FP11: WebSphere MQ File Transfer Edition V7.0.1 for System Z

Performance Evaluation

December 2009

Tony Sharkey

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

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 2009.
This edition applies to WebSphere MQ File Transfer Edition for System Z V7.0.1 (and to all subsequent releases and modifications until otherwise indicated in new editions).

Notices

DISCLAIMERS
The performance data contained in this report were 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 NONINFRINGEMENT, 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 File Transfer Edition V7.0.1. 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 File Transfer Edition V7.

Table of Contents

Overview

WebSphere MQ File Transfer Edition is a managed file transfer product that uses WebSphere MQ as its transport layer.

This is the first performance report on System Z and as such, there is no comparison to make between versions.

This performance report details WebSphere MQ File Transfer Edition in a range of scenarios, giving the reader information on transfer rates and costs.

This report is based on measurements taken from the WebSphere MQ Performance Sysplex that consists of 3 logical partitions, plus an internal Coupling Facility on a z10-EC64 running z/OS version 1.11.

The data costs are gathered from Resource Management Facility (RMF) reports and are based on the total cost across the Sysplex.

These measurements were run in a controlled environment where only required subsystems were active during the measurements.

Performance Headlines

The measurements for the performance headlines are based upon the time taken to transfer a set of files and the associated CPU cost.

Typically we measure transporting approximately 1 Gigabytes worth of data whilst varying the size of the files as follows:

- 1MB
- 10MB
- 100MB
- 1024MB

To illustrate a typical test, if a test is using 1MB files, then 1024 files will be moved in a single performance run. Varying the file size but keeping the overall data volume transferred demonstrates the cost of the open and close file operations on transfer time and CPU cost.

This document provides information as to the achieved transfer rate and cost of transferring the data on our systems for a number of configurations as well as offering advice on what to look out for on your system.

The document also provides information of WebSphere MQ File Transfer Edition (hereafter known as MQFTE) and some of the tuning options that may affect data transfer rates.

For messages of 1MB we achieved data transfer rates between 20-25MB/Second. By altering the MQFTE tuning options we were able to sustain transfer rates of 30MB/Second.

For messages of 10MB and greater we achieved data transfer rates of 52MB/second over a single MQ channel.

MQFTE cannot match the performance characteristics of the FTP protocol as there are acknowledgement flows from the receiving Agent to the sending Agent which allows MQFTE to take checkpoints which aid recovery from failure. Despite this, we are able to transfer data at 50% of the rate of FTP for binary transfers and using multiple concurrent transfers over an MQ channel for text transfers we are able to exceed FTP’s transfer rate from Linux to System Z.

How does MQFTE perform “out of the box”?

To demonstrate the performance characteristics of MQFTE on z/OS, the initial measurements are run using default options.

There are 3 basic configurations used. In each case, the MVS systems have been defined to use 3 dedicated CP-type processors.

The measurements shown later include writing to MVS datasets and writing to ZFS files. Measurements are primarily to z/OS from a single remote machine, although there are measurements both for z/OS to a remote machine as well as z/OS to z/OS.

In each case we aim to move 1GB’s worth of data.

4 sizes of files are used – 1MB, 10MB, 100MB and 1GB.

Both binary and text transfers are measured. The cost of the text transfer includes translating the data into the appropriate code page for the receiving machine.

The costs reported are the costs across the Sysplex as determined from the RMF reports.

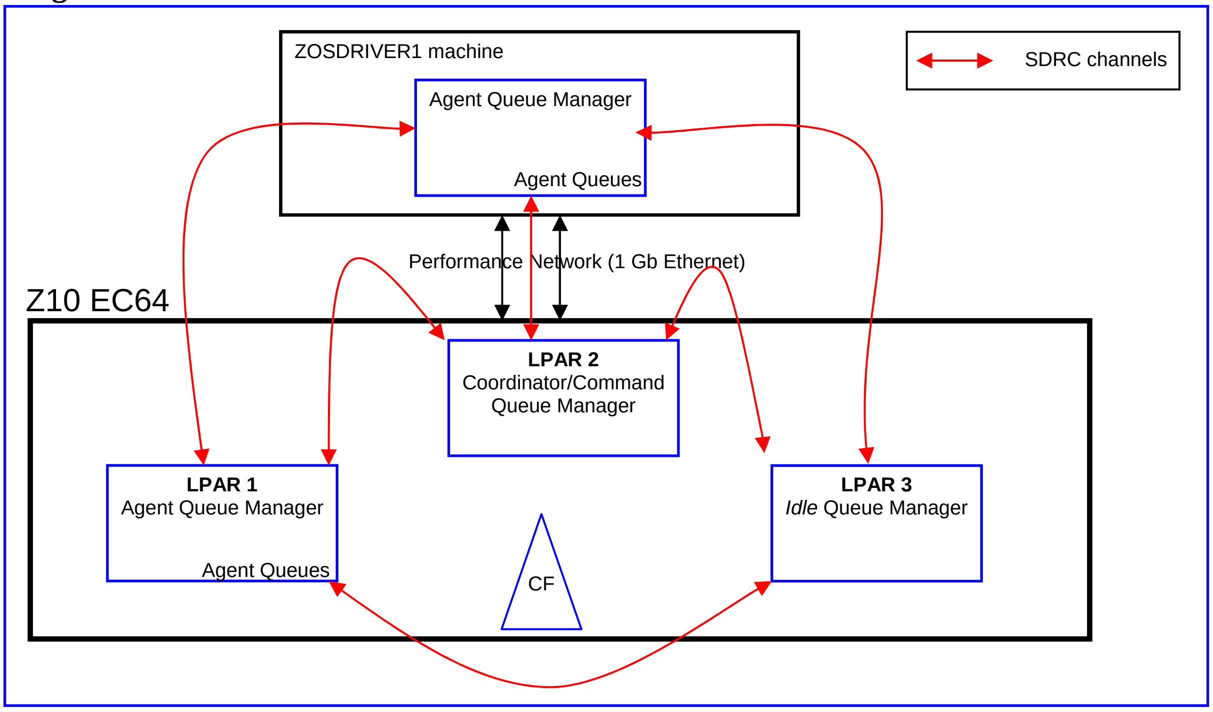
The following scenarios are measured:
1. Private queues using Sender-Receiver channel pairs
2. Private queues using Cluster-Sender / Cluster-Receiver channel pairs.
3. Shared queues using Sender-Receiver channel pairs.

In these baseline measurements, all transfers are single threaded

Private Queues using Sender-Receiver Channel Pairs

The following diagram shows the configuration used when the agent queues are defined as private queues and the queue managers used sender-receiver channels to send the files across the network.

Diagram 1



The significance of the Sender-Receiver channels between LPARs 1, 2 and 3 are that they are hitting the network cards, and indeed LPAR 1 and 2 are on a different subnet to LPAR 3.

In the following measurements, each MVS LPAR was defined with 3 fixed CP-type processors only.

The channels were started with default options in all cases except the DISCINT attribute was set to 0 to ensure channels did not disconnect.

Inbound to System Z

Chart 1

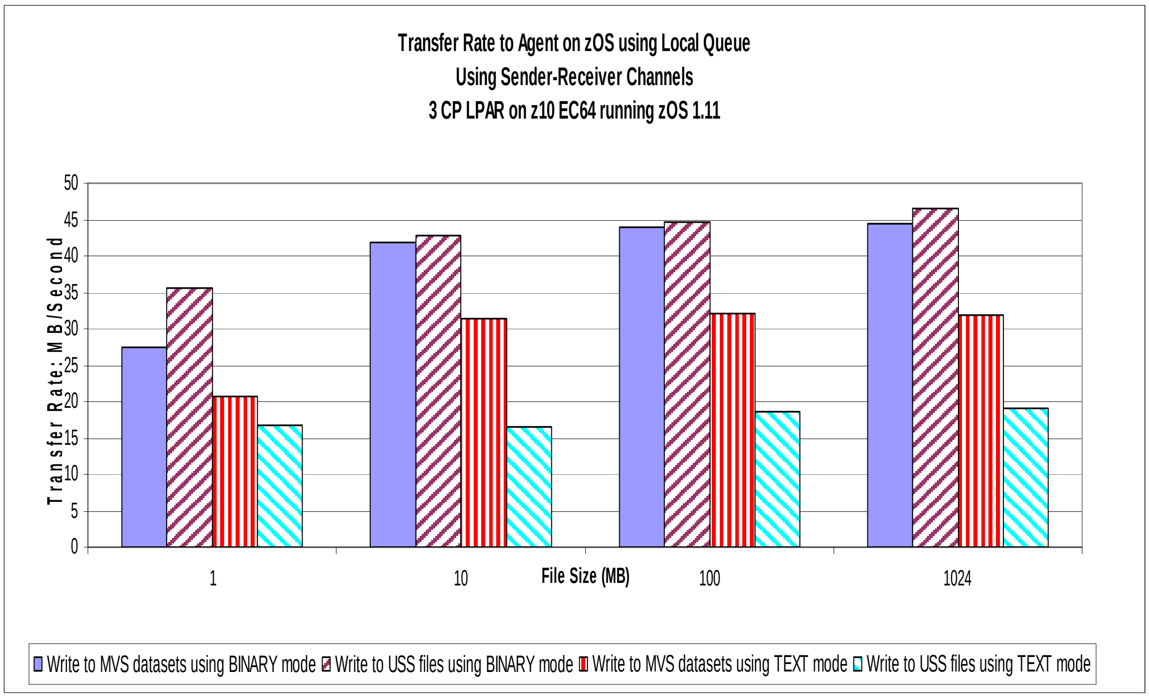


Chart 1 shows the achieved transfer rate when transferring files from the Linux machine into System Z.

Chart 2

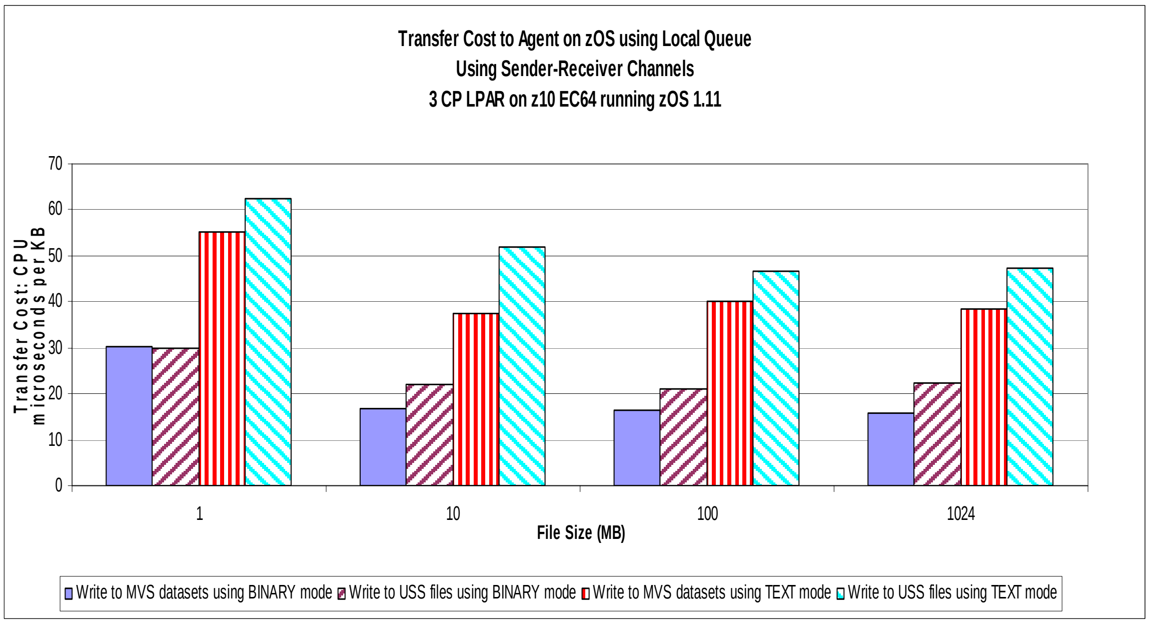
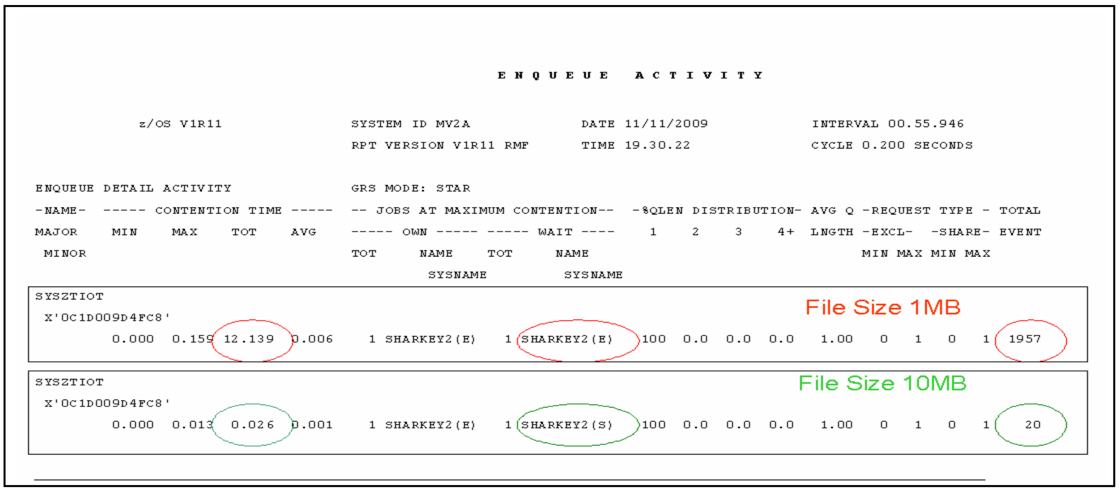


Chart 2 shows the actual transfer cost across the Sysplex when transferring files from the Linux machine into System Z. The costs are measured in microseconds per KB of data transferred.

As can be seen in chart 1, the rate at which data can be transferred in binary mode exceeds the text transfer mode quite significantly.

Chart 2 shows the additional translation cost incurred when converting the transferred data from ASCII to EBCDIC.

Chart 3



```
                                          E N Q U E U E   A C T I V I T Y

           z/OS V1R11                SYSTEM ID MV2A            DATE 11/11/2009           INTERVAL 00.55.946
                                     RPT VERSION V1R11 RMF     TIME 19.30.22             CYCLE 0.200 SECONDS

ENQUEUE DETAIL ACTIVITY              GRS MODE: STAR
-NAME-   ----- CONTENTION TIME ----- -- JOBS AT MAXIMUM CONTENTION--  -%QLEN DISTRIBUTION- AVG Q -REQUEST TYPE - TOTAL
MAJOR     MIN     MAX     TOT    AVG ----- OWN ----- ----- WAIT ----    1     2     3    4+ LNGTH -EXCL-  -SHARE- EVENT
 MINOR                               TOT      NAME     TOT     NAME                               MIN MAX MIN MAX
                                              SYSNAME          SYSNAME
SYSZTIOT                                                                                 File Size 1MB
 X'0C1D009D4FC8'
          0.000   0.159  12.139  0.006    1 SHARKEY2(E)    1 SHARKEY2(E) 100   0.0   0.0   0.0   1.00   0   1   0   1  1957

SYSZTIOT                                                                                 File Size 10MB
 X'0C1D009D4FC8'
          0.000   0.013   0.026  0.001    1 SHARKEY2(E)    1 SHARKEY2(S) 100   0.0   0.0   0.0   1.00   0   1   0   1    20
```

The above chart is an extract of the Enqueue Activity report from RMF for the binary transfer into MVS datasets for files of size 1MB and 10MB.

The sysname "SHARKEY2" is the process that is running the Agent task.

As can be seen, there is a significant number of enqueues when processing smaller files – 1957 compared to 20 enqueues when processing 10MB files.

Also of note is the total contention time waiting for enqueue SYSZTIOT[1] – for 1MB files this is 12.139 seconds within the interval of 55.946 seconds.

This is a significant factor in the slow transfer of smaller files.

[1] SYSZTIOT is related to disk I/O.

Outbound From System Z

Chart 4

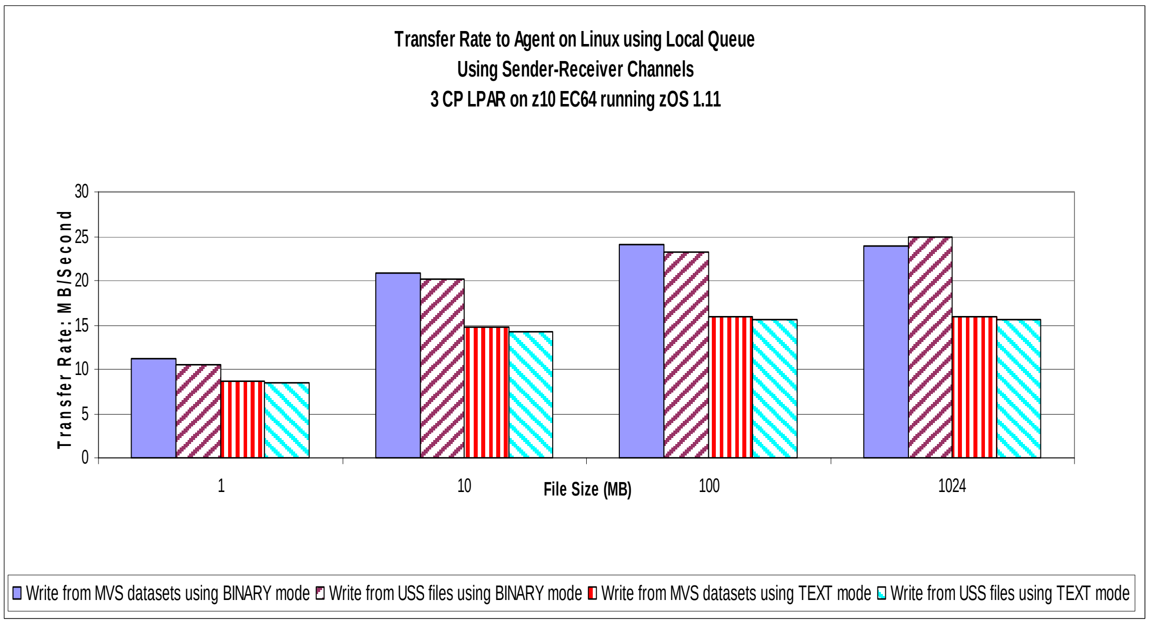


Chart 4 shows the achieved transfer rate when sending files from MVS to the Linux machine across the Performance network.

Note the rate of the files being sent to the Linux machine is being constrained by using slow disks. With faster disks on the Linux machine, such as via a SCSI interface to a SAN, we would expect a higher write rate.

Chart 5

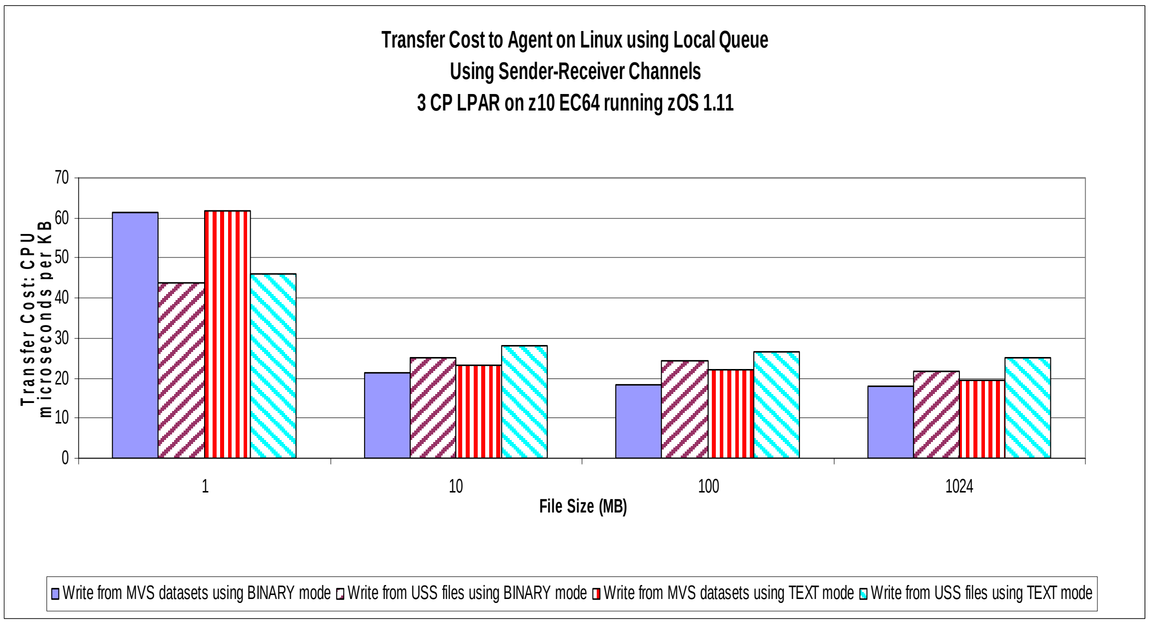


Chart 5 shows the actual transfer cost in microseconds per KB of data transferred from MVS to the Linux machine across the Performance network.

In a Text file transfer, any data conversion is performed by the receiving side.

System Z to System Z

In this scenario, we measure the transfer rate and cost between Agents running on LPAR 1 and LPAR 3.

A total of 1GB of data is moved for each measurement – the files in each measurement are fixed and are of size 1MB, 10MB, 100MB or 1GB.

Each LPAR has 3 dedicated CP-type processors

The 2 LPARs hosting Agents are on different subnets of the network and the MQ channels are configured such that they go out to the network.

MVS dataset file transfer has been used to avoid the additional XCF signalling to LPAR 2 – which is the root of the shared ZFS.

Chart 6

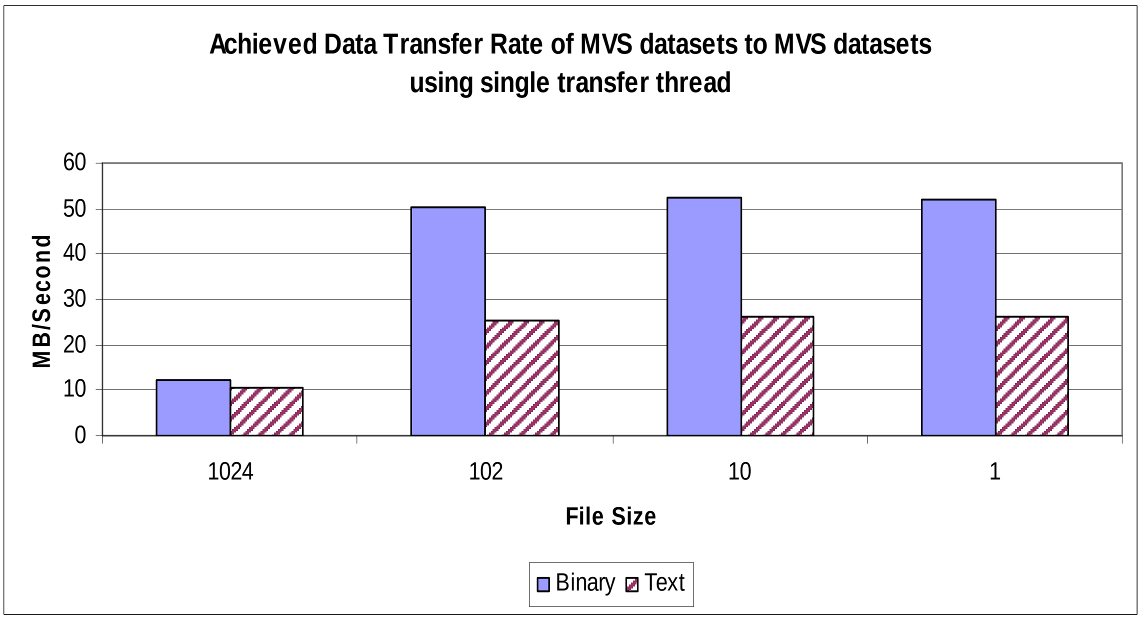


Chart 6 shows that with MVS to MVS transfers, the maximum throughput achieved is 52MB/second.

Text-type transfers show similar throughput reductions due to the additional work involved in translating the contents of the file as seen in the System Z Inbound measurements.

Chart 7

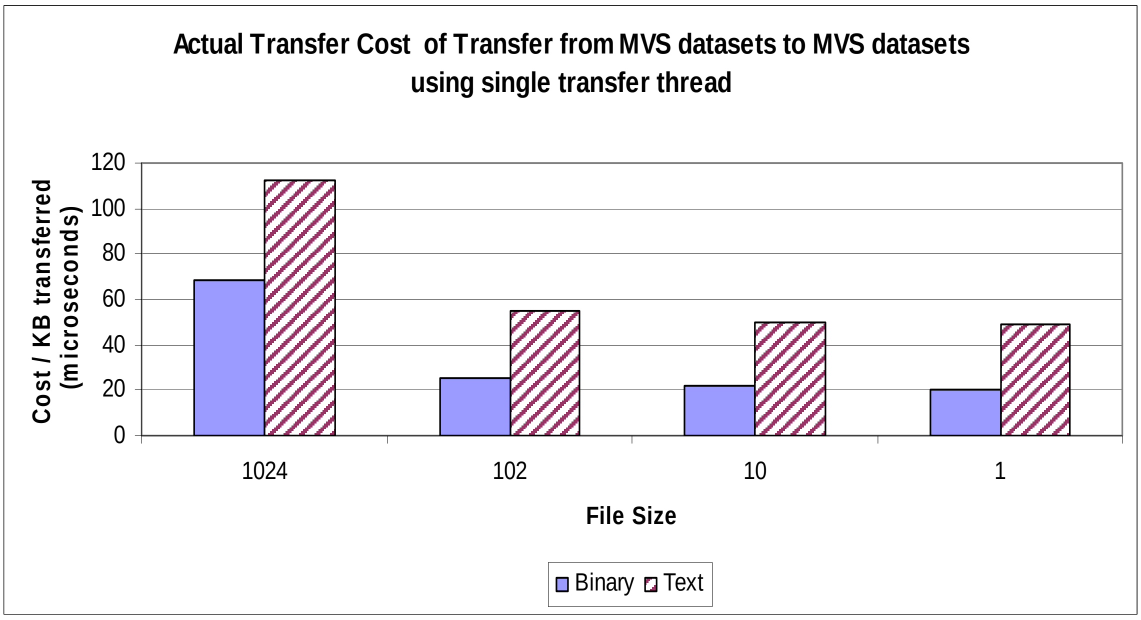


Chart 7 shows the total cost to the Sysplex of performing a file transfer from one LPAR to another.

This chart highlights the cost of attempting to translate the files when they are already in the correct code page.

Consider whether a text transfer is required between Agents!

Chart 8

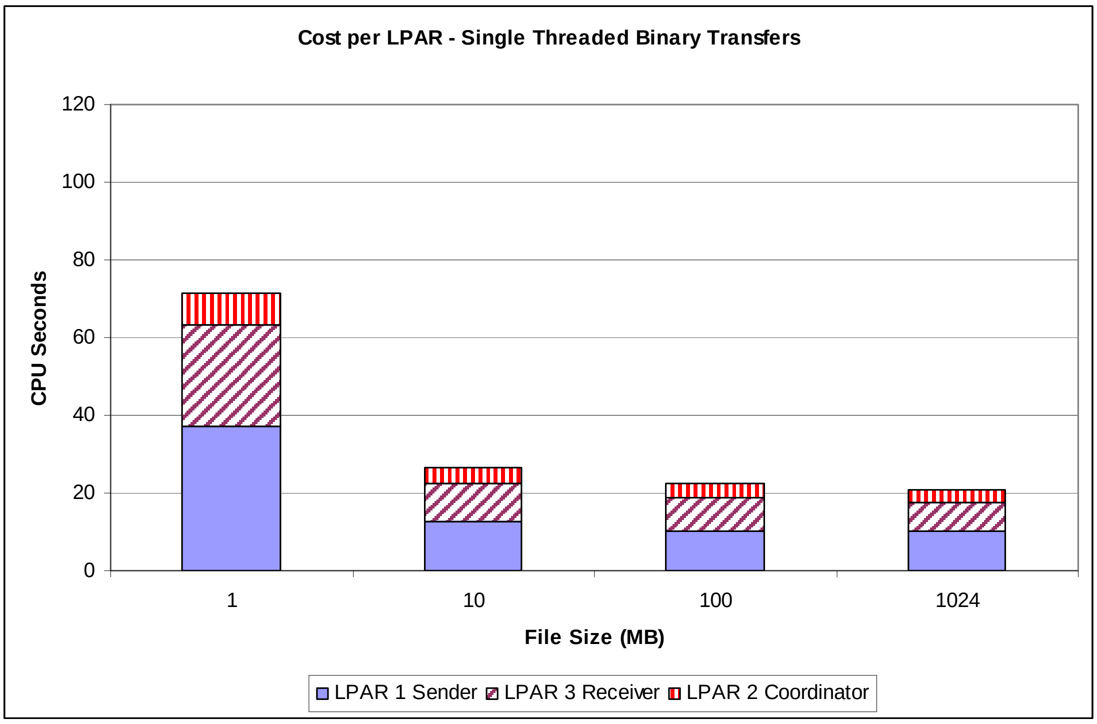
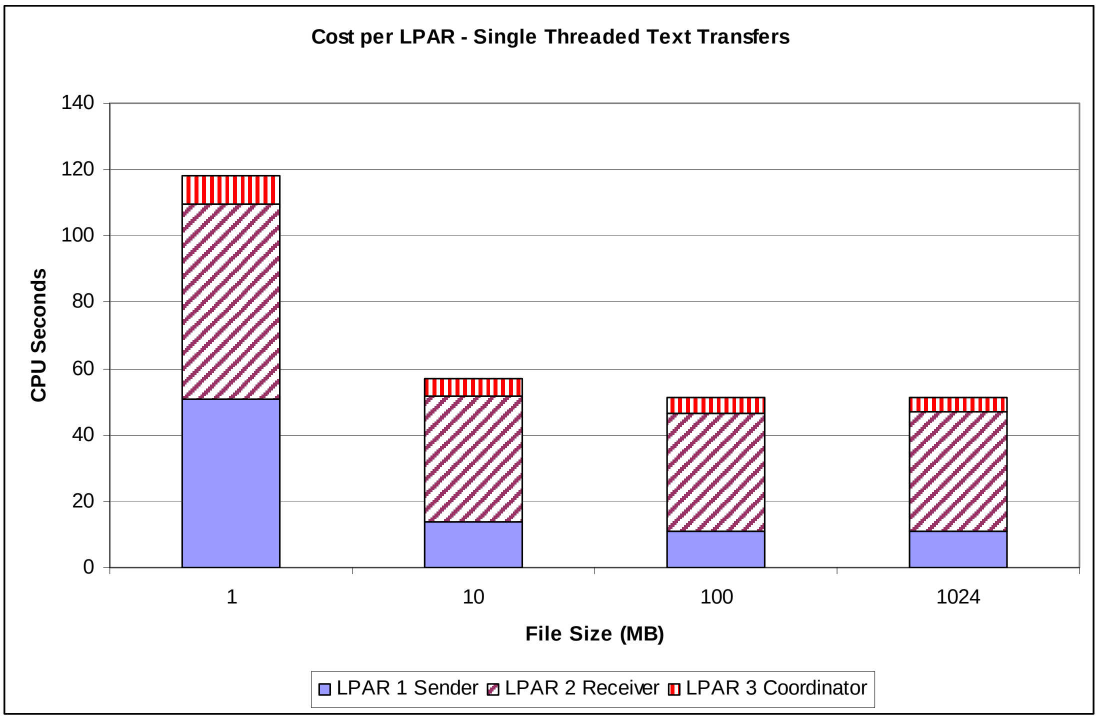


Chart 9



Charts 8 and 9 shows the total cost per LPAR when moving 1GB of files between 2 agents as per diagram 1 using binary and text mode transfers.

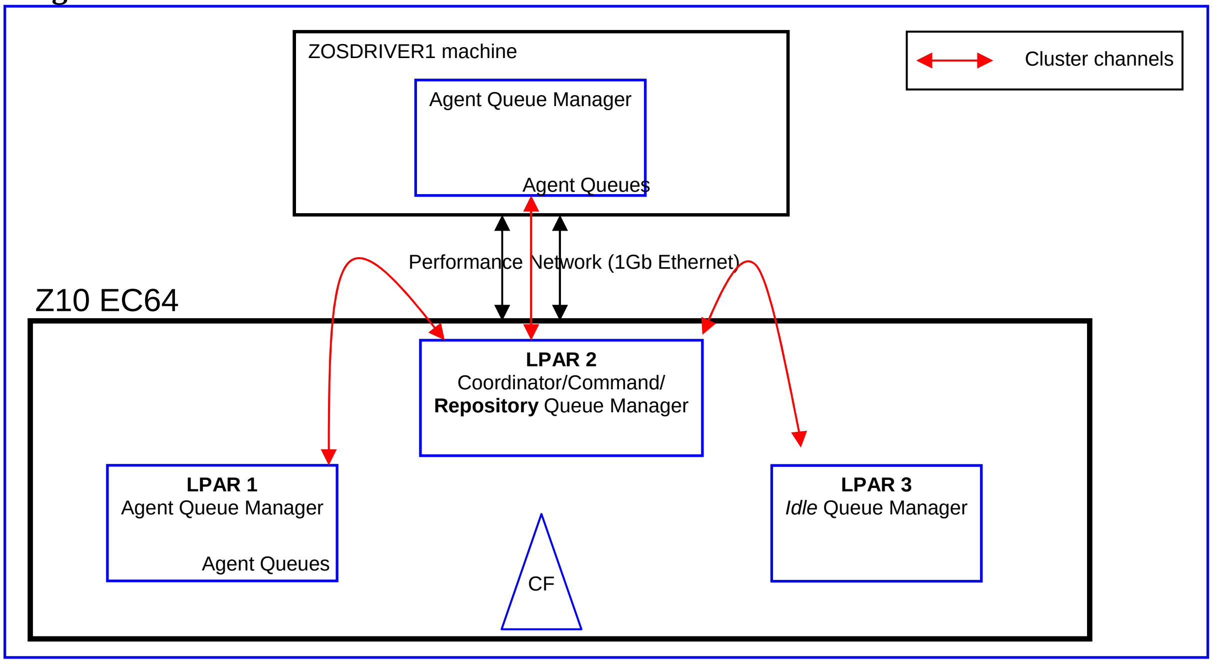
For binary mode transfers the cost attributed to the LPAR with the sending Agent is similar to the cost attributed to the LPAR with the receiving Agent.

For text mode transfers the cost attributed to the LPAR with the receiving Agent is between 2 and 4 times the cost attributed to the LPAR with the sending Agent.

Private Queues using Cluster Channels

The following diagram shows the configuration used when the agent queues are defined as private queues and the queue managers used cluster channels to send the files across the network.

Diagram 2



Only the cluster channels shown by the red lines with arrows are defined manually. WebSphere MQ's clustering technology auto-defines channels between the queue manager on the Linux machine and the queue manager on LPAR 1.

Chart 10

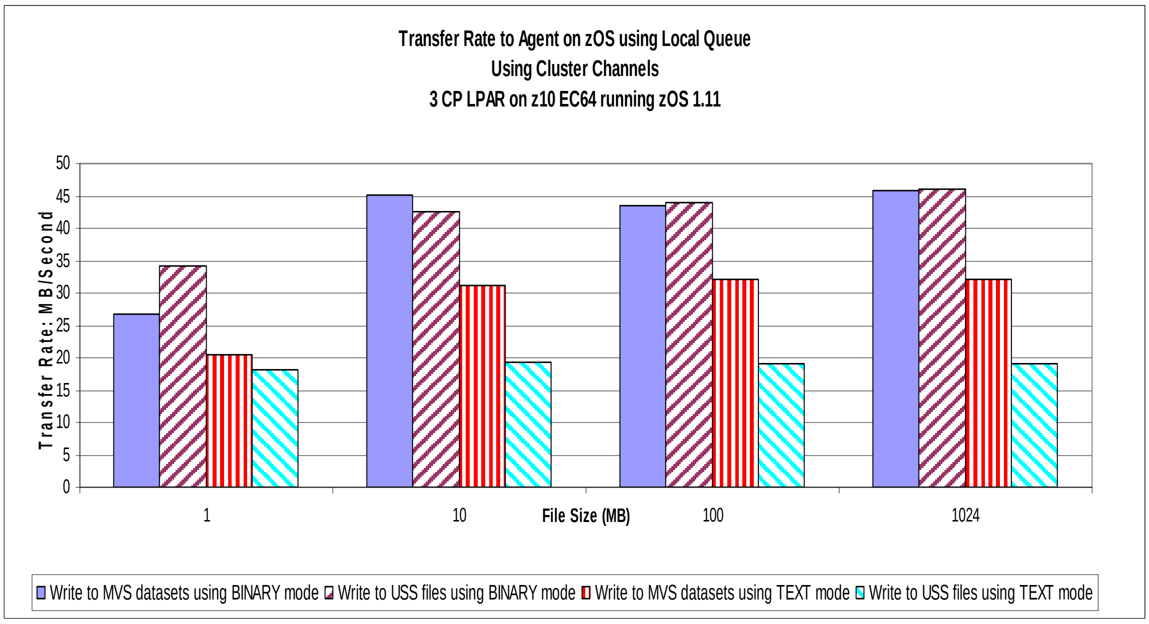


Chart 10 shows the achieved transfer rate in MB per second when transferring data from Linux to MVS over the performance network.

Chart 11

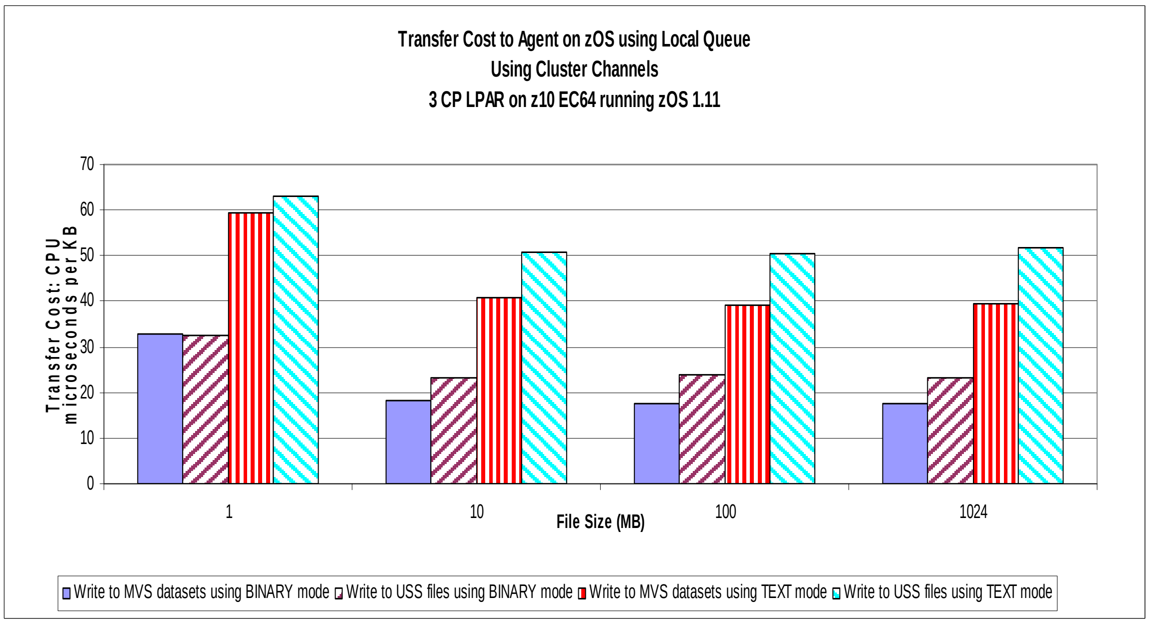


Chart 11 shows the actual cost across the Sysplex in microseconds per KB of data transferred from Linux to MVS.

Cluster Channels or Sender-Receiver Channels?

As can be seen when comparing chart 1 with chart 8, the achieved transfer rate is similar, as is the cost per KB transferred (as seen in charts 2 and 9), so is there any benefit in using one channel type over another?

The simplest answer is that it depends!

Cluster channels offer less administration – there is no need to make additional channel definitions or transmission queues between each of the agent queue managers.

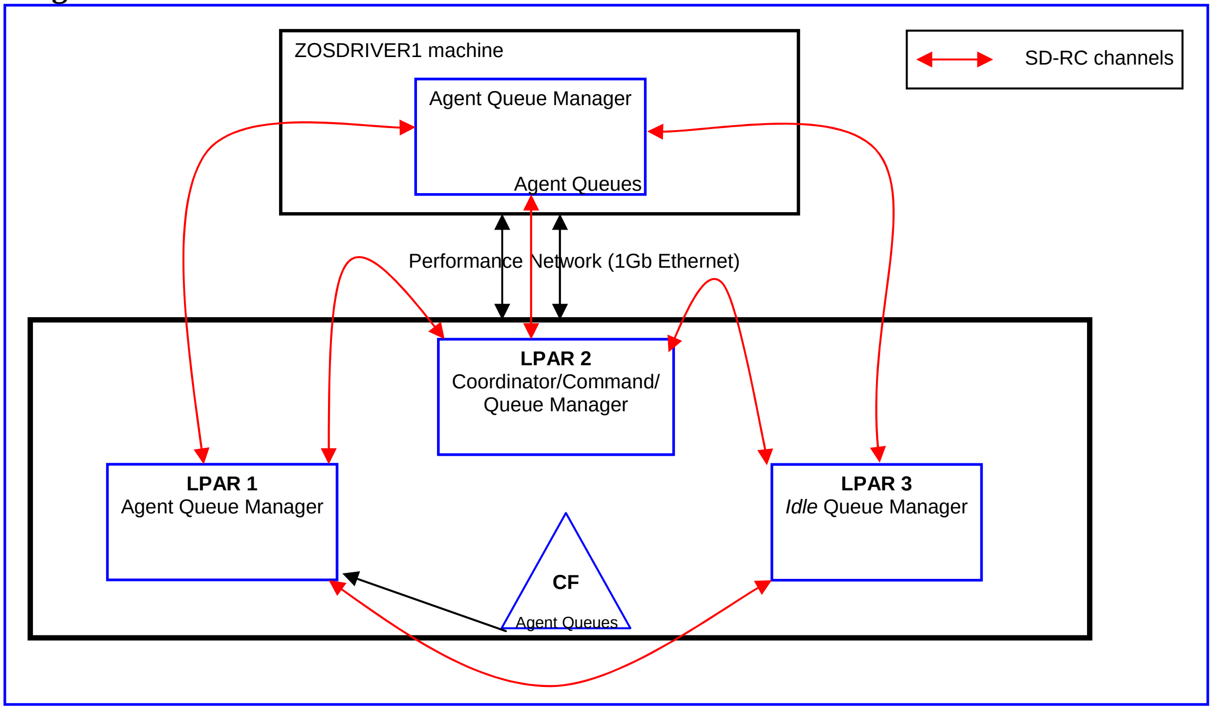
However, if the cluster is busy processing non-MQFTE work, then the FTE messages will have to wait their turn. This will be visible as an increased CURDEPTH on queue “SYSTEM.CLUSTER.TRANSMIT.QUEUE”.

By using separate Sender-Receiver channels between the MQFTE agents, there is a dedicated route between agents.

Shared Queues using Sender/Receiver Channels

The following diagram shows the configuration when the agent queues on MVS have been defined in the coupling facility. The channels defined are all sender-receiver as in scenario 1.

Diagram 3



The default configuration for MQFTE is to use 256KB messages.

In a shared queue environment, the messages would be stored in DB2 tables. This has the potential for significant performance degradation. However, as the channels are defined with NPMSPEED of FAST and the messages are processed out of syncpoint, they are eligible for “put to waiting getter”, meaning they are able to bypass the queues and therefore DB2 also.

This WebSphere MQ for z/OS optimisation is discussed later.

When defining an Agent to a z/OS queue manager, the “fteCreateAgent” script provides MQSC to define 5 queues.

These 5 queues are prefixed “SYSTEM.FTE”, followed by the role of the queue and then suffixed with the Agent name e.g. SYSTEM.FTE.COMMAND.QM01

When defining these queues, the MAXMSGL is 4MB, so the queues need to be defined to structures of CF level 4 or greater.

Whilst MQFTE sends non-persistent messages to the “DATA” queue, the other queues may use persistent messages, so they need to be in a structure defined with RECOVER(YES).

Chart 12

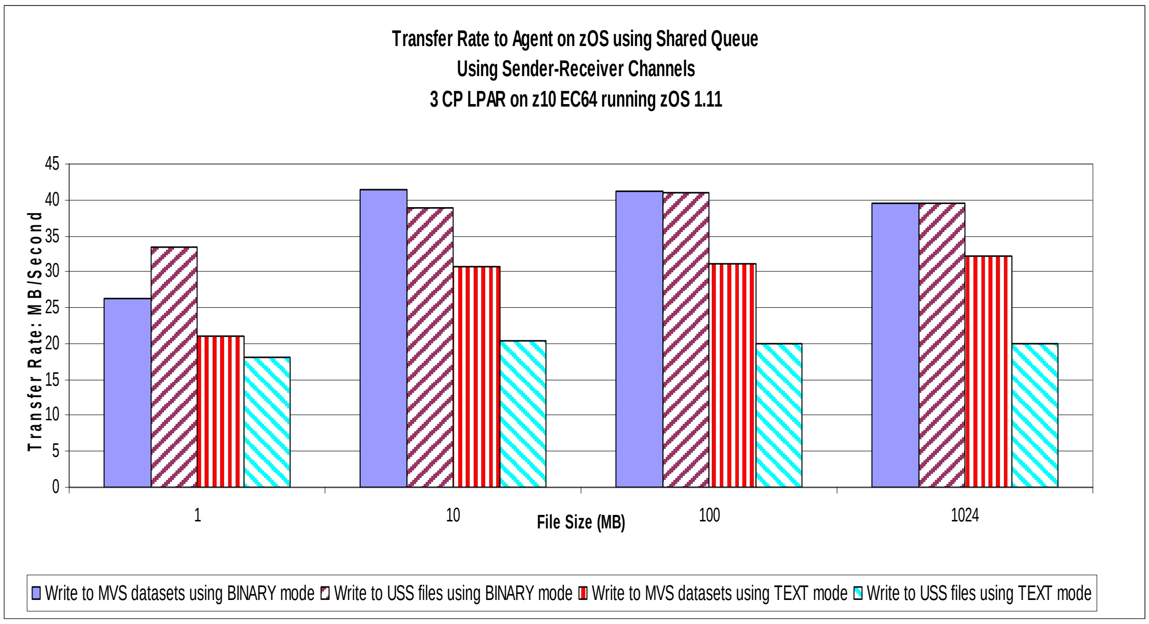


Chart 12 shows the achieved transfer rate in MB per second when sending files from Linux to MVS using Shared Queues.

Chart 13

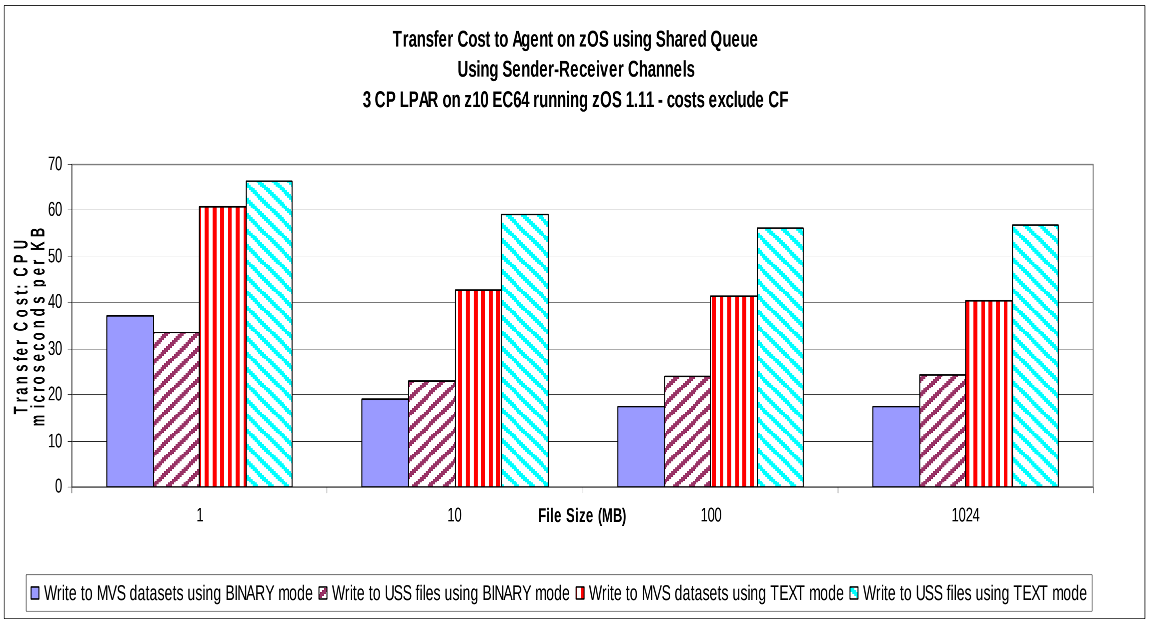


Chart 13 shows the actual transfer cost to the Sysplex in microseconds per KB when sending files from Linux to MVS where the MVS agents' queues are defined to the Coupling Facility.

Tuning your system for MQFTE

When introducing MQFTE to your System Z, there are a number of topics that should be reviewed. These include:

1. Network
   a. Tuning your network (stack)
   b. What to look out of in your network
2. Processors
   a. Does the number of processors affect MQFTE
   b. Will a zAAP processor save processing costs?
3. DASD
4. Unix System Services environment
5. WebSphere MQ Queue Manager configuration

The advice given in the sections below should be reviewed with your system experts as they may not be appropriate on your system.

Network

Tuning your Network (Stack)

The TCP/IP and z/OS UNIX System Service Performance Tuning advice found at http://publib.boulder.ibm.com/infocenter/tsminfo/v6/index.jsp?topic=/com.ibm.itsm.perf.doc/c_network_tcpip_zos_unix_sys_svcs.html offers tuning advice including:

- Set the client/server TCP window size to the allowed maximum. Set the TCP window size on z/OS to the allowed maximum by setting TCPRCVBUFRSIZE to 64K or larger.
- Spread z/OS UNIX user HFS datasets over more DASD volumes for optimal performance.
- Ensure TCP/IP and all other traces are turned off for optimal performance. Trace activity does create an extra processing overhead.

High latency networks are more efficient with a larger batch window. The concept is that the sender has not yet sent the last bit of the window size before it receives an ACK for the first bit of the current window.

In some networks, the maximum window size of 512K may not be high enough. z/OS v1r11 allows a maximum window size of 2MB.

What to look out for in your network

The command "/D TCPIP,,N,CONFIG" can be used to see what settings are implemented on your system, including the following:

- DEFAULTRCVBUFSIZE: 00065536 DEFAULTSNDBUFSIZE: 00065536

When running a transfer, issuing "TSO NETSTAT ALL (CLI *qmgr**" will report the achieved Send/Receive buffer size e.g.

- ReceiveBufferSize: 0001789828   SendBufferSize: 0000065536

z/OS v1r11 introduced a "dynamic right sizing" (DRS) algorithm into Communications Server with the aim of keeping the pipe full and preventing the sender from being constrained by the advertised buffer size. This allows the window to grow up to 2MB. The TCP/IP stack will disable the function if the application doesn't keep up.

The "NETSTAT ALL" report shows the DRS-adjusted receive buffer size. The current window size is shown in "RcvWnd" – e.g. 2,097,151 bytes.

The "TcpPrf" byte will be set to x'40' if DRS is enabled.

DRS is not available if the Send/Receive buffer size is less than 64KB.

Processors

In this section we investigate whether the number of processors can improve the transfer rate and whether zAAP processors can reduce the running costs of MQFTE.

Does the number of processors affect MQFTE?

If you are not CPU constrained, then making additional CPU available to your file transfer workload does not make significant improvements to the transfer rate and indeed the cost of processing the data is slightly higher as more processors are made available.

The following charts shows the achieved transfer rate and cost for binary type transfers whilst increasing the number of CP-type processors from 1 to 5.

Chart 14

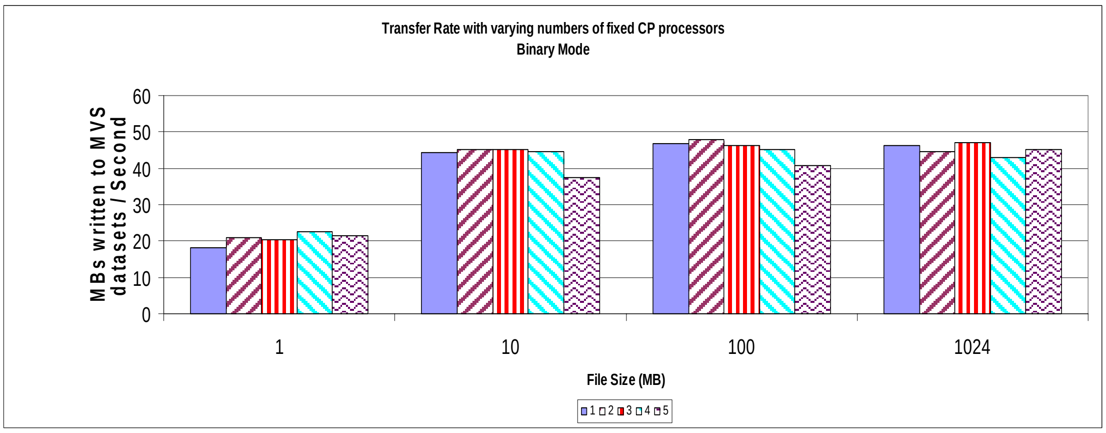
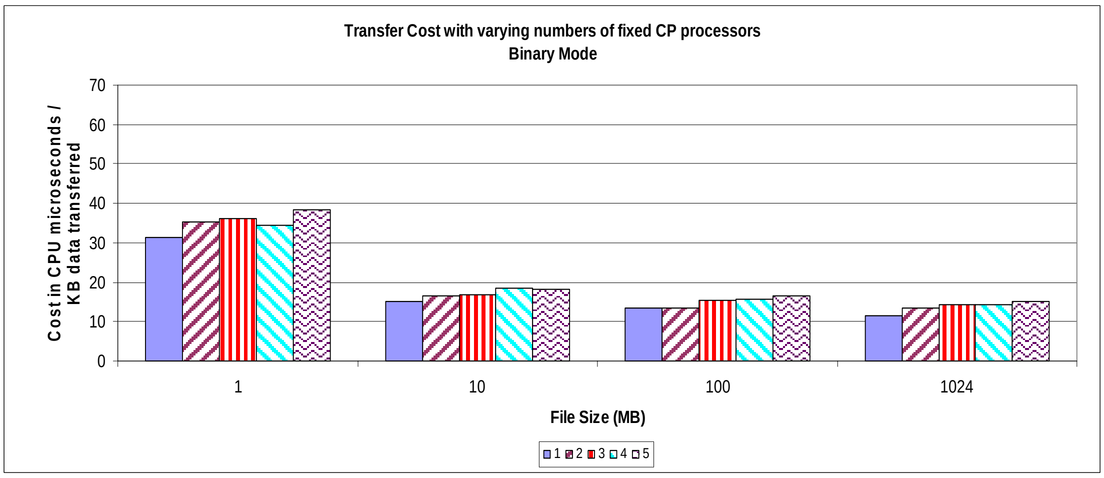


Chart 15



Despite the cost of the text transfer being significantly more than a binary transfer, provided there is sufficient CPU capacity available to process the file transfer, making additional processors available typically does not make a significant difference.

However, there is a noticeable difference when moving from 1 processor to 2 processors.

Chart 16

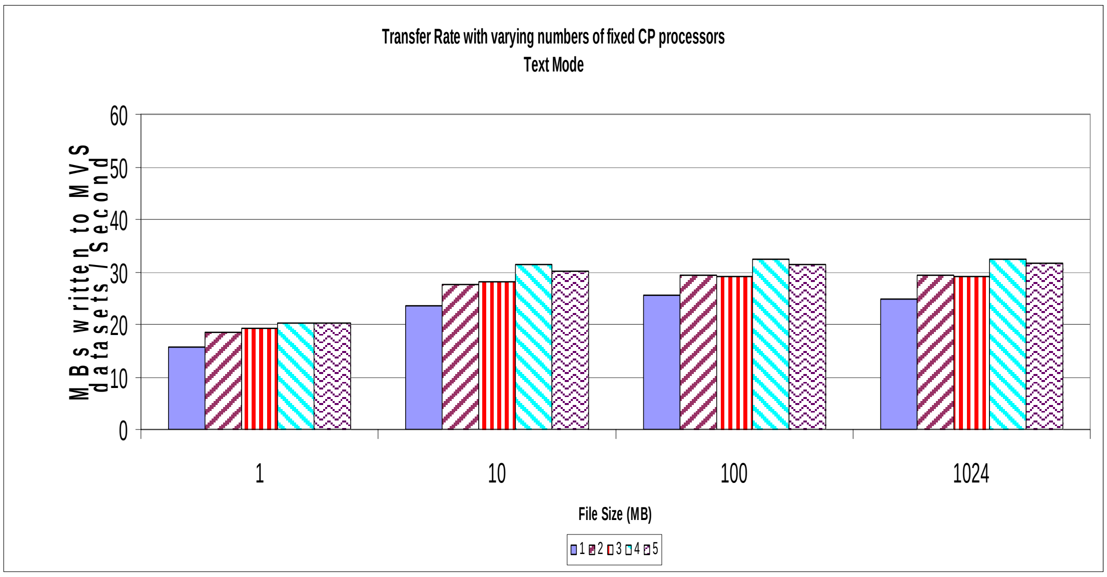
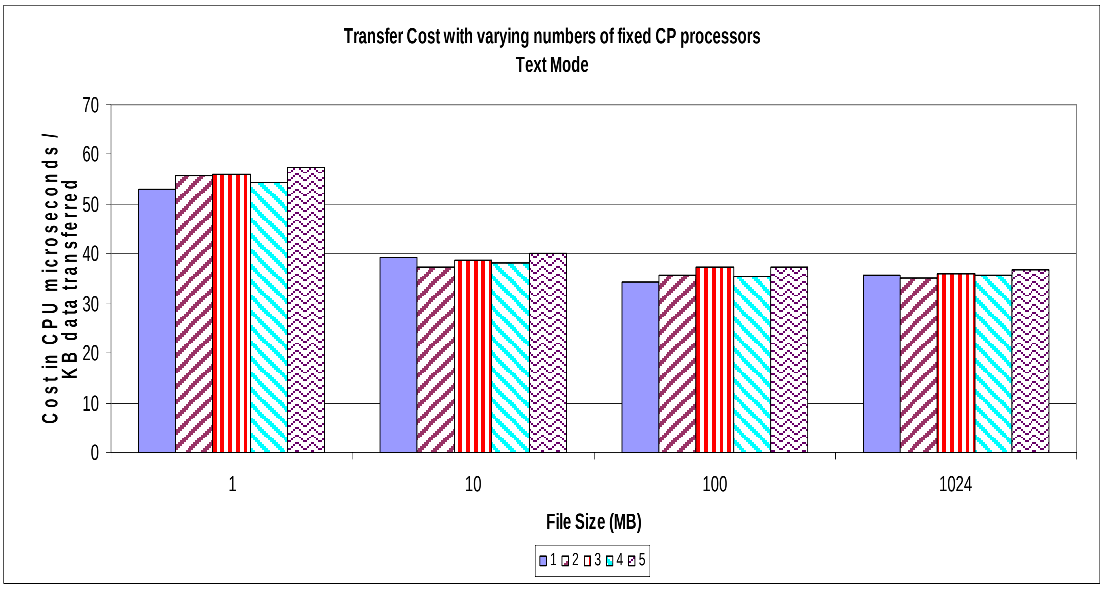


Chart 17



Will a zAAP processor save processing costs?

The IBM System z Application Assist Processors (zAAP) are available on all IBM System z10, IBM System z9, IBM eServer zSeries 990 (z990) and IBM eServer zSeries 890 (z890) systems.

For Java workloads, zAAP may enable customers to lower the overall cost of computing for Java technology-based applications, through hardware, software and maintenance savings. For further information on zAAP processors see: http://www-03.ibm.com/systems/z/advantages/zaap/index.html

MQFTE is written in Java and as such can benefit from the availability of zAAP processors.

If the zAAP processor is already in use, the MQFTE workload may run on a regular CP-type processor, unless the IFAHONORPRIORITY=NO.

WebSphere MQ is not written in Java and does not benefit directly from the use of zAAP processors, however the offloading of MQFTE work to zAAP processors relieves some of the workload on the CP-type processor for other work.

Chart 18

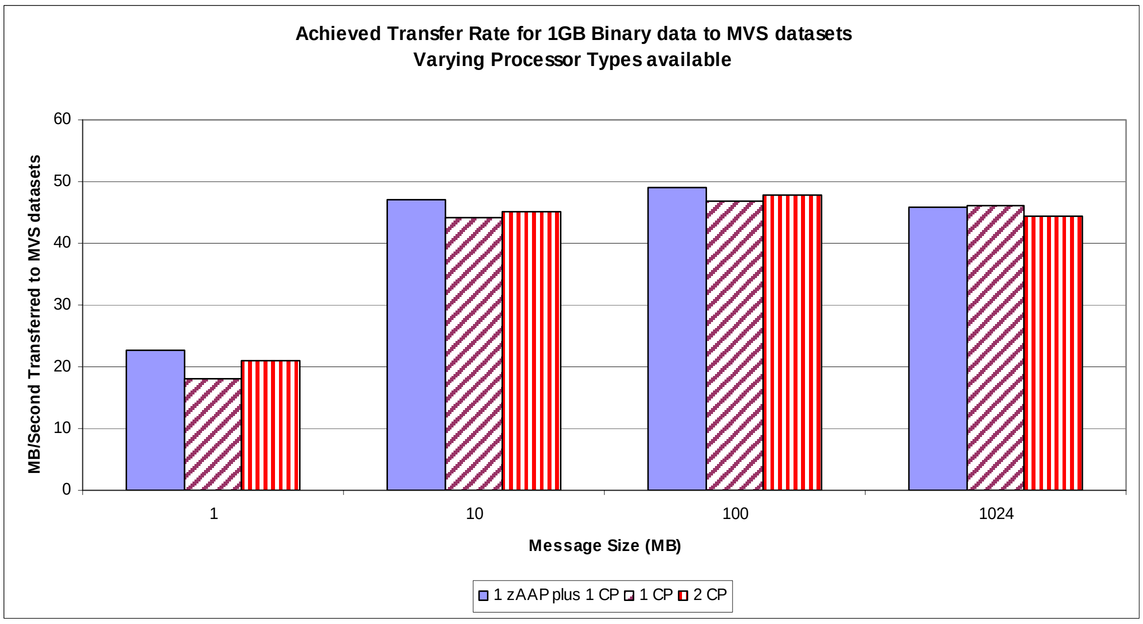


Chart 18 compares the achieved transfer rate when running a binary transfer to MVS datasets. There are 3 different environments – running with 1 zAAP and 1 CP-type processor compared to running with 1 or 2 CP type processors.

As can be seen in the above chart, there is no significant difference in the achieved transfer rate.

Chart 19

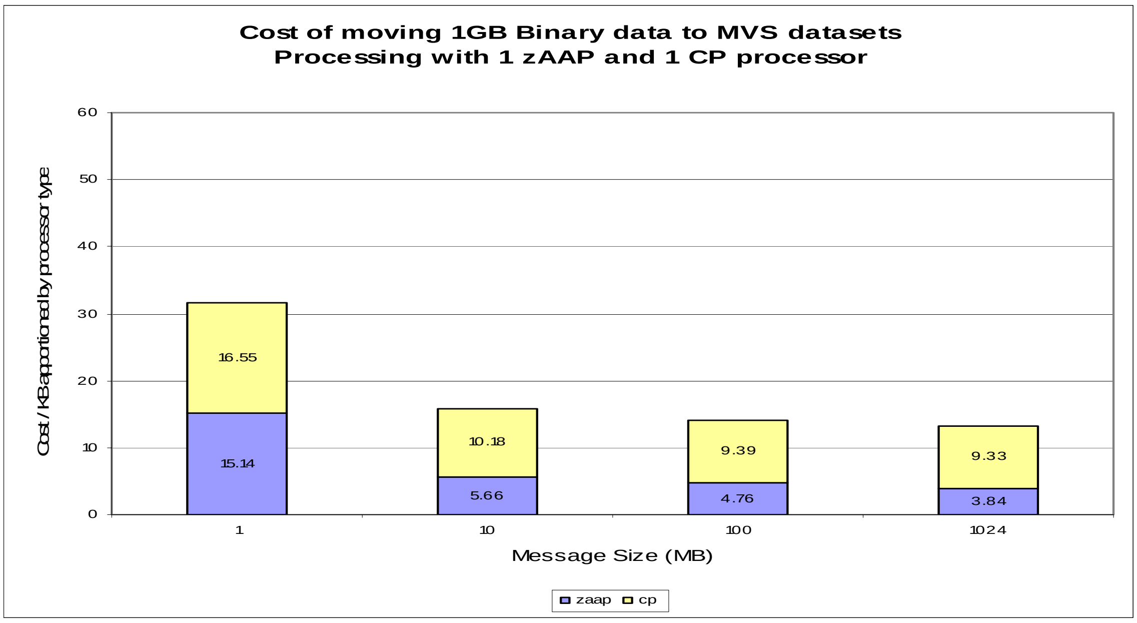


Chart 19 shows the actual cost of moving 1GB's worth of binary data onto MVS datasets (in microseconds per KB) when the system is configured with 1 zAAP and 1 CP processor.

- For small files, approximately 50% of the cost of the transfer can be offloaded onto zAAP processors.
- For larger files, approximately one-thirds of the cost of the transfer can be offloaded onto zAAP processors.

Chart 20

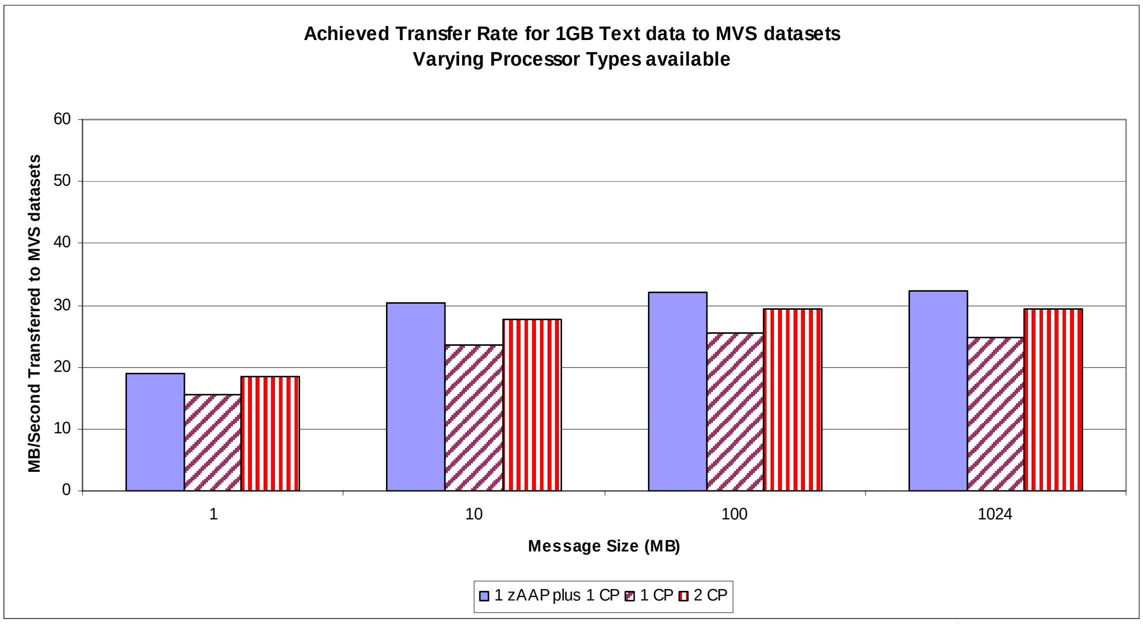


Chart 20 shows that using 1 zAAP plus 1 CP provides the equivalent transfer rate as 2 CP type processors when moving text data from Linux to MVS datasets. The act of transferring text data uses more CPU as the data needs to be converted using Java methods.

Chart 21

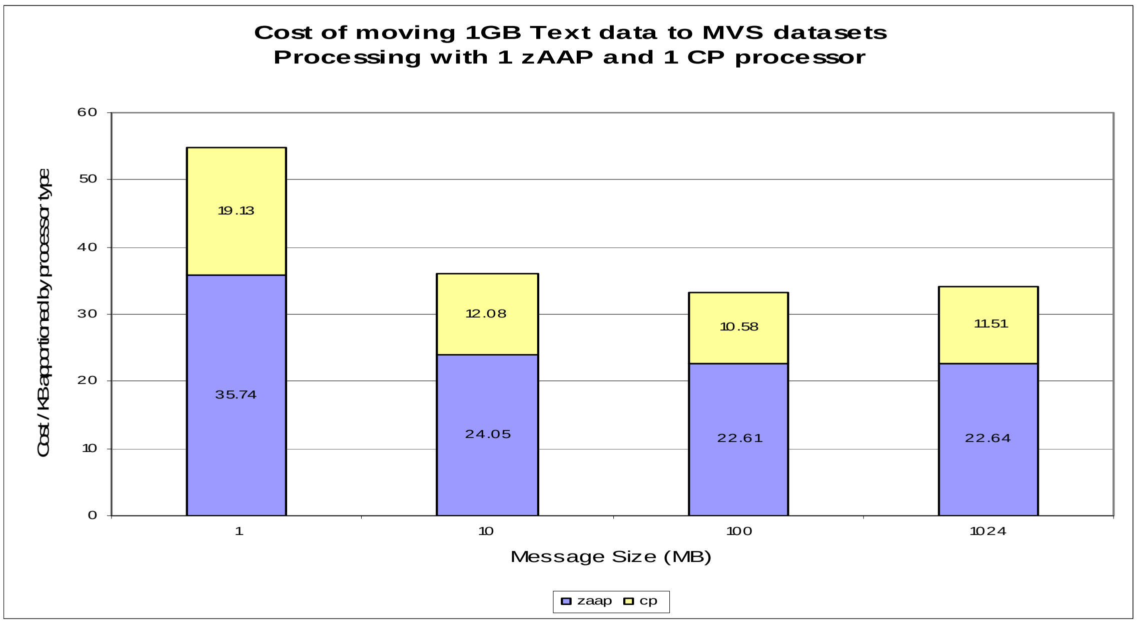


Chart 21 shows the actual cost of moving 1GB's worth of text data onto MVS datasets (in microseconds per KB) when the system is configured with 1 zAAP and 1 CP processor.

As the data conversion takes place with MQFTE, the cost can be offloaded onto the zAAP processor.

- Approximately two-thirds of the cost of the transfer can be offloaded onto zAAP processors.

Chart 22

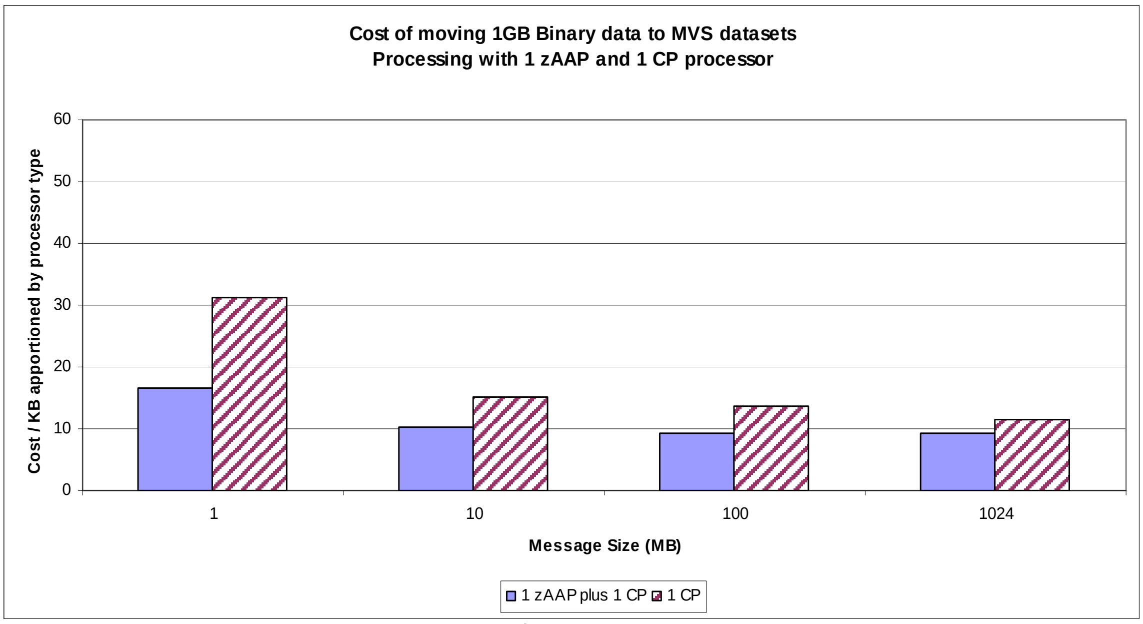


Chart 22 compares the cost per KB transferred in binary mode when a zAAP processor is available against when a zAAP processor is not available.

The cost shown for "1 zAAP plus 1 CP" is the actual cost measured on the CP-type processor. The costs attributed to the zAAP processor are not included as IBM does not impose software charges on zAAP capacity[2].

Chart 23

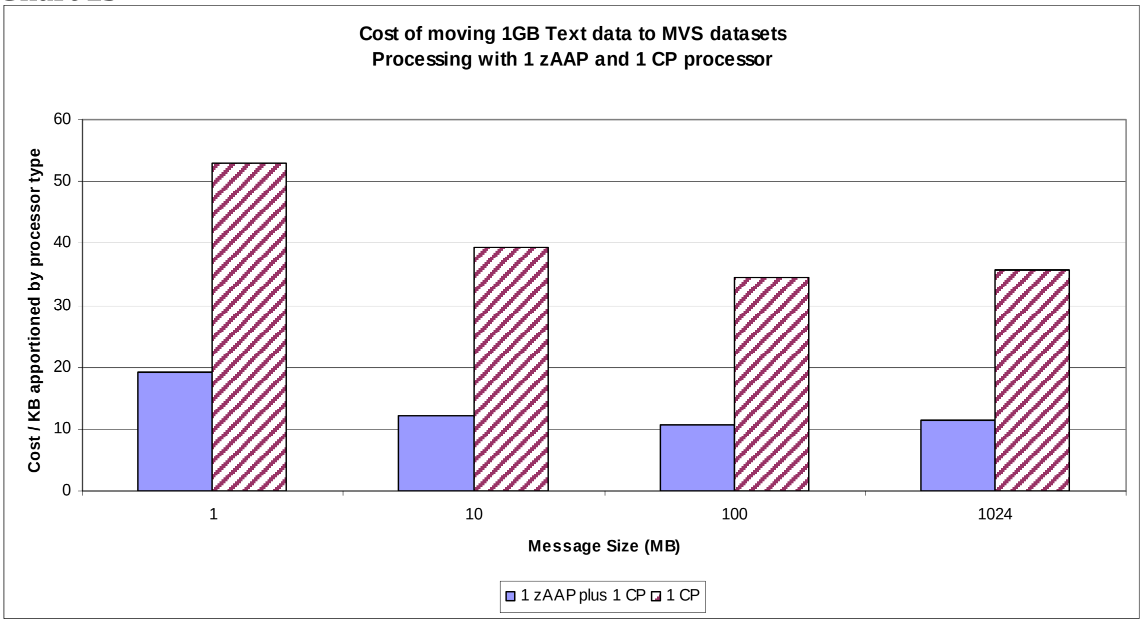


Chart 23 compares the cost per KB transferred in text mode when a zAAP processor is available against when a zAAP processor is not available.

The cost shown for "1 zAAP plus 1 CP" is the actual cost measured on the CP-type processor.

[2] As per statement in http://www-03.ibm.com/systems/z/advantages/zaap/index.html

DASD

Optimising BLKSIZE

When transferring data into MVS datasets, MQFTE allows you to specify the LRECL and BLKSIZE attributes for the target dataset.

It is recommended that an optimum BLKSIZE is used for the data being transferred.

For example, consider a 1MB file that is written to a file with a record length of 80 bytes but varying the BLKSIZE:

o BLKSIZE(27920)	uses 18.9 tracks	(1.02MB)
o BLKSIZE(6400)	uses 21.0 tracks	(1.135MB)

Using a BLKSIZE that is less than optimum size has resulted in using 11% more disk space.

For best performance, aim for 2 blocks per track.

High Performance FICON for System Z

zHPF is a performance and reliability, availability, serviceability enhancement of the z/Architecture and the FICON channel architecture implemented in System z10.

Exploitation of zHPF by the FICON channel, the z/OS operating system and the control unit is designed to help reduce the FICON channel overhead.

A "High Performance FICON for System z Technical Summary for Customer Planning" document can be found at ftp://ftp.software.ibm.com/common/ssi/sa/wh/n/zsw03058usen/ZSW03058USEN.PDF that provides a high level overview of the significance of the enhancements for zHPF.

In the single threaded file transfer measurements detailed earlier, we saw no additional benefit when enabling zHPF.

In an environment the system is running other workload where disk I/O is frequent and the file transfer data is not segregated onto its own volumes and specific channel paths, some benefit may be seen by enabling zHPF.

Unix System Services Environment

The MQFTE Agent runs as a Java application within our Unix System Services (USS) environment.

Our performance Sysplex has been configured such that we have a shared zFS file system that is owned by one LPAR – in our case LPAR2.

When transferring files into USS on LPAR 1, there is additional cost incurred by LPAR 2.

Additionally there is cost incurred by XCF to send the requests from LPAR 1 to LPAR 2.

The following chart shows an extract of the XCF Activity Report which is available from the RMF III Monitor which was run when transferring 1MB files from Linux to USS files.

Chart 24

```
                                                                 X C F   A C T I V I T Y
z/OS V1R11                SYSTEM ID MV2B              DATE 11/19/2009            INTERVAL 02.33.000
                                      RPT VERSION V1R11 RMF       TIME 12.44.30                CYCLE 0.200
    SECONDS

                                                        XCF USAGE BY SYSTEM
-----------------------------------------------------------------------------------------------------------
REMOTE SYSTEMS                                                                        LOCAL
-----------------------------------------------------------------------------------------------------------
OUTBOUND FROM MV2B                                      INBOUND TO MV2B                   MV2B
-----------------------------------------------------------------------------------------   ---------------------------
                                        ----- BUFFER -----       ALL
TO         TRANSPORT BUFFER           REQ    %    %    %    %    PATHS      REQ    FROM              REQ      REQ
SYSTEM     CLASS     LENGTH           OUT  SML  FIT  BIG  OVR  UNAVAIL   REJECT    SYSTEM             IN   REJECT
MV2A       DEFAULT      956         5,021    0   99    1  100        0        0    MV2A            4,989        0
MV25       DEFAULT      956       132,588    0   68   32  100        0          0   MV25          132,556
    0
                               ----------                                                        ----------
TOTAL                             137,609                                          TOTAL       137,545
```

Explanation of fields in XCF Activity Report – chart 24

- Transport Class
  Name of transfer class used by XCF for message transfer, e.g. DEFAULT
- Buffer Length
  Internally defined size of message buffer that XCF uses for the named transport class. Is the maximum size of the message that can be contained in the buffers currently being used for this transaction class.
- REQ OUT
  Total number of messages that XCF accepted for delivery to system in indicated transaction class.
- %FIT
  Percentage of messages sent that fit into the defined buffer length
- %BIG
  Percentage of messages needing a larger than defined buffer length.
- %OVR
  Percentage of BIG messages sent that suffered performance degradation.

In the above report, 32% of the 132,588 messages accepted for delivery were regarded as BIG messages, i.e. larger than 956 bytes.

WebSphere MQ Queue Manager Configuration

The "WebSphere MQ for z/OS Capacity Planning & Tuning Guide" SupportPac, available at http://www-1.ibm.com/support/docview.wss?uid=swg24007421&rs=171, offers advice on the general configuration of an MQ queue manager. This supportpac is also known as MP16.

The following advice is relevant to queue managers used for MQFTE:

1. Data transmitted between MQFTE agents is non-persistent and are sent over channels with NPMSPEED(FAST) enabled.

The impact of using NPMSPEED(NORMAL) channels is discussed later in this section.

2. Typically the messages sent are 256KB and the transfer only takes place when both ends are available, so messages should not build up on the destination queue.

Even on a CPU constrained destination system where the messages are remaining on the MQ queues for a period of time, the depth of the SYSTEM.FTE.DATA.<agent> queue should not exceed 50 – this is defined by the MQFTE tuning attributes agentWindowSize * agentFrameSize.

Consider a case where the receiving agent is stopped mid-transfer. The sending agent will continue to send messages until it reaches its unacknowledged limit, which is dependent upon the agentFramesize and if using the default values will be between 1 and 50 messages – depending on how far through the agentWindowSize * agentFrameSize batch the transfer had gotten at the point of failure. In this case, the worst case is that 50 messages will reside in the queue managers' buffers until the agent is able to get the messages.

This means that the queue manager would use:

256KB * 50 = 12800 KB (or 3200 pages of buffers)

To avoid the messages being written to page set, it is advisable to ensure the target buffer pool has sufficient capacity to cope with this situation.

Queue Manager – Put to Waiting Getter

Since WebSphere MQ for z/OS version 6.0, out-of-syncpoint non-persistent messages can be MQPUT directly to a waiting out-of-syncpoint MQGETter, rather than placed into the queue and then read from the queue.

Receiving channels which have been defined with NPMSPEED(FAST) use out-of-syncpoint MQPUTs for non-persistent message; applications which issue out-of-syncpoint MQGETs against these messages are therefore eligible to benefit from this improvement.

The larger the message, the greater the CPU reduction – for 4MB messages there is a CPU reduction of approximately 40% and in an MVS to MVS environment, there is a throughput improvement of approximately 40%. See SupportPac MP1E for further details.

MQFTE's Queue Usage

When using the fteCreateAgent script to define an Agent, it is necessary to define a number of MQ queues. These are prefixed "SYSTEM.FTE" and suffixed with the Agent name:-

1. **COMMAND**
   This queue is used when the transfer is being initiated / terminated
2. **DATA**
   This queue is used when the transfer is in progress to hold the files being transferred. By default these will hold 256KB non-persistent messages.
3. **REPLY**
   This queue is used during the transfer process to allow agents to exchange acknowledgment of progress.
4. **STATE**
   The STATE queue is maintained to allow MQFTE to track the progress of the file transfer. The size of the message put to the queue depends on:
   - The number of files being transferred
   - The path length to the file(s) being transferred.

   When transferring a large number of files or transferring files with long path names, the size of the message put to the STATE queue will increase.

   Putting a large message to the queue can cost significantly more than putting a small message, for example a put of a 16KB persistent message costs 138 microseconds on our system (z10-EC64) compared to 2561 microseconds for a 1MB persistent message.

   If the STATE queue has been defined using the default MQFTE definitions, the maximum message length will be 4MB. Transferring a large number of files with long path names can result in a STATE message being larger than 4MB – it may be necessary to increase the value of MAXMSGL for the STATE queue.

5. **EVENT**
   This queue is used when monitors are triggered and for other non-regular transfer functions.

Channel Attribute NPMSPEED

The attribute NPMSPEED defines the class of service for non-persistent messages on this channel.

FAST
Fast delivery for non-persistent messages, messages might be lost if the channel is lost. Messages are retrieved using MQGMO_SYNCPOINT_IF_PERSISTENT and so are not included in the batch unit of work. This is the default
NORMAL
Normal delivery for non-persistent messages.

Chart 25

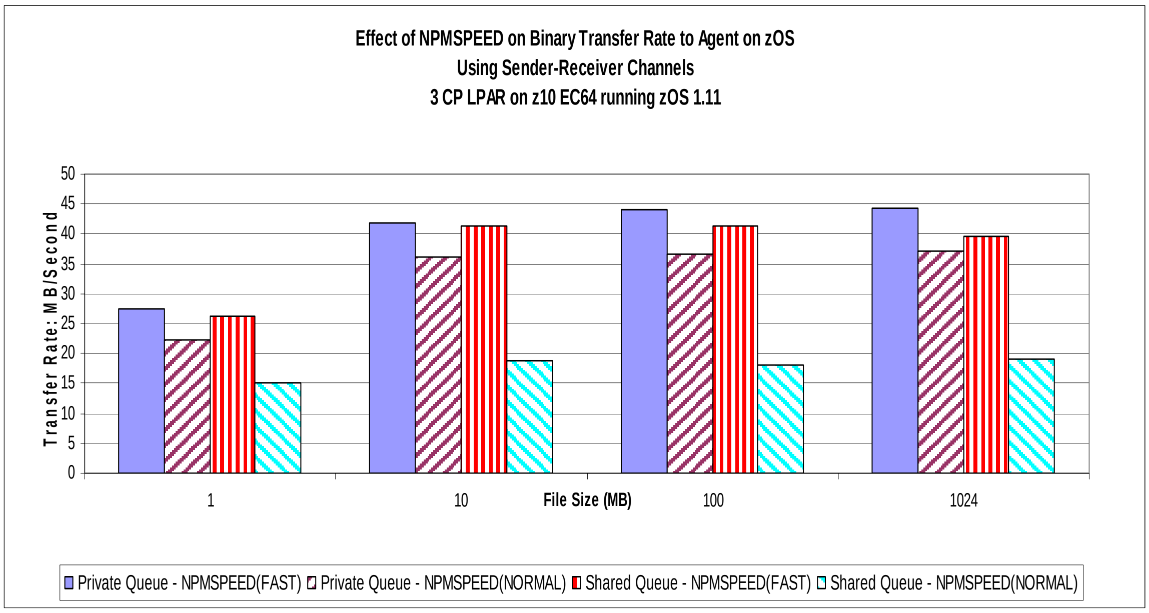


Chart 25 compares the achieved transfer rate for binary data from Linux to MVS datasets.

Chart 26

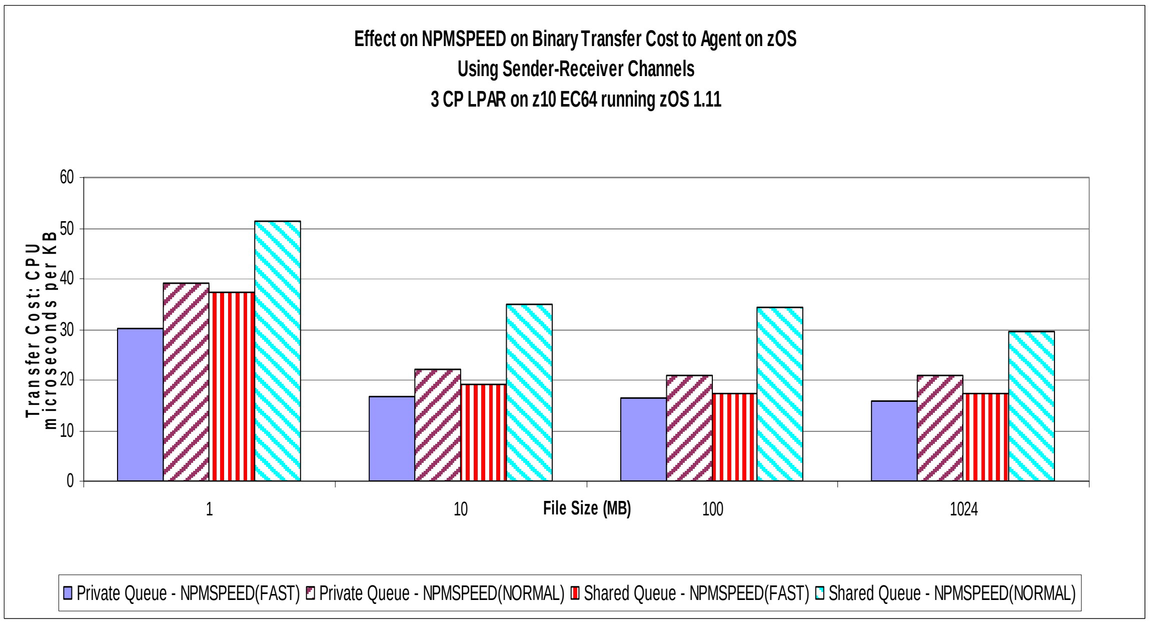


Chart 26 compares the actual transfer cost for the binary data from Linux to MVS datasets.

Both charts 25 and 26 compare NPMSPEED(FAST) and NPMSPEED(NORMAL) for both private and shared queues.

In the private queue measurements we can see that NPMSPEED(NORMAL) does degrade throughput by approximately 20%.

In the shared queue measurements, the NPMSPEED(NORMAL) channel degrades the achieved transfer rate by 50%.

The increased cost of NPMSPEED(NORMAL) on shared queue is due to the message being written to DB2 as they are larger than 63KB in size.

There is no benefit in using channels defined with NPMSPEED(NORMAL) for MQFTE.

Channel Attributes COMPHDR and COMPMSG

The WebSphere MQ channel attributes COMPHDR and COMPMSG were introduced in MQ version 6.0 and offer the opportunity to compress the header and message data in a network constrained environment.

COMPHDR(SYSTEM) will help reduce the size of the MQ headers being transmitted.

- Typically an MQ message header is 476 bytes
- Using COMPHDR(SYSTEM) reduces this to approximately 65 bytes.

Considering the size of the default message used by MQFTE is 256KB, saving 411 bytes on MQ headers is unlikely to make a significant difference.

COMPMSG(RLE) may help if your data contains repeating characters, such as a printed report with long strings of blanks.

COMPMSG(ZLIB[HIGH|FAST]) uses ZLIB encoding to compress the message payload, either prioritised for compression or speed. Whether this is of benefit is dependent upon the actual message payload.

Compressing the message payload may reduce the amount of data being flowed over the network but remember that there is CPU cost involved in compressing and decompressing the data at either end of the channel.

Performance SupportPac MP1E discusses these channel attributes in more detail and is available at: http://www-01.ibm.com/support/docview.wss?rs=171&uid=swg24009932&loc=en_US&cs=utf-8&lang=en

MQFTE – Advanced Tuning Options

Each agent has its own properties files that contain the information that an agent uses to connect to its queue manager. This "agent.properties" files can be altered to use advanced agent properties. Some of these agent properties are shown below with their default value.

o agentChunkSize	262144 bytes (256KB)
o agentWindowSize	10
o agentFrameSize	5
o agentCheckpointInterval	1

For each agentWindowSize messages received on the DATA queue, a reply message is sent to the sending agent.

If the sending agent sends agentWindowSize * agentFrameSize messages and does not receive acknowledgement for the first agentWindowSize batch of messages, the sending agent will stop sending any more messages until acknowledgement is received.

For every agentWindowSize * agentFrameSize * agentCheckpointInterval messages received on the DATA queue, an update is made to the STATE queue, which involves an MQGET and MQPUT of a persistent message.

For the default agentCheckpointInterval the total amount of data moved is:

agentChunkSize	* agentWindowSize	* agentFrameSize		
256KB	* 10	* 5	* 1	= 12,800 KB data

This means that 12.5MB of data will be transferred per checkpoint when the agentCheckpointInterval is 1.

When altering the advanced tuning options, consider the relationship between attributes. For example consider the following table:

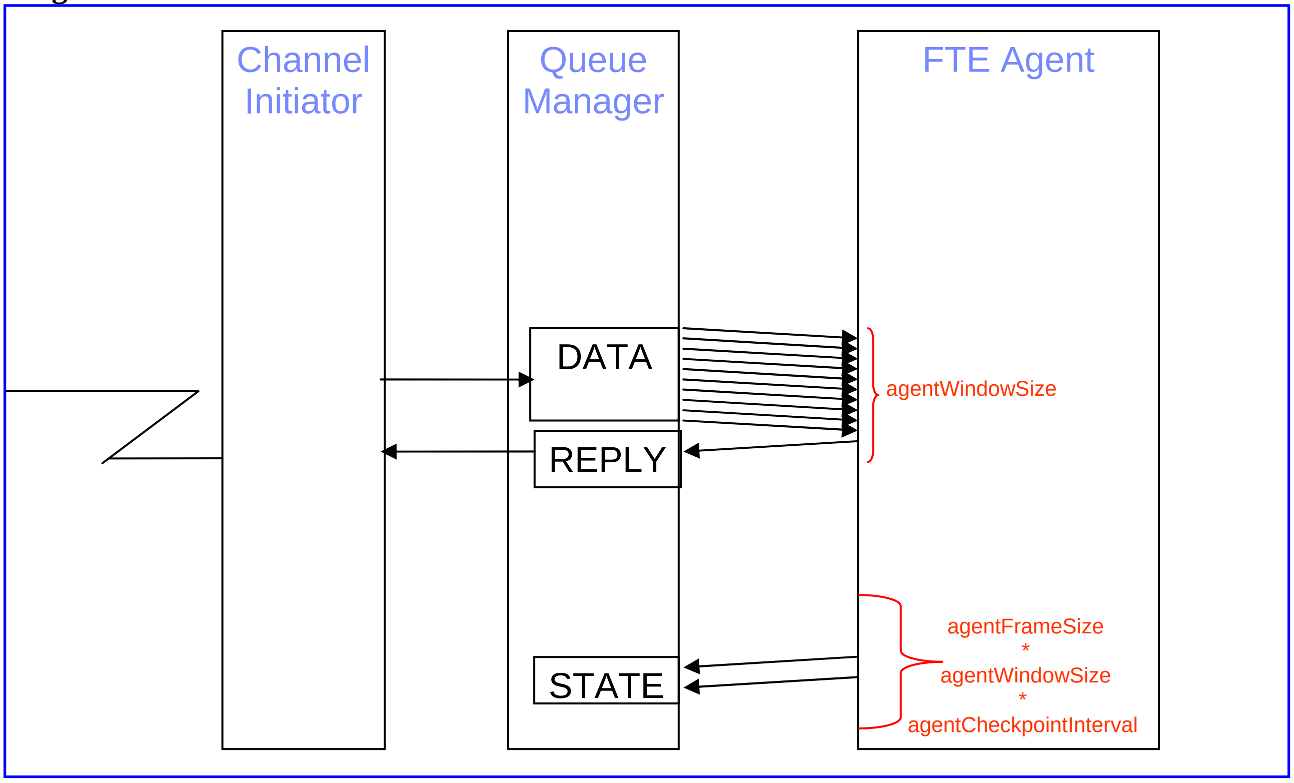
Configuration	agentWindowSize	agentFrameSize	agentCheckpoint Interval
1	10	5	2
2	10	10	1

In the above table, for each configuration the STATE queue is updated every 100 messages, i.e.: agentWindowSize * agentFrameSize * agentCheckpointInterval

In configuration 2, the increased agentFrameSize means that the sending agent will send twice as many batches of data before pausing in the event of the acknowledgement message not being received from the first batch.

Diagram 4 attempts to show the flow of messages from a remote agent to a z/OS queue manager.

Diagram 4



The messages flow in from the network to the channel initiator and for every 10 (agentWindowSize) messages put to the SYSTEM.FTE.DATA.<receivingAgent> queue, a single acknowledgement message is sent to the SYSTEM.FTE.REPLY.<sendingAgent> queue.

For every 5 (agentFrameSize * agentCheckpointInterval) acknowledgement messages sent, an MQGET (destructive) followed by an MQPUT is made to the SYSTEM.FTE.STATE.<receivingAgent> queue

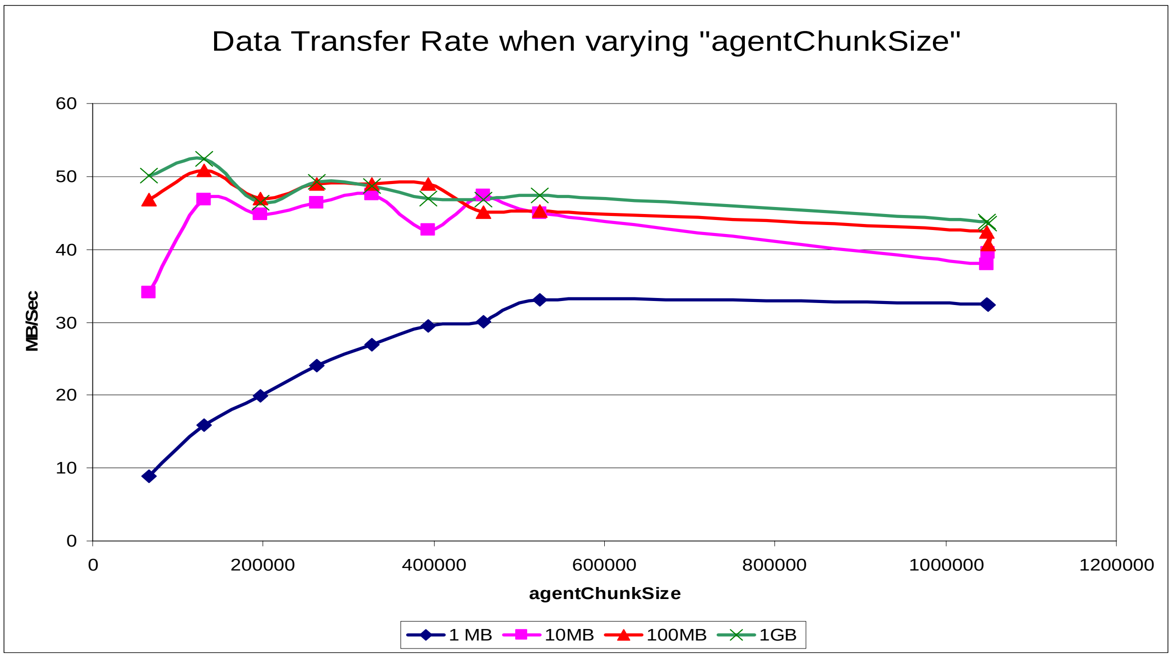
Depending on your network, you may find it beneficial to increase or decrease the agent properties mentioned previously in this section.

agentChunkSize

Varying the size of the message transmitted over your network may offer some benefit to the rate at which the data is transferred between agents.

The following chart shows the data transfer rate for a binary-type transfer between Linux files and MVS sequential datasets, whilst varying the agentChunkSize agent property from 64KB to 512KB.

Chart 27



For our network configuration, Chart 27 suggests that the optimum agentChunkSize is different for small files to that of large files.

For small messages of 1MB, an agentChunkSize of 512KB allows a 20% increase in transfer rate over the default size.

For large messages, an agentChunkSize of 128KB appears to give the best data transfer rate on our systems.

agentWindowSize

The agentWindowSize property can be used to control the amount of syncpoints committed, as well as the number of acknowledgements sent between two agents when transferring files.

Each agentWindowSize batch requires an acknowledgement message but the agent will allow agentFrameSize windows to be sent before the first acknowledgement message must be received else the transfer will pause until the message is received. Once the first acknowledgment is received, the next batch can be sent, at which point the second acknowledgment must be received before the subsequent batch can be sent, and so on.

The agentWindowSize property has a default value of 10. This means that for every 10 chunks of data sent over WebSphere MQ, the sending agent will take an internal checkpoint.

Increasing this property increases the amount of data that could potentially need to be re-transmitted if a recovery is required and is not recommended for unreliable networks.

Chart 28

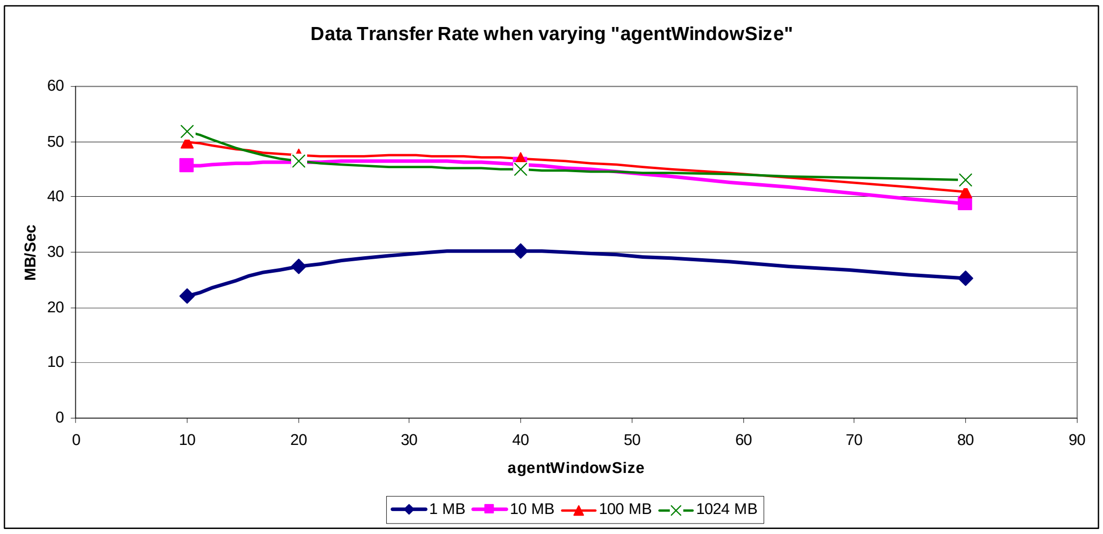


Chart 28 shows the effect of varying the agentWindowSize property on various sizes of files being transmitted in binary mode from Linux to MVS.

For larger files, the optimum size is the default of 10, but for smaller messages, the optimum size is 40, which allows a 25% improvement in throughput.

agentFrameSize

The agentFrameSize property can be used to control the number of windows for the transfer frame.

Chart 29

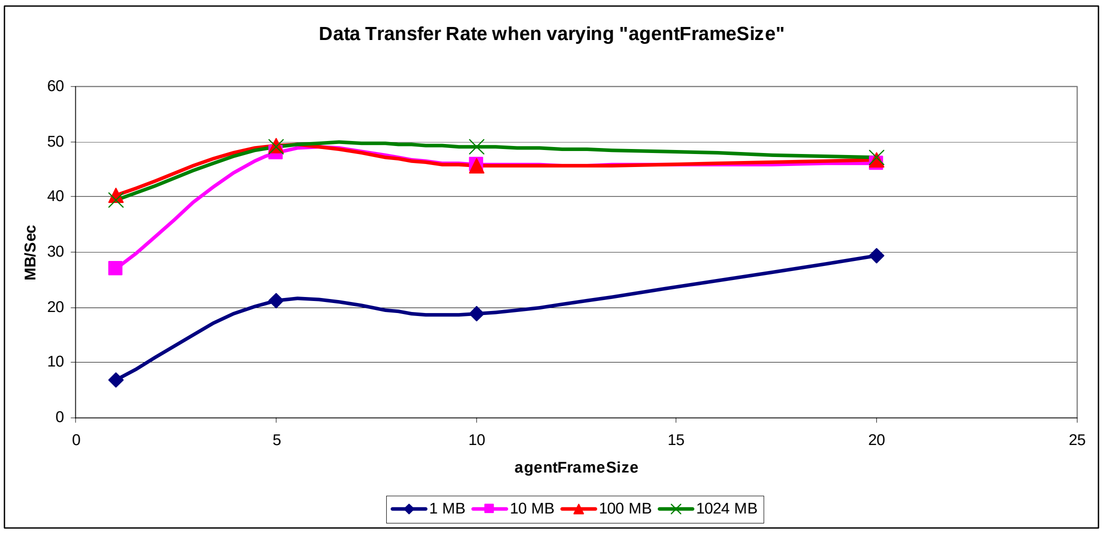


Chart 29 shows the effect of varying the agentFrameSize property on a range of file sizes that are transferred between Linux files and MVS datasets.

The default value of 5 for agentFrameSize is most appropriate in our configuration for large files, i.e. of 10MB and larger.

For smaller files, increasing the agentFrameSize may increase the data transfer rate.

Chart 30

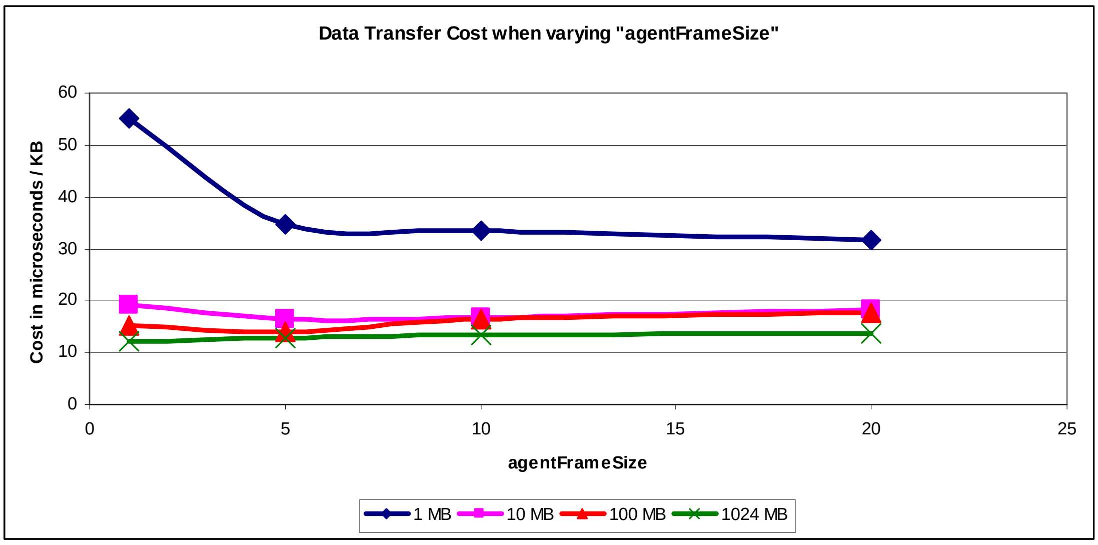


Chart 30 shows the actual transfer costs on MVS when transferring binary files from Linux files to MVS datasets whilst varying the agentFrameSize.

Typically the cost of moving the data is relatively constant for a file size regardless of the agentFrameSize, however for small messages and a low agentFrameSize (i.e. 1), the cost of the transfer per KB is significantly higher.

By setting the agentFrameSize to "1", and leaving the agentCheckpointInterval as 1, the Agent is processing 2.5MB of transferred data and then updating the "STATE" queue.

agentCheckpointInterval

Varying the size of the checkpoint interval affects the number of complete frames of data at which a checkpoint is taken for recovery purposes. If a transfer fails, the transfer can only recover at checkpoint boundaries. Hence the larger the value of agentCheckpointInterval, the longer the agent will take to recover failed transfers.

With a reliable network and system, there may be benefit in increasing the size of the agentCheckpointInterval.

When a frame is complete, the receiving agent will apply an update to its STATE queue.

Chart 31

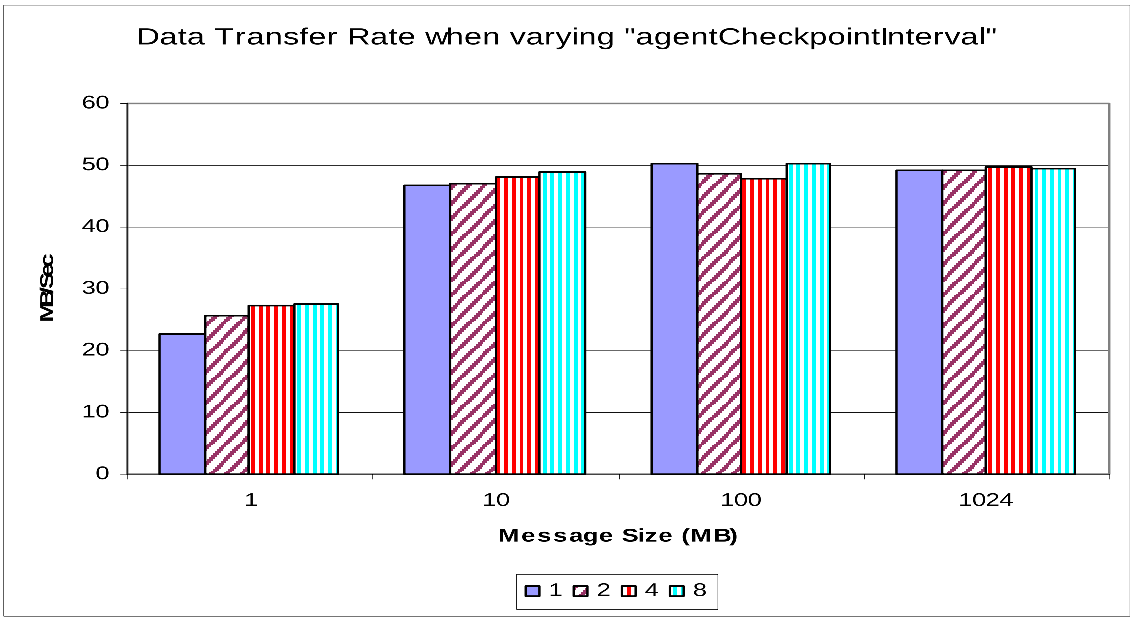


Chart 31 shows the achieved transfer rate when moving binary-type files from Linux to MVS datasets whilst varying the size of the agentCheckpointInterval from 1 to 8.

By using default agent properties for all attributes other than agentCheckpointInterval means that the check point will be between 12.5MB and 100MB.

In our network, varying the value of agentCheckpointInterval does not make a significant difference.

When the value of agentCheckpointInterval is increased to 8, a saving of 4 microseconds per KB of data received is seen for small files (of 1MB). This saving is not seen when transporting larger files.

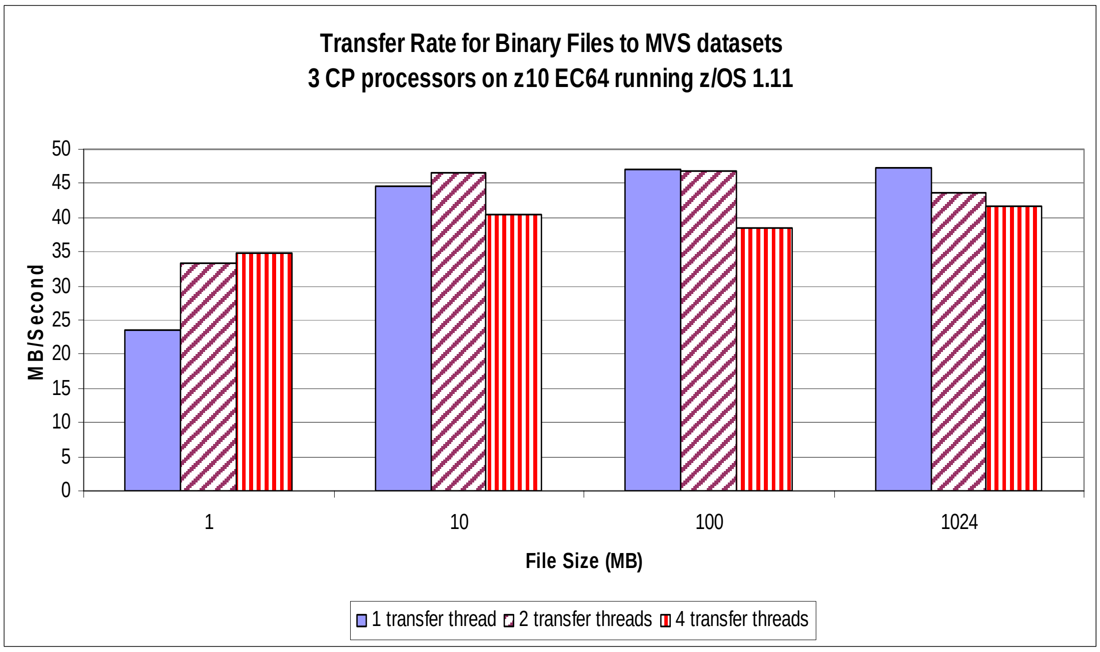
Multiple Concurrent File Transfers

For the single threaded file transfers we have seen a peak transfer rate of 50MB per second in our 1 Gb-rated network[3]

Whilst we do not expect to match the capacity of the FTP protocol as we effectively pause the sending of the files to wait for the acknowledgement of the batch, we can be sending a subsequent batch over the MQ channel whilst the first is waiting for the acknowledgement.

The following charts show the data transfer rate when sending performing multiple concurrent file transfers from Linux to MVS.

Chart 32



Charts 32 shows the achieved transfer rate when sending binary files from Linux files to MVS datasets whilst increasing the number of concurrent threads.

For smaller files, we are able to increase the transfer rate by 30% by running 2 concurrent transfers.

For larger files there is no significant benefit in using multiple transfers as the channel is unable to send the data any faster.

The actual cost of transferring the binary files from Linux to MVS datasets is relatively constant regardless of the number of concurrent transfers.

[3] Whilst the performance backbone is 10Gb, we only have 1Gb network cards.

Chart 33

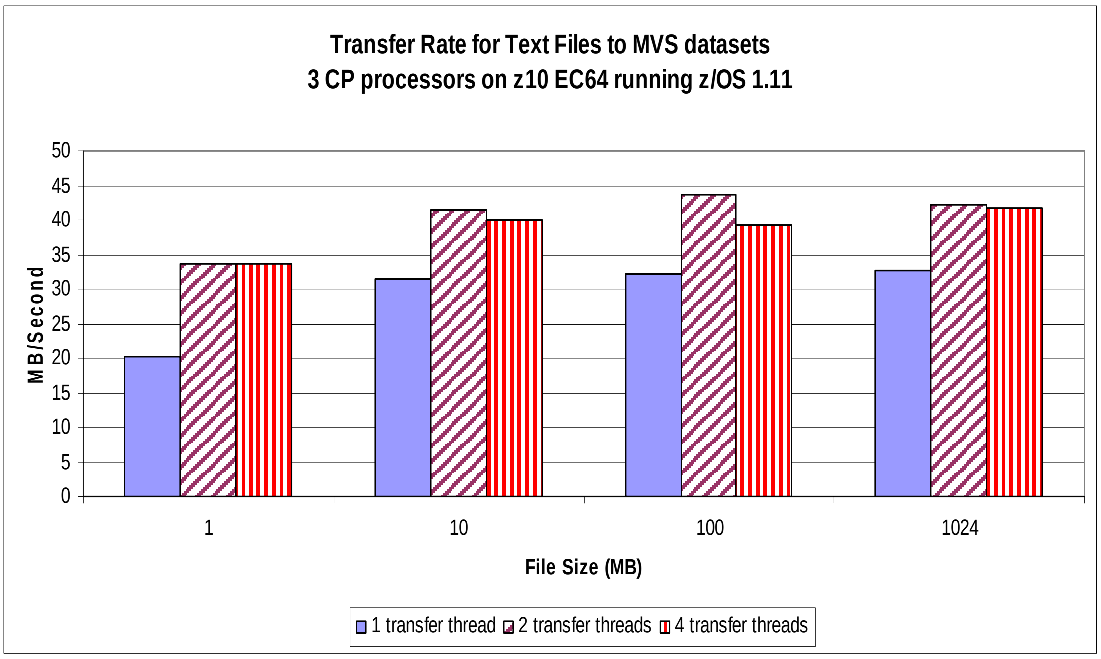


Chart 33 shows the achieved transfer rate for a text-type transfer between Linux files and MVS datasets, whilst increasing the number of concurrent transfer threads.

There is clear benefit in running multiple threads in the text transfer type as this allows multiple Java threads perform translation on the data.

By running 2 concurrent file transfers we were able to match the data transfer rate of the binary transfer.

The actual cost of transferring the text files from Linux to MVS datasets is relatively constant regardless of the number of concurrent transfers.

WebSphere MQ File Transfer Edition Recommendations

The following are a list of bullet-pointed recommendations when planning your WebSphere MQ File Transfer Edition network when System Z is involved:

- When file sizes are small, send them over multiple concurrent transfers rather than a single large transfer. This increases the efficiency of the I/O involved in transferring the files as well as driving the MQ channel more efficiently.
- When transferring files that require translation, i.e. text-mode, it is advisable to run multiple concurrent threads.
- Consider whether a text transfer is required between Agents. Even if the Agents are moving data from compatible platforms, the cost of attempting unnecessary translations is significant.
- Transferring large numbers of files with long path names can result in large messages being put to the STATE queue – it is suggested that when transferring large numbers of files, the files are identified by a short path and file name.
- Test your typical transfers using a range of agentChunkSize parameters. Depending on the underlying hardware, you may find an optimum value for your setup.
- Multiple smaller files place the agent under strain due to the Operating System open/close costs associated with more files. Where possible, configure your file creation processes to generate archives of smaller files, enabling FTE to use less open/close calls.
- Reading and writing to physical disk is often going to be the performance constraint. For agents that will see a large number of incoming and outgoing transfers, it would be best if high performance disks were used to read data from and write data to.
- Ensure your Agent has sufficient memory available. On z/OS, prior to starting the Agent, we specified:
  ```
  export FTE_JVM_PROPERTIES="-Xmx1024M -Xms1024M"
  ```
- Default MQ channel settings allow MQFTE to work in its optimal manner. Overriding attributes such as NPMSPEED can have a significant detrimental effect.
- If NPMSPEED(NORMAL) is required on the MQ channel, it is advisable to test a range of typical transfers whilst varying the advanced tuning option. MQFTE has been optimised for use with NPMSPEED(FAST).
- Review supportPac MP16 “Capacity Planning and Tuning Guide” for advice on configuring your queue manager on MVS.

Hardware and Software

The hardware configuration was:

- **MVS:** **z10 EC64 (2097-EC64)** configured thus:
  LPAR 1
  Between 1 and 17 CP processors
  1 zAAP processor available.
  All processors bar '0' are able to be varied off.
  LPAR 2
  Between 1 and 3 CP processors available.
  All processors bar '0' are able to be varied off.
  LPAR 3
  Between 1 and 5 CP processors available.
  All processors bar '0'' are able to be varied off.
  Default configuration:
  3 dedicated CP processors on each LPAR
  zAAP varied off

  Coupling Facility
  3 shared processors (with priority weighted as 5 times other processors, taken from remaining 38 processors)

  DASD - FICON-connected Enterprise Storage Server (ESS) Model F20.
  A separate 3390 has been set aside for MVS dataset measurements. This has 8GB storage available and has been configured such that all MVS datasets prefixed "MQM.FTE.*" are written to it.
  The least used LCU on the system has been defined to be used when writing to the above volume.
  zFS is shared across the sysplex – and has a root from LPAR 2. When data is written to Unix System Service files, the storage group SGSYSTEM is used and at least 16 volumes prefixed "P3H" are available.
  zHPF is available but by default is configured as disabled.
  1 Gb Ethernet OSA Network Adapter.

- **Linux:**

eServer x366	zosdriver1
Processor	Intel® XEON™ MP CPU 3.66GHz
Architecture	4 CPU
Memory (RAM)	8GB
Disk	Internal Disks for measurements
Network	1Gbit Ethernet Adapter (onboard)

- **Network:**
  Performance Network Backbone is rated as 10 Gb Ethernet.
  1 Gb nodes

LPAR 1 and 2 are on same subnet	(9.20.36.n)
LPAR 3 and zosdriver1 are on same subnet	(9.20.37.n)

The software configuration was as follows:

MVS
Running z/OS 1.11 with Coupling Facility using CFCC level 16.

On each LPAR, 1 queue manager has been defined at V701 GA plus latest PTFs.

Each MVS queue manager are in the same queue sharing group, however agent queues will be defined as per configuration being measured.

Each MVS queue manager has been defined as follows:

- Archiving enabled, with
  - LOGLOAD=4000000,
  - PRIQTY=400,
  - SECQTY=50,
  - ALCUNIT=CYL
- 4 buffer pools –
  - 0 has 20,000 buffers,
  - 1 has 20,000 buffers,
  - 2 has 50,000 buffers,
  - 3 has 99,000 buffers
- Trace status:
  - TRACE(G) disabled,
  - TRACE(S) enabled,
  - TRACE(A) CLASS(1) enabled
- Channel Initiator is configured with:
  - CHIADAPS=30
  - CHIDISPS=20
  - MAXCHL=999
  - ACTCHL=200

Linux
ZOSDRIVER1 running Red Hat Linux 5.3 (Tikanga 64-bit)
Queue manager using WebSphere MQ v7.0.1 as per GA release.

CPU Cost Calculations on other System Z Systems

CPU costs can be translated from a measured system to the target system on a different z/Series machine by using Large Systems Performance Reference (LSPR) tables. These are available at:

http://www-03.ibm.com/systems/z/advantages/management/lspr/zOS19_SI_Oct_2008.html

This example shows how to estimate the CPU cost for a zSeries 2094-703 where the measurement results are for a 2097-703:

1. The LSPR gives the **2097-703** an Internal Throughput Ratio (ITR) of **4.15**
(this is for a “Mixed Workload”, which we found best fits WMQ in our environment).

2. As the 2097-703 is a 3-way processor, the single engine ITR is
4.15 / 3 = **1.383**

3. The “Mixed Workload” ITR of the **2094-703** used for the measurement is **2.72**.
The 2094-703 is also a 3-way processor. Its single engine ITR is
2.72 / 3 = **0.907**

4. The **2094-303 / 2097-703 single engine ratio** is
0.907 / 1.383 = 0.655 approx

this means that a single engine of a 2094-303 is two-thirds as powerful as that of a 2097-703.

5. Take a CPU cost of interest from this report, say **x** CPU microseconds (2097-303) per message, then the equivalent on a 2094-703 will be
x / 0.655 CPU microseconds/message

6. To calculate CPU busy, calculate using the number of processors multiplied either by 1000 (milliseconds) or 1000000 (microseconds) to find the available CPU time per elapsed second.

I.E. a 2097-703 has 3 processors so has 3,000 milliseconds CPU time available for every elapsed second.

So, for a CPU cost of interest from the report of 640 milliseconds on a 2097-703, the CPU busy would be:

640 / (3*1000) * 100 (to calculate as a percentage) = 21.33 %