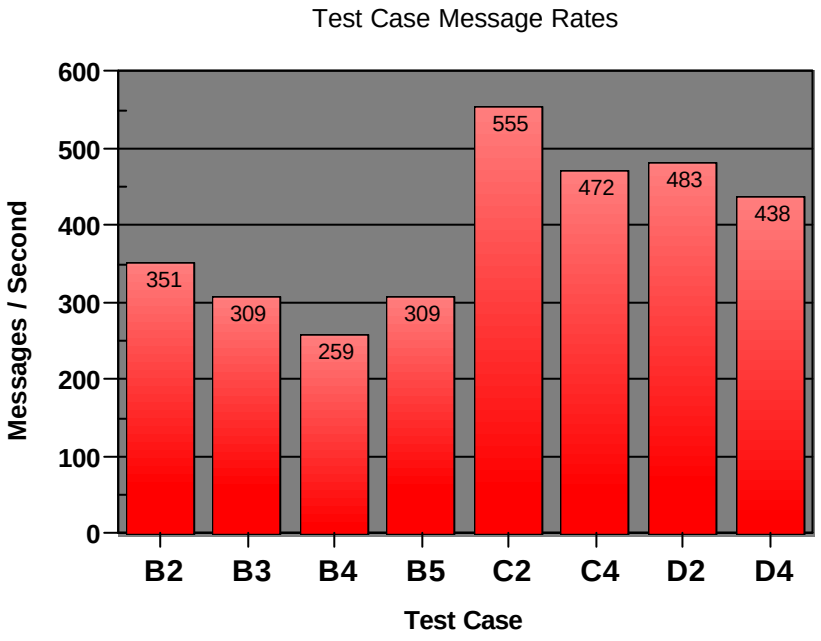


Figure 6 - Message Rate Graph

Application Parameter Tuning

Settings

Next a series of application tuning parameters were adjusted. In these test cases, we modified the identified parameters within the C2 test scenario.

Scenario ID	Input Files	Messages Processed	Qmgr Count	Input Queues / Adapters	Rule Eng. / Input Queue	Msg/Sec
C2	4	100K	4	2	2	555
C2t3_40	Altered Input Adapter from 1 to 40 recs. per LUW					672
C2t3_100	Altered Input Adapter from 40 to 100 recs. per LUW					690
C2t3_200	Altered Input Adapter to 100 to 200 recs. per LUW					689
C2t1_25	Altered MQSI MaxHandles=25					740
C2t1_100	Altered MQSI MaxHandles=100					740
C2t1_50	Altered MQSI MaxHandles=50					740
C2t5	Altered Output Adapter from 1 to 100 recs. per LUW					909
C2t5_500	2 Million records, 1.72 million messages					835
C2t6	Altered MQ maxdepth on input queues = 5k from 100k					1,060
C2t6_500	2 Million records, 1.72 million messages					914

Figure 7 - Tuning Parameter Results

Notes:

- The highlighted parameters represent the values that were retained through subsequent tests.
- An LUW (Logical Unit of Work) within the input adapter consisted of a number of MQPUTs under syncpoint to the MQSI input queue before the data was committed via an MQCMIT. For the output adapter, an LUW consisted of a number of MQGETs under syncpoint from the MQSI output queue before the data was committed to the output file.
- The MQSI MaxHandles parameter specifies the maximum number of entries allowed in the output queue handle cache. If MaxHandles is not specified (which is the default), the MQSI daemon disables its cache.

Results

As exhibited above, this combination of application tuning parameters increased the throughput rate by 90%. Due to time and machine availability restrictions, we were not able to rerun the complete B, C and D series of preliminary tests.

After determining the best configuration of queue managers, queues, and rule engines, we found that pacing all processes to run evenly over time (versus completing individual process segments as quickly as possible) appeared to yield the most optimal results.

The pacing was achieved through the altering of the input queue depth from a maxdepth of 100,000 to 5,000. During the execution of the tests, the IA would load records to the queues faster than the rules engines could remove them. Once the IA had finished writing all of the input records to the input queues, the rate at which the rules engine would process and remove input messages increased measurably. This same behavior was observed on the output message queue. We believe that this is an indication of some type of contention surrounding queue access. By lowering the maxdepth, and altering the IA program to sleep for 5 seconds on an MQSeries Q_Full return code, we reduced the contention on the queue and increased our throughput by greater than 10% (i.e. from 909 msg/sec to 1060 msg/sec).

Final Testing

Downsizing Tests

The final phase of the testing involved an incremental reduction in both available CPUs and physical memory. This phase of testing was conducted to determine the smallest hardware configuration that would achieve satisfactory results.

Hardware Variations

Summary of the Hardware Variations				
Scenario ID	MQ Messages	CPU Count	Physical Memory GB	Relative Performance msg/sec
C2t6	100K	22	64	1,060
C2t6_1	1.72 million	22	32	850
C2t6_2	100K	22	16	893
C2t6_3	100K	22	8	893
C2t6_4	100K	12	64	943
C2t6_5	100K	12	32	943
C2t6_6	100K	12	16	943
C2t6_7	**100K**	**12**	**8**	**943**
C2t6_8	100K	6	64	870
C2t6_9	100K	6	32	870
C2t6_10	100K	6	16	877
C2t6_11	100K	6	8	877
C2t6_12	1.72 million	6	8	838

Figure 8 - Hardware Variation Results

Results

- The performance across these tests remained satisfactory.
- The performance of tests C2t6_2 and C2t6_3 dropped unexpectedly and are unexplained. The utilities *vmstat* and *iostat* do not indicate any system resource shortage.
- Tests C2t6_8 to C2t6_12 were CPU Bound with an average CPU of 96.1 percent.
- The **best results** came from test **C2t6_7** with an average CPU 68.1 percent and a maximum CPU of 72 percent, iowait of 2.9 percent and no paging.

Conclusions

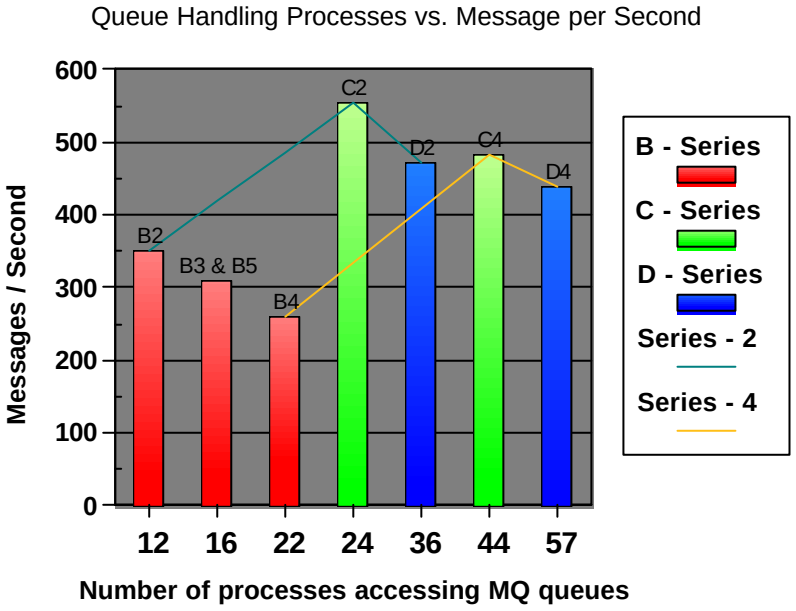
The following conclusions were gathered from the tests.

Process Count

An interesting metric is the number of processes that are engaged in MQ interface activity. In each series of tests, the number of Input Adapters and Output Adapters remained static, while the number of active rule engines varied. Note that the highest performance rate was obtained with the least number of MQ engaging processes, per queue manager.

Scenario	IA	OA	Rule Engines	Total	# QMgrs	Processes per QMgr	msg/sec
B2	2	2	8	12	2	6	351
B3	2	2	12	16	2	8	309
B4	2	2	18	22	2	11	259
B5	2	2	12	16	2	8	309
C2	4	4	16	24	4	6	555
C4	4	4	36	44	4	11	483
D2	6	6	24	36	6	6	472
D4 Note 1	6	6	45	57	6	~10	438

The best performance, C2, was obtained through a combination of additional rules engines and additional queue manages. Additionally, the solution does not continue to scale linearly. Adding additional processes within a queue manager, or additional queue managers in the solution caused performance to degrade, while not constrained by system resources. The relative performance of each scenario is depicted in the following chart.



The connected points on the chart (Series - 2 and Series - 4) depict the performance curve when incrementally adding additional queue managers. Notice there was a performance gain in the C - Series (incrementing from two to four Queue Managers) but performance degraded in the D - Series (incrementing from four Queue Managers to six Queue Managers).

Considerations

The following considerations should be noted:

- The operating system software was not tuned specifically for this application, and we believe that performance could be improved with a well tuned environment.
- The use of SSA (Serial Storage Architecture) disks with fast/write cache enabled (to instruct the disks to utilize the cache on the adapters) is an important performance criteria when disk I/O is high (as is the case with MQSeries persistent messages).
- All of the messages use MQSeries persistence and the tested configurations would not necessarily be the optimum configuration for non-persistent message processing. A final run was made in the C2t6_12 environment with non-persistent messages. The throughput was 1300 msg/sec but the run was CPU bound and frequently hit a maximum CPU of 100%.
- After the tests had concluded, we were informed that the LogBufferPages setting in the Log stanza of the QM.INI could have been increased from the default setting of 17 to 32. In this particular scenario it would have further improved disk I/O.

Finally

This testing was not optimally conducted, but was executed to satisfy a short turnaround for a sales prospect. We believe that additional performance gains can be achieved, and that the numbers in this report should not be viewed as a capacity ceiling. Instead they should be viewed as an indicator of the solutions performance potential. Additionally, this report provides some insight into the affect of the solution configuration, and the affect of the identified application tuning parameters.

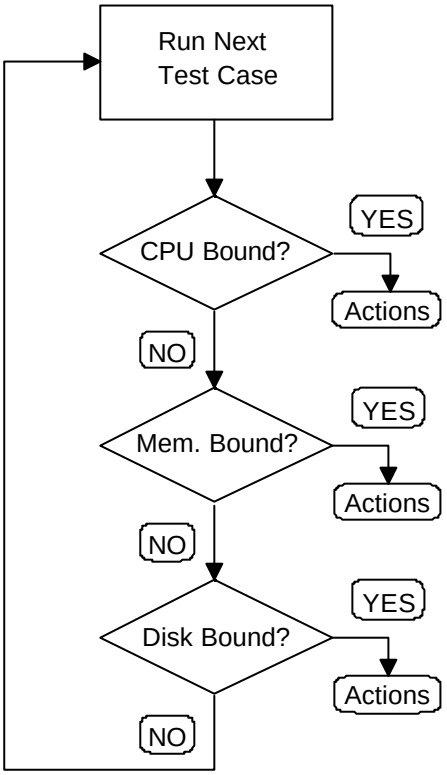
Appendix A - Performance Tuning

Performance Goals

- Maximize persistent message throughput of MQSeries Integrator for AIX V1.1 with MQSeries V5.0 CSD 6.

Approach

The solution testing was conducted in a series of tests as described in the Test Cases Section. Vmstat and iostat data was captured during the test case execution, and reviewed after each test case. Our approach was as follows:



Our decision of being resource bound was based on the following limits:

- CPU Bound if avg. CPU > 70% or max. CPU > 90% or 50% of the time CPU > 85%
- Memory Bound if < 2 free frames per meg., pagein rate (rmss) > 5/sec,
- Disk Bound if iowait >40% or avg. util > 50% for all disks

Note:

Tuning Approach, including flowchart and limit guidelines were drafted from *AIX Versions 3.2 and 4 Performance and Tuning* at:
http://www.rs6000.ibm.com/doc_link/en_US/a_doc_lib/aixbman/prftungd/toc.htm

Tuning Parameters

The following table summarizes the tuning parameters used during the testing:

Parameter	**Product**	**Default Value**	**Optimal Value for this Configuration**
LogLevel	MQSI	0 - Logs everything	3 - Logs fatal errors only
NN_ALERT	MQSI	Unset - logs all Alert messages to NEONetStatusLog	NN_ALERT=OFF
Syncpoint_Count	Input Adapter	0 - syncpoint after each MQPUT	100 - syncpoint after 100 MQPUT requests
Syncpoint_Count	Output Adapter	0 - syncpoint after each MQGET	100 - syncpoint after 100 MQGET requests
MaxHandles	MQSI	0 - cache disabled	50 - cache enabled
MaxDepth	MQ	5,000	5,000 Other values tested

Messages

There were three types of input message used in this performance test, the majority of which were message type 1. For each message type, a single rule checked for the existence of a field.

Message Type	% of all input records	Size of input record	# fields in input record	Size of output record	# fields in output record	# repeating field formats
1	91%	~140 bytes	37	226 bytes	34	1
2	4%	~280 bytes	45	226 bytes	34	2
3	5%	~420 bytes	53	226 bytes	34	3

End of document