



**John Burgess
Arndt Eade**

IBM CICS Performance Series: A Processor Usage Study of Ways into CICS

The IBM® CICS® Performance Series is a collection of IBM Redpaper™ publications that are focused on the performance of CICS and written by members of the IBM Hursley CICS development community. The topics are chosen based on feedback from CICS customers, with the goal of expanding your understanding of the rich features that are offered by the CICS Transaction Server (CICS TS) product portfolio.

This Redpaper publication is the fourth in the series, following these earlier publications:

- ▶ *IBM CICS Performance Series: CICS TS V4.2 and Java Performance*, REDP-4850
<http://www.redbooks.ibm.com/abstracts/redp4850.html?Open>
- ▶ *IBM CICS Performance Series: CICS, DB2, and Thread Safety*, REDP-4860
<http://www.redbooks.ibm.com/abstracts/redp4860.html?Open>
- ▶ *IBM CICS Performance Series: CICS and VSAM RLS*, REDP-4905
<http://www.redbooks.ibm.com/abstracts/redp4905.html?Open>

This latest publication, *A Processor Usage Study of Ways into CICS*, focuses on processor usage costs and documents the results of tests that are performed to assess some widely used methods of getting data in and out of CICS.

Introduction

This paper examines a server application program with the minimum amount of code that is required to receive data from a client. It highlights various scenarios that involve changing the last four bytes of data and returning the data to the client.

The number of measurements and combinations of parameters are limited to keep the study manageable. Except for the base 3270 scenario, which is limited to 1920 bytes, all scenarios are limited to 32 KB of data coming into CICS and 32 KB going back out to the client.

The tests represent only basic execution of the selected infrastructures. Although the performance characteristics are shown to be running within certain bounds, the qualities of service for each protocol might vary significantly. Therefore, the choice of protocols and infrastructure cannot always be based on performance alone.

These scenarios show the CICS costs and the costs that are incurred with all of the other address spaces that are associated with each scenario. The other address spaces are often overlooked in capacity planning, but must be included so that you can understand the full cost.

The following scenarios are included in this paper:

- ▶ 3270 clients
- ▶ CICS Web support
- ▶ CICS Transaction Gateway
- ▶ CICS Web services
- ▶ Communication Server IP Sockets
- ▶ IBM WebSphere® MQ
- ▶ Distributed Program Link
- ▶ IBM Virtual Telecommunications Access Method (IBM VTAM®) LU6.2 Distributed Transaction Processing

High-level summary

Figure 1 on page 3 summarizes the results from the tests that are documented in this paper. The graph shows the total microseconds of processor usage per transaction to get 32 KB of data into CICS and back to the client using the access methods tested. In all cases, only the processor usage on the server IBM MVS™ system is accounted for. For cases where TCP/IP was the transport protocol, data came in over a 1 GB OSA-Express3 channel card. Although the client MVS system was in the same sysplex, this usage was transparent because no sysplex optimizations such as IBM HiperSockets™ were used.

Figure 1 shows a high-level summary of study measurements.

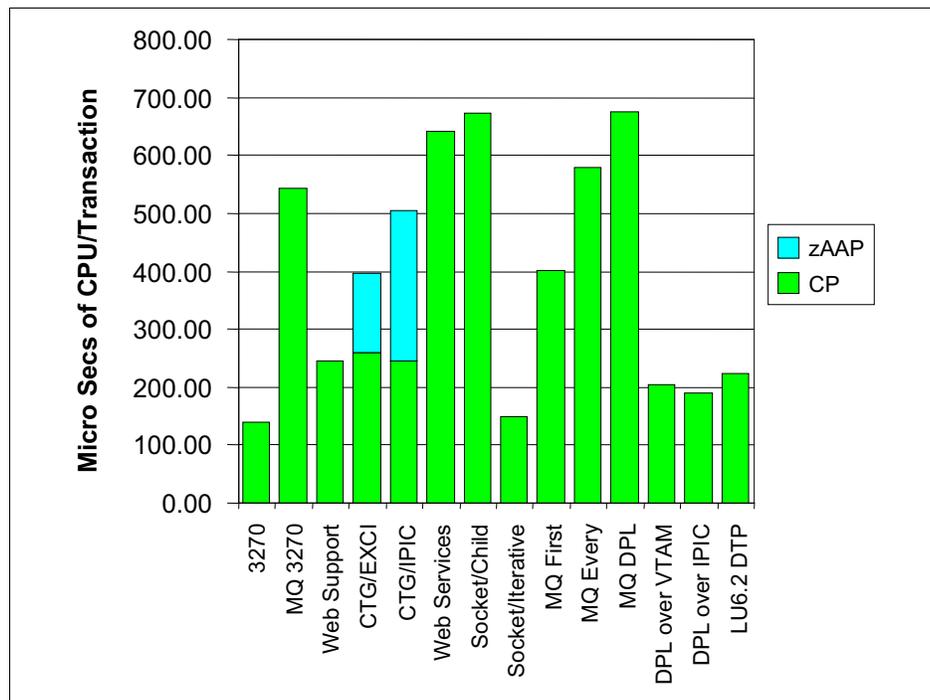


Figure 1 High-level summary of study measurements

For more information about any of the measurements that are shown in Figure 1, such as the individual address space breakdown, see the charts and tables in the relevant sections that follow.

Environment

The testing environment was set up as described here.

Hardware

The following hardware was used:

- ▶ IBM zEnterprise® 196 (Z196) 2817-779 model M80
- ▶ LPAR with three dedicated central processors (CPs) and one dedicated IBM System z® Integrated Information Processor (zIIP) (running System z Application Assist Processor (zAAP) on zIIP)
- ▶ DASD DS8800
- ▶ Internal coupling facility peer (ICP)
- ▶ 1 GB OSA-Express3
- ▶ Crypto Express3 coprocessor
- ▶ CP Assist for Cryptographic Function (CPACF) feature (enabled)

Software

The following software was used:

- ▶ IBM z/OS® Version 1, Release 13
- ▶ CICS Transaction Server Version 4, Release 2

- ▶ IBM Teleprocessing Network Simulator (TPNS)
- ▶ IBM WebSphere MQ Version 7, Release 1
- ▶ Communication Server IP Sockets for CICS
- ▶ CICS Transaction Gateway Version 8, Release 1

Methodology

The measurements were restricted to a single CICS region to process the data with transactions that arrive from different types of clients. The environments were reasonably tuned but no attempt was made to enhance the scalability of the environments, for example, by using multiple regions.

The workloads were driven mostly by TPNS simulating the different clients. TPNS was placed on another MVS system, away from the system that was being tested. The client *think times* were reduced to increase the load on the system, and five separate measurements were taken for each scenario. SMF data for IBM RMF™, CICS Statistics, and CICS Monitoring were collected and analyzed.

Think time: *Think time* is a delay in the processing of an HTTP request to reproduce the time that a human would take to read or examine the data that is displayed from a previous user action. Think time is calculated from the time that a request is received (that is, the display is complete on the monitor) until the time that the user clicks a key or link to perform an action.

The tables that are presented in this paper provide a summary of the data that is collected for each scenario. No measurements were taken for the client side.

Terminology

Become familiar with the following terms that are used throughout the paper:

ETR	The external transaction rate, or the number of transactions that are run per second.
Resp (ms)	The internal transaction response time, which is measured in milliseconds by IBM Resource Measurement Facility™ (RMF) or CICS Monitoring.
GCPU%	The percentage of one general-purpose CP used by the address space.
AAPCP%	The percentage of processor usage that was zAAP-eligible, but ran on a general-purpose CP. This number is included in the GCP% column.
zAAP CPU%	The percentage of processor usage that ran on zAAP. This number has to be added to GCPU% to determine the total processor usage.
Total ms CPU per Transaction	The total milliseconds of processor usage used by all the relevant address spaces for the scenario, divided by the transaction rate. The relevant address spaces are the individual spaces that are listed in the table of results for each scenario.

Data sources

The extracts of the applicable Resource Measurement Facility (RMF) reports show the origin of the data that is documented in the tables in this paper.

The GCPU% values in the tables in this paper come from the Workload Manager (WLM) report class for the address space in the Job Entry Subsystem (JES) subtype, as shown in Example 1.

Example 1 WLM report for the address space in the JES subtype

SERVICE TIME	---	APPL	%---
CPU	144.943	CP	80.84
SRB	99.908	AAPCP	0.00
RCT	0.000	IIPCP	0.00
IIT	0.000		
HST	0.000	AAP	N/A
AAP	N/A	IIP	0.00
IIP	0.000		

The figures for ETR and Resp (ms) come from the WLM report class for the CICS region for the CICS subtype, as shown in Example 2.

Example 2 WLM report class for the CICS region for the CICS subtype

-TRANSACTIONS-	TRANS-TIME	HHH.MM.SS.TTT
AVG	1.00	ACTUAL 13
MPL	1.00	EXECUTION 0
ENDED	3077678	QUEUED 0
END/S	10160.81	R/S AFFIN 0
#SWAPS	0	INELIGIBLE 0
EXCTD	0	CONVERSION 0
AVG ENC	0.00	STD DEV 68

The figures for zAAP CPU% and AAPCP come from the WLM report group for the JES subtype, as shown in Example 3.

Example 3 WLM report group for the JES subtype

SERVICE TIME	---	APPL	%---
CPU	221.605	CP	37.37
SRB	13.441	AAPCP	0.00
RCT	0.000	IIPCP	2.94
IIT	0.000		
HST	0.000	AAP	N/A
AAP	N/A	IIP	40.23
IIP	121.848		

The figures for Total ms CPU per Transaction are calculated by adding the processor usage that is used by the individual address spaces in the table (relevant address spaces) and dividing the result by the external transaction rate (ETR).

zAAP and zIIP: zAAP usage is referred to throughout this paper, but in the reports it is displayed as *System z Integrated Information Processor (zIIP)* usage. The reason is because the system is defined with a zIIP only, but it is running with zAAP on zIIP. Therefore, Java, which runs on a zAAP, is displayed on the zIIP.

Scenario: 3270 clients

Two scenarios are relevant to 3270 data streams:

- ▶ Basic 3270 terminals
- ▶ WebSphere MQ 3270 Bridge

Basic 3270 terminals

The 3270 scenario is included in this paper for completeness. The 3270 clients were simulated by using TPNS running in one LPAR in the same sysplex as the system under test. Data therefore came into CICS via XCFAS and the NET address spaces through a coupling facility (CF).

A simple 3270 data stream of the maximum 1920 bytes was sent into CICS. The application issued the **EXEC CICS RECEIVE** command, changed the last 4 bytes, and then returned the data by issuing the **EXEC CICS SEND** command. The application was not run as **CONCURRENCY (REQUIRED)**, and the **SEND** and **RECEIVE** commands were the bulk of this application. This scenario shows that there is no advantage because these commands are not threadsafe and require a Change Mode to the quasi-reentrant task control block (QR TCB) to process.

Raw data

Table 1 summarizes the performance of the data that was collected and shows the three main address spaces that were used in the scenario.

Table 1 3270 raw data

ETR	Resp (ms)	CICS GCPU%	NET GCPU%	XCFAS GCPU%	Total ms CPU per transaction
1245.14	6	9.73	5.71	2.50	0.144
2475.81	5	19.04	10.96	4.82	0.140
4925.46	5	36.77	21.09	9.37	0.136
10160.81	13	80.84	43.76	19.21	0.141

Table 2 shows the average processor usage or a transaction.

Table 2 Processor usage per transaction average

Average ms of CPU per Transaction	0.139
--	-------

Conclusions

As the transaction rate increases, the processor usage increases in a linear fashion. A maximum of 10,160 transactions per second was achieved for this single region. Further decreasing the think time did not provide any more throughput. Further investigation showed that, although neither the LPAR or the CICS region were processor usage-constrained, the point of contention was the dispatch time on the QR TCB. This scenario is shown in the extract from a 1-minute CICS Interval Statistics report. See Table 3.

Table 3 Accumulated times

MVS Waits	Accumulated time in MVS Wait	Accumulated time dispatched	Accumulated time/TCB
0	00:00:00.000000	00:01:00.982635	00:00:34.217026

By looking at the extract, you can see that, although the QR TCB only used 34 seconds of processor usage in the 1-minute interval, it was dispatched for 60 seconds with no MVS Waits. This status usually indicates a processor usage constraint at the address space level. It also indicates that the TCB is getting interrupted and higher priority work is getting dispatched by MVS rather than by the QR TCB after the interrupt was handled. However, in this case, the LPAR was only 57% busy. The contention is likely to be the interaction between the Virtual Telecommunications Access Method (VTAM) service request blocks (SRBs) running in the CICS address space and the QR TCB, such as a local lock contention.

WebSphere MQ 3270 Bridge

In this scenario, the client issues a non-persistent **PUT** command for a 1920-byte message to a remote queue that represents a WebSphere MQ 3270 bridge queue, and then issues a **GETWAIT** command with a correlation ID. The WebSphere MQ Manager then moves the message to the XMIT queue. The WebSphere MQ Channel then transmits this information to the server WebSphere MQ Manager over TCP/IP. The server WebSphere MQ puts this message on the Bridge Request queue. As required, this queue has a *TriggerType* FIRST event specified; therefore, a **CKBR** transaction starts.

The Bridge Monitor **CKBR** task, which is now constantly browsing this queue, recognizes that a *Start unit of work* message is waiting and so starts CKBP, the CICS 3270 bridge transaction that removes the message from the request queue and starts the user transaction.

The user transaction continues as a 3270-driven application. It issues a **CICS RECEIVE** command and then a **CICS SEND** command, returning 1920 bytes, which are placed on the relevant reply-to queue.

The message is then transmitted back to the client via the WebSphere MQ channel. The client program, which is in a **GETWAIT**, ends when it receives the message and the client transaction terminates.

Raw data

Table 4 summarizes the performance data that is collected and shows the four main address spaces that are used in the scenario.

Table 4 CICS WebSphere MQ 3270 Bridge for raw data

ETR	RESP (ms)	CICS GCPU%	MQ CHINN GCPU%	MQ MSTR GCPU%	TCP/IP GCPU%	TOTAL ms CPU per transaction
624.72	1	20.27	11.35	2.93	0.83	0.566
1245.94	1	41.61	20.71	5.94	1.40	0.559
1780.99	1	57.25	29.21	8.29	1.84	0.542
3511.26	1	111.38	54.39	15.69	3.31	0.526
5268.54	1	170.63	78.03	23.03	5.01	0.525

The ETR in Table 5 represents the number of user transactions per second (one for each message). It does not include the CKBP transaction. Therefore, effectively, we are counting business transactions. CKBP and the user transaction count as one.

Table 5 Processor usage per transaction average

Average ms of CPU per Transaction	0.543
--	-------

Conclusions

Keep in mind that, because this scenario uses the 3270 bridge, the maximum data that can be handled is 1920 bytes. This amount is compared to the 32 KB that are sent and received in most of the other measurement scenarios. Less data in the payload means less processor usage in the CICS, WebSphere MQ, and TCP/IP address spaces. It also means increased throughput because of faster transmission of data across the network that is compared to the larger 32 KB payloads in the other scenarios.

Summary

Figure 2 depicts the 3270 clients summary. It shows the distinction between the basic 3270 application interface and the use of the WebSphere MQ 3270 Bridge to access the same application.

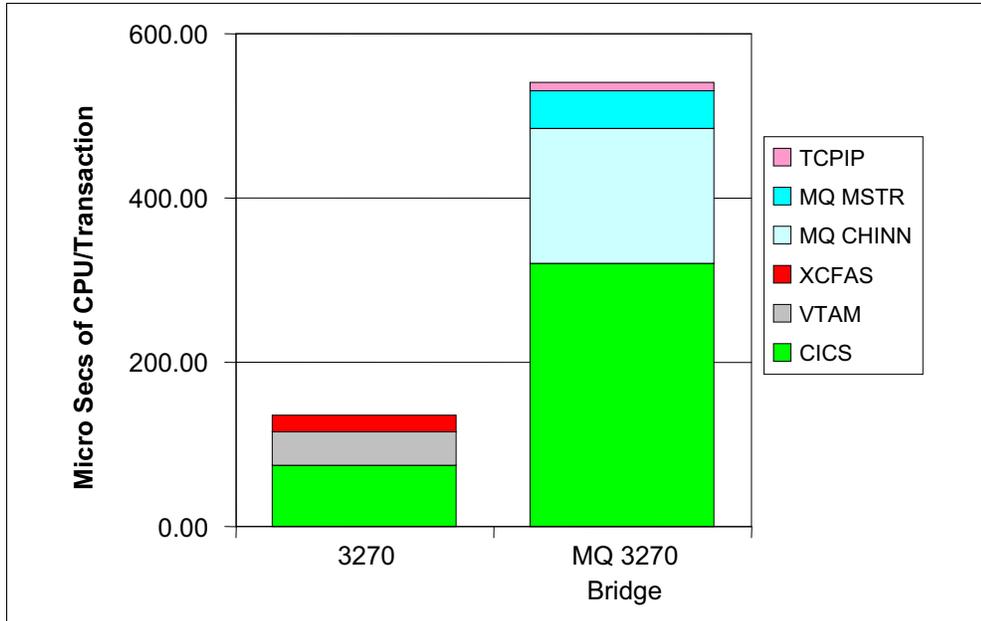


Figure 2 3270 clients summary

WebSphere MQ MSTR is the WebSphere MQ master address space. WebSphere MQ CHINN is the WebSphere MQ Channel Initiator address space that is used for moving messages to and from remote systems.

As with all of the summary charts in this paper, the cost in terms of microseconds of processor usage per transaction is taken from the average of the specified measurement points in the raw data tables.

Scenario: CICS Web support

Five scenarios are related to *CICS Web support*. Two scenarios relate to unsecured requests over HTTP connections, and three scenarios relate to secured requests over HTTPS.

HTTP

Non-pooled connections

In the CICS Web support scenario, the client is a CICS region which runs in a separate LPAR from the system under test. The client opens a session with the server CICS region and sends 32 KB of data by using the HTTP POST protocol. The server application issues an **EXEC CICS WEB RECEIVE** command, gets the data, changes it, and then sends it back to the client with the **EXEC CICS WEB SEND** command. The client application receives the data and then terminates.

In CICS V4R2, the client application can take advantage of *pooled connections*. With this facility, the application refers to a *Uniform Resource Identifier (URI) Map*, which is defined as

having a pool of connections that are already connected to a particular server. This configuration saves on processor usage by not having to connect and disconnect socket configurations.

The first format of the command, which is shown here, uses a non-persistent connection:

```
OPEN      EXEC  CICS WEB OPEN HOST(HOSTNAME)
          HTTP  SESSTOKEN(TOKEN) PORTNUMBER(4025) HOSTLENGTH(24)
```

This second format uses a pooled-persistent connection:

```
OPEN      EXEC  CICS WEB OPEN URIMAP('HTTPPOOL')
          SESSTOKEN(TOKEN)
```

Because these methods have performance differences, a comparison between the two scenarios is appropriate.

Raw data

Table 6 shows a summary of performance data for non-pooled HTTP connections.

Table 6 HTTP web support (non-pooled connections) for raw data

ETR	RESP (ms)	CICS GCPU%	TCP/IP GCPU%	TOTAL ms CPU per transaction
331.86	2	10.57	2.24	0.386
397.14	2	12.69	2.71	0.387
495.86	2	15.77	3.34	0.385
658.66	2	20.82	4.37	0.382
983.03	2	30.99	6.46	0.380

ETR in Table 7 represents the number of CWBA transactions per second.

Table 7 Processor usage per transaction average

Average ms of CPU per Transaction	0.384
--	-------

Conclusions

Running a web support client in CICS without using the pooled connection facility causes a new socket connection to be created for every new transaction. Not only does making these connections cost processor usage cycles, it limits the throughput (when the client is another CICS region) to about 1000 connections per second. This limitation occurs because for each connection, the client uses a new ephemeral port, and at the end of the transaction when the socket it closed, this ephemeral port stays in existence for approximately 60 seconds before it is deleted. An LPAR has a limit of 65,000 ports. Because these ports are included in contributing to this value, if we run much more than a 1000 new connections per second, this limit is soon reached (Concurrency = Rate x Time in System).

Pooled connections

Use of the *pooled connections* feature, which is available in CICS TS V4R2, is suggested if your client is a CICS application.

Raw data

Table 8 shows a performance summary of HTTP pooled connections.

Table 8 HTTP web support (pooled connections) for raw data

ETR	RESP (ms)	CICS GCPU%	TCP/IP GCPU%	TOTAL ms CPU per transaction
496.82	1	10.46	2.08	0.252
821.14	1	17.28	3.07	0.247
1232.03	1	25.88	4.06	0.243
2462.41	2	52.63	7.29	0.243
3560.45	9	76.77	11.69	0.248

ETR in Table 9 represents the number of CWBA transactions per second.

Table 9 Processor usage per transaction average

Average ms of CPU per Transaction	0.246
-----------------------------------	-------

Conclusions

As you can see from the results, using pooled connections is much more efficient in terms of processor usage per transaction because the cost of making and closing a connection is eliminated. The throughput rate that is achieved is also higher because it is not constrained by ephemeral TCP/IP port usage in the client LPAR.

HTTPS

This set of measurements used the same server application as the previous CICS Web support workload. However, this time the protocol was changed from HTTP to HTTPS by using the 3DES cipher suite.

Secure Sockets Layer (SSL) involves making an SSL handshake at the start of any new connection. If an SSL session ID is cached for this client from a previous full handshake, only a partial handshake has to occur. If not, a full handshake takes place. A full handshake runs on the Cryptographic Coprocessor if configured on the system, but it is much more expensive than a partial handshake, which is done in software.

With regards to CICS, specifying **SSLDELAY=0** in the server region forces a full handshake for every connection because the SSL session ID is never retained. Specifying, for example, **SSLDELAY=600** will cause the SSL session ID to be retained and reused for up to 600 seconds if the same client comes back after closing its connection.

For a persistent connection, the handshake is made once and then for subsequent data flows, only encryption and decryption on the data has to occur.

Encryption and decryption for Triple Data Encryption Algorithm (3DES) is done by using System SSL over CP Assist for Cryptographic Function (CPACF).

Three scenarios for SSL were run covering the preceding descriptions, all sending and receiving 32 KB of data:

- ▶ Non-persistent connections. No SSL session ID is retained; a full handshake occurs.
- ▶ Non-persistent connections. An SSL session ID is retained; a partial handshake occurs.
- ▶ Pooled persistent connections. No handshake occurs; there is only encryption and decryption of the 32 KB payload.

The cipher suite that is used on the CICS TCPIP SERVICE for all of these HTTPS measurements was *0A*, which relates to the following settings:

- ▶ Cipher Suite: Triple Des
- ▶ Key length: 168 bytes
- ▶ MAC algorithm: SHA-1
- ▶ Key Exchange: RSA

Full handshake

For non-persistent connections, no SSL session ID is retained. A full handshake occurs.

Raw data

Table 10 shows a performance summary of data for HTTPS connections with full handshakes.

Table 10 HTTPS CICS Web support (full handshake) for raw data

ETR	RESP (ms)	CICS GCPU%	TCP/IP GCPU%	TOTAL ms CPU per transaction
331.05	6	37.42	3.49	1.235
395.98	7	45.43	4.08	1.250
492.21	7	57.36	4.77	1.262
637.63	15	77.59	6.09	1.312
787.34	51	97.99	7.49	1.548

ETR in Table 11 represents the number of CWBA transactions per second.

Table 11 Processor usage per Transaction average

Average ms of CPU per Transaction	1.321
--	-------

Conclusions

In this scenario, the SSLDELAY in the CICS server address space was set to 0 so that an SSL session ID is never reused and each connection has to do a full handshake. With a Crypto Express3 coprocessor configured on this system, all of these full handshakes are done on this hardware, as shown in the RMF extract in Example 4 on page 13. However, the full handshake is the most expensive part of SSL processing. When possible, for better performance, avoid this step by using an SSLDELAY value that is greater than zero or by using persistent connections.

Example 4 Cryptographic coprocessor summary

```

----- CRYPTOGRAPHIC COPROCESSOR -----
----- TOTAL ----- KEY-GEN
TYPE  ID  RATE  EXEC TIME  UTIL%  RATE
CEX3C  0  0.00  0.000    0.0    0.00
      1  0.00  0.000    0.0    0.00
      2  0.02  0.192    0.0    0.00
      3  0.00  0.000    0.0    0.00
      4  0.00  0.000    0.0    0.00
      5  0.00  0.000    0.0    0.00
      6 787.7  0.710   56.0    0.00
      7 1579  0.577   91.1    0.00
  
```

The RMF extract in Example 4 shows SSL full handshakes being done on the Cryptographic coprocessor. ID 6 is the hardware that is associated with this LPAR and shows a rate of 787 handshakes per second, which matches the transaction rate of the last measurement point. To see which hardware is associated with an LPAR, use the Integrated Cryptographic Support Facility (ICSF) in TSO.

Partial handshake

For non-persistent connections, an SSL session ID is retained. A partial handshake occurs.

Raw data

In this scenario, the **SSLDELAY** parameter in the CICS server region was set to 600. Pooled connections were not used in the client region; therefore, each new transaction still resulted in a new TCP/IP connection being created for each web open request. Because the SSL session ID from the original handshake from this client was cached due to the SSLDELAY setting, it can be reused with this new connection and a partial handshake can be done instead of a full handshake. See Table 12 for a partial handshake for raw data.

Table 12 HTTPS CICS Web support (partial handshake) for raw data

ETR	RESP (ms)	CICS GCPU%	TCP/IP GCPU%	TOTAL ms CPU per transaction
330.39	3	30.71	2.78	1.013
397.52	3	37.47	3.49	1.030
495.13	3	46.56	4.29	1.027
658.67	3	62.08	5.67	1.028
981.57	3	92.21	8.25	1.023

ETR in Table 13 represents the number of CWBA transactions per second.

Table 13 Processor usage per transaction average

Average ms of CPU per transaction	1.041
--	-------

Conclusions

As can be seen in the data, the use of a partial handshake is less expensive in terms of processor usage than using a full handshake. As in the full handshake, a partial handshake is performed by the *CWXXN* transaction. CWXXN also decrypts inbound data. CWBA encrypts outbound data.

No handshake, pooled connections

For pooled persistent connections, no handshake occurs. There is only encryption and decryption of the 32 KB payload.

Raw data

In Table 14, the client CICS application used pooled connections as described in the CICS Web support section. Pooled connections mean that they are maintained as persistent connections and after the initial full handshake, no more handshaking is required. The cost for SSL in this environment is the encryption and decryption only.

Table 14 HTTPS CICS Web support (no handshake, pooled connections) for raw data

ETR	RESP (ms)	CICS GCPU%	TCP/IP GCPU%	TOTAL ms CPU per transaction
495.85	1	28.68	2.07	0.620
821.38	1	47.29	3.28	0.615
1223.66	1	69.96	4.65	0.609
2384.40	2	136.3	8.64	0.607
3499.03	10	205.10	12.62	0.622

ETR in Table 15 represents the number of CWBA transactions per second.

Table 15 Processor usage per transaction average

Average ms of CPU per Transaction	0.614
-----------------------------------	-------

Conclusions

As demonstrated, avoiding SSL handshakes is the best option for performance, both in terms of processor usage and throughput. As shown in Table 14, only the encryption and decryption of the 32 KB of data was performed. A rate of 3499 transactions per second was reached with CICS by using 2.05 CPs concurrently. Most of this processor usage time was accumulated on the S8 TCBs, as shown in the 1-minute interval statistics extract in Example 5 for the highest transaction rate: 73 seconds of processor usage time on the S8 TCBs, 38 seconds on the QR TCB, and a further 19 seconds on the SO TCB. Therefore, for SSL workloads, getting enough S8 TCBs is essential to maintain performance.

Example 5 HTTPS CICS Web support, no handshake

TCB Mode	< TCBs Attached > Current	> Peak	TCB Attaches	Attach Failures	MVS Waits	Accumulated Time in MVS wait	Accumulated Time Dispatched	Accumulated Time / TCB
QR	1	1	0	0	379875	00:00:21.818431	00:00:41.164024	00:00:38.716120
SL	1	1	0	0	131990	00:01:02.093076	00:00:00.889425	00:00:00.893424
SO	1	1	0	0	1268919	00:00:43.914826	00:00:19.067638	00:00:19.266394
S8	130	130	1	0	2698526	02:46:08.936419	00:02:02.565503	00:01:13.496820

Summary

Figure 3 shows the CICS Web support summary. The graph depicts the relative costs of using CICS Web support to send a 32 KB message into CICS and receive a 32 KB message back at the client. The chart also shows the cost using HTTP with a non-persistent connection, where a new connection is made for each message that is sent into CICS and HTTP with a persistent connection.

The graph in Figure 3 depicts the cost of the three HTTPS scenarios: A full SSL handshake for each message, a partial SSL handshake for each message (where the SSL IF is retained), and a persistent connection with no handshake, just encryption and decryption.

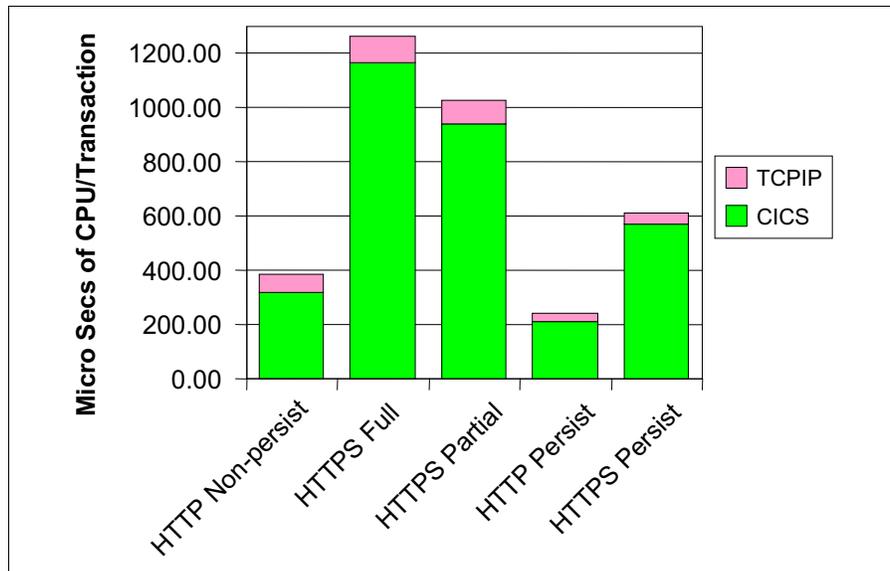


Figure 3 CICS Web support summary

Scenario: CICS Transaction Gateway

Two scenarios relate to CICS Transaction Gateway (CICS TG):

- ▶ External CICS interface (EXCI) connections
- ▶ Internet Protocol interconnectivity (IPIC) connections

External CICS interface

This scenario consists of 32 KB *external call interface* (ECI) requests being sent into CICS over a transaction gateway running on z/OS. Each of the clients maintained a persistent connection with the gateway. The connection between the gateway and CICS was EXCI.

CTG AAPCP: CTG AAPCP is the percentage of one CP that was eligible to run on a zAAP but ran on a general-purpose CP. This number is included in the CTG GCP CPU% column.

In these measurements, we set **IFAHONORPRIORITY=YES**, which enabled zAAP-eligible code to spill over and run on a general-purpose CP if the zAAP was currently busy. With **IFAHONORPRIORITY=NO**, all zAAP eligible code must process on a zAAP.

Table 16 summarizes the performance figures for the CICS Transaction Gateway by using EXCI connections.

Table 16 CICS Transaction Gateway (EXCI) for raw data

ETR	RESP (ms)	CICS GCPU%	TCP/IP GCPU%	CTG GCP CPU%	CTG AAPCP	CTG zAAP CPU%	TOTAL ms CPU per trans.
329.65	1	3.58	2.14	4.21	0.18	5.36	0.463
658.32	1	7.09	3.22	6.89	0.39	9.90	0.411
962.08	1	10.26	3.95	10.26	0.65	13.10	0.390
1822.53	1	19.02	6.58	18.66	1.07	22.52	0.366
3564.75	1	37.11	11.73	37.37	2.94	40.23	0.354

Table 17 shows the average processor usage cost per transaction.

Table 17 Processor usage per transaction average

Average ms of CPU per transaction (including zAAP)	0.396
Average ms of CPU per transaction (excluding zAAP)	0.259

Internet Protocol interconnectivity

In this scenario, 32 KB ECI requests were sent into CICS through a transaction gateway that runs on z/OS. Each of the clients maintained a persistent connection with the gateway. The connection from the gateway to CICS was over IPIC.

Table 18 summarizes the performance figures for CICS Transaction Gateway using IPIC connections.

Table 18 CICS Transaction Gateway (IPIC) for raw data

ETR	RESP (ms)	CICS GCPU%	TCP/IP GCPU%	CTG GCP CPU%	CTG AAPCP	CTG zAAP CPU%	TOTAL ms CPU per trans.
329.89	1	5.62	2.27	0.66	0.50	8.93	0.529
658.76	1	11.34	3.48	1.03	0.72	17.86	0.511
982.36	1	17.87	4.08	1.50	1.09	26.46	0.508
1940.74	1	34.86	7.29	4.79	4.33	48.64	0.492
3408.16	1	59.68	12.54	12.24	11.75	80.15	0.482

Table 19 shows the processor usage cost per transaction.

Table 19 Processor usage per transaction averages

Average ms of CPU per Transaction (including zAAP)	0.504
Average ms of CPU per Transaction (excluding zAAP)	0.245

Conclusions

In the CICS Transaction Gateway scenarios, EXCI used the least amount of processor usage per transaction when compared to the IPIC protocol. However, with EXCI, the data transfer was restricted to 32 KB. Because the IPIC configuration can use channels and containers, there is no restriction on the data transfer size.

In these measurements, all of data was passed in CICS communication areas (COMMAREAs).

Most of the processing in the transaction gateway address space was zAAP-eligible.

Summary

Figure 4 shows the CICS Transaction Gateway summary. The graph shows the difference in processor usage cost between EXCI and IPIC. The CICS address space costs were less when using EXCI, because of the efficiency of using cross-memory MRO for communication between the gateway address space and the CICS address space. In the gateway address space, although the overall cost to use IPIC was greater, the amount of processing offloaded to a zAAP was also greater, which made the use of general-purpose CPs in the gateway address space less critical for IPIC.

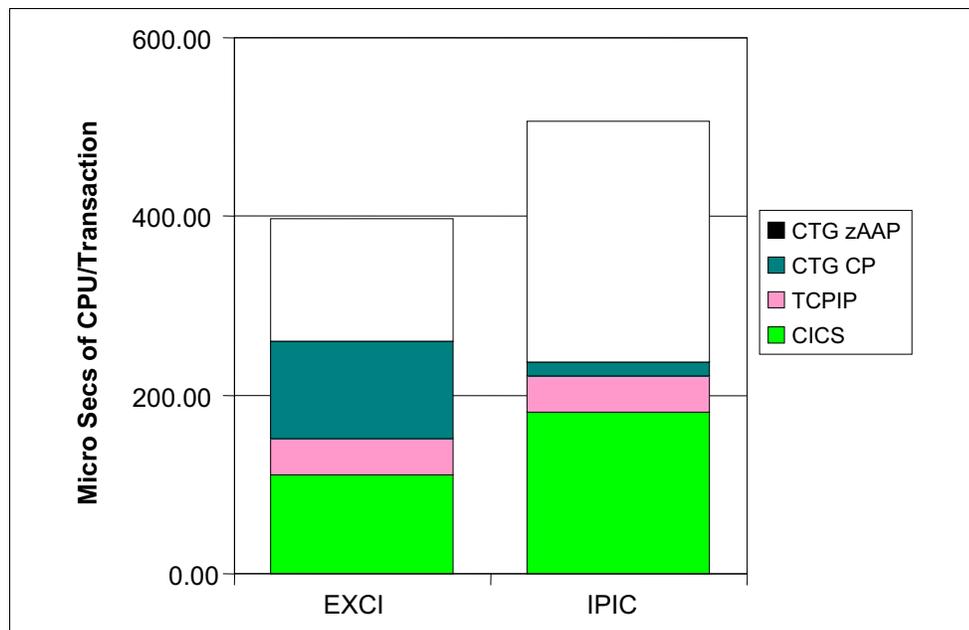


Figure 4 CICS Transaction Gateway summary

Much of the CICS Transaction Gateway is written in Java. Therefore, improved performance is most likely if optimization by the Java just-in-time compiler (JIT) can be used. Leaving the gateway address space running for extended periods of time without restarting enables the JIT to collect extensive sampling data and enables recompilation of frequently used methods to the appropriate level of optimization.

Scenario: CICS Web services

Two scenarios relate to *CICS Web services*:

- ▶ Use of a simple XML message.
- ▶ Use of a more complex XML message.

One element of 32 KB

In this scenario, the clients are CICS applications that run in a CICS region on another zOS system. A client transaction uses a pooled HTTP connection and sends a web service request to the server CICS address space. The server region attaches the CWXN transaction, which after determining this is a web service request using the URI map, starts the *CPIH* transaction. CPIH runs the web services pipeline, which decomposes the XML into, in this case, a CICS container, and links to the target COBOL application program. The application program returns the changed container and the pipeline transforms it back to XML to be transmitted to the client.

The client transaction receives a response and then terminates.

Key to the performance of CICS Web services is the complexity of the XML. In all other scenarios, except for the 3270, 32 KB data flows were used. The same approach was used here, too, but with two benchmarks: The first measurement was a single XML element that is 32 KB in length, and the second measurement was 1000 elements that are 32 bytes in length. Table 20 summarizes the results of this scenario.

Table 20 *CICS Web services (single 32k byte element) for raw data*

ETR	RESP (ms)	CICS GCPU%	TCP/IP GCPU%	TOTAL ms CPU per transaction
820.64	1	47.06	3.72	0.618
1633.58	2	95.77	5.75	0.621
2339.00	2	139.22	8.83	0.632
2558.38	5	158.47	11.30	0.663
2711.30	7	171.24	11.49	0.673

ETR in Table 21 represents the number of CWBA transactions per second.

Table 21 *Processor usage per transaction average*

Average ms of CPU per transaction	0.641
-----------------------------------	-------

CICS XML processing: Some of the CICS XML processing can be offloaded to a zAAP through the support of XML system services. However, in this scenario, the offload was insignificant and is therefore not included in Table 20 on page 18.

When the Axis2 web services SOAP engine is used in CICS, the zAAP offload is a much higher proportion of the total. However, we did not include Axis2 in this study because its principal role in CICS is in providing improved programming facilities for web service provider applications when handling complex WSDL documents. Or, when standardized Java based web service application programming interfaces (APIs) are required.

1000 elements of 32 bytes

In Table 22, 32 KB of data were transmitted. However, this time, instead of sending them in just a single element, the complexity of the XML was increased by using 1000 elements of 32 bytes each.

Table 22 CICS Web services (1000 32-byte elements) for raw data

ETR	RESP (ms)	CICS GCPU%	CICS AAPCP	CICS zAAP CPU%	TCP/IP GCPU%	TOTAL ms CPU per trans.
413.08	3	53.78	0	5.43	3.35	1.514
495.06	3	64.20	0	6.47	3.92	1.501
617.36	3	79.99	0	8.03	4.62	1.500
818.18	3	105.77	0	10.49	5.72	1.490
1527.27	5	200.95	0	18.36	7.97	1.488

ETR in Table 23 represents the number of CPIH transactions per second.

Table 23 Processor usage per transaction averages

Average ms of CPU per transaction (including zAAP)	1.498
Average ms of CPU per transaction (excluding zAAP)	1.371

No AAPCP in this environment: There was no AAPCP in this environment, which indicates that all code that was eligible to run on a zAAP actually ran on a zAAP. IFAHONORPRIORITY was set to NO, which forced all zAAP code onto a zAAP. Be careful with this setting, because it can cause code to wait for the processor if a zAAP is not available at that specific moment.

Conclusions

One of the main factors that affects CICS Web services performance in terms of processor usage is the complexity of the XML. Two scenarios were run, one with a single 32 KB element and one with 1000 32-byte elements.

Part of the XML processing in CICS is offloaded to a zAAP processor. With the single-element scenario, the offload processing was insignificant. However, with 1000 elements, the offload was significant because the amount of processing of the XML elements increased.

Reducing the complexity of XML is a key element in CICS Web services performance.

Summary

Figure 5 shows the CICS Web services summary. The graph depicts a comparison between CICS Web services using a simple one-element XML structure and a more complex 1000-element XML structure.

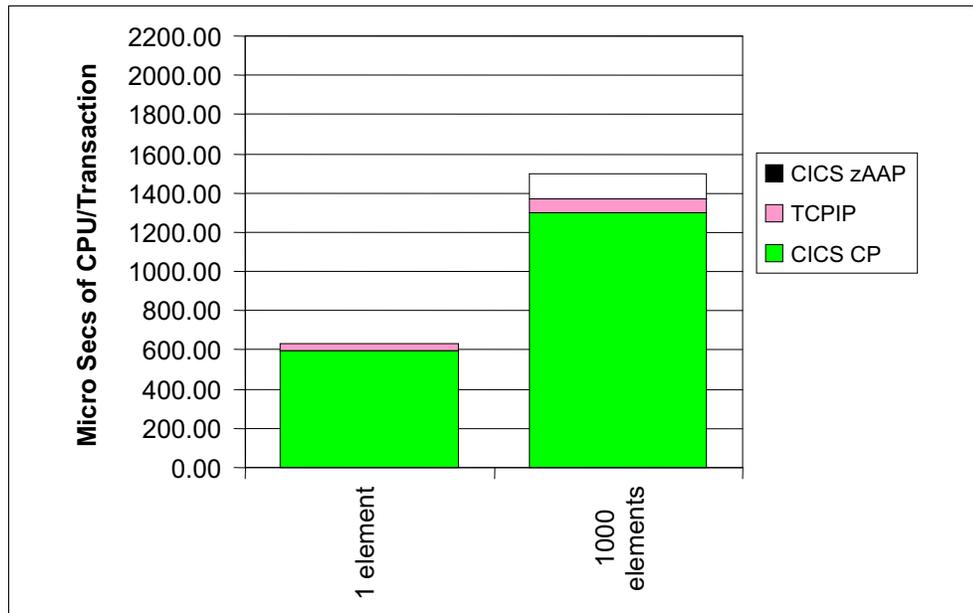


Figure 5 CICS Web services summary

Scenario: Communication Server IP Sockets

The Communication Server IP Sockets API has two main programming models, the *concurrent child server* and the *iterative server*.

Concurrent child server

A CICS transaction that runs in a client CICS region uses the Communications Server IP socket interface to connect to a CICS server region. A 32K byte message is sent to the server. The CICS server port, monitored by the IBM-supplied enhanced concurrent server, receives the request and issues an **EXEC CICS START** command for a child transaction passing the socket by using the GIVESOCKET API call. The child transaction issues a TAKESOCKET request to take ownership of the socket and issues a READ request until all data is received. The 32 KB message is then sent back to the client using a WRITE request. When complete, both client and child server transactions issue a socket CLOSE request and terminate.

Table 24 summarizes the results of this scenario.

Table 24 Communication Server IP Sockets (concurrent child server) for raw data

ETR	RESP (ms)	CICS GCPU%	TCP/IP GCPU%	BPXOINIT GCPU%	TOTAL ms CPU per transaction
495.51	4	24.03	3.83	5.41	0.671
618.69	4	30.05	4.75	6.56	0.668
1223.84	4	59.26	9.06	12.61	0.661
2124.87	5	102.54	14.05	23.53	0.659
3162.22	5	159.77	21.41	40.71	0.701

Table 25 shows the average processor usage cost per transaction.

Table 25 Processor usage per transaction average

Average ms of CPU per Transaction	0.672
--	-------

Iterative server

A CICS transaction that runs in a client CICS region uses the Communications Server IP socket interface to connect to a CICS server region. A 32 KB message is sent to the server. The CICS server port is monitored by a long-running *iterative server*. The iterative server issues an ACCEPT request for the client socket and issues multiple READ requests until the 32 KB message is read. The message is then returned to the client by using a WRITE request. When complete, the iterative server issues a CLOSE request and enters a Wait status for the next socket request. After receiving the 32 KB message back, the client CICS transaction issues a socket CLOSE request and terminates.

Iterative server: The *iterative server* is a long running transaction. Because no new transactions were attached during the measurement period, the ETR in this case is a count of transactions that are completed in the client region. The response times are also those that are recorded in the client.

Table 26 summarizes the performance data for this scenario.

Table 26 Communications Server IP Sockets (iterative server) for raw data

ETR	RESP (ms)	CICS GCPU%	TCP/IP GCPU%	BPXOINIT GCPU%	TOTAL ms CPU per transaction
382.61	111	2.93	3.23	NA	0.160
420.65	194	3.22	3.49	NA	0.159
598.51	231	4.34	4.62	NA	0.149
797.06	187	5.79	5.93	NA	0.147
957.62	166	6.51	6.26	NA	0.133

Table 27 on page 22 gives the average processor usage cost per transaction.

Table 27 Processor usage per transaction average

Average ms of CPU per Transaction	0.149
-----------------------------------	-------

Conclusions

Clearly, the iterative child server has no processor usage for starting a new transaction for each message that arrives. The iterative child server has no processor usage for starting a new transaction for each message that arrives. However, in this iterative model, where there is only one server application running, the process of taking messages from TCP/IP, linking to the application code, and sending data back to the client across the network is single threaded, and 957 transactions per second was the maximum that could be achieved without starting multiple servers. Also, with the iterative child server, there was no processor usage that was used in the BPXOINIT address space, because there is no constant creation of sockets.

Summary

Figure 6 depicts the Communication Server IP Sockets summary. The graph shows that the iterative model had considerably smaller processor usage. This result is partly because there was only one long running task, which eliminated the cost of transaction initiation and termination for each message that needed processing. However, as shown by the reduced throughput that was achieved, this model is less scalable and therefore requires multiple long running tasks to service higher volumes of messages. It also requires a mechanism to manage the appropriate numbers of these long running tasks.

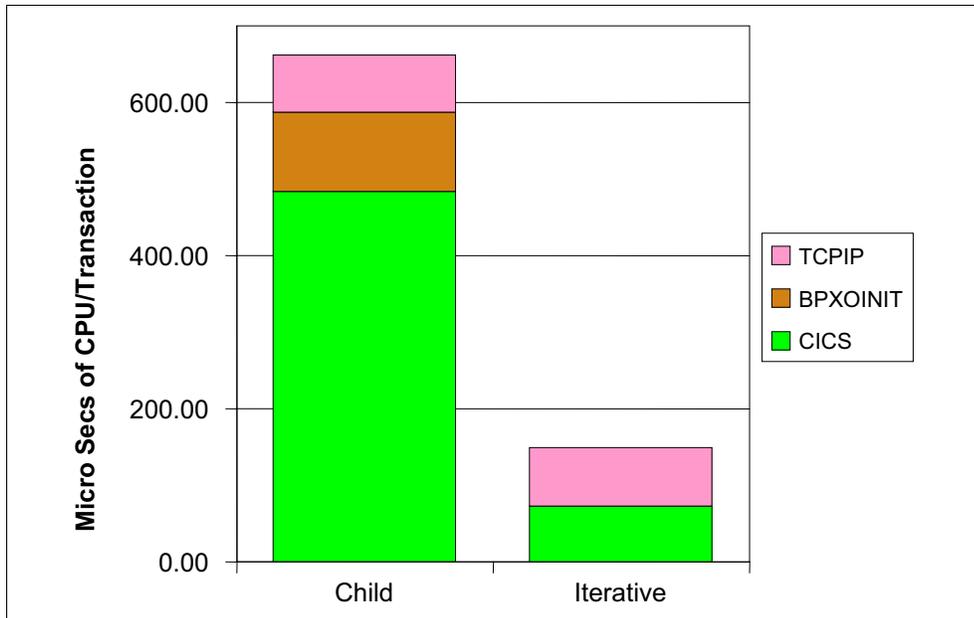


Figure 6 Communication Server IP Sockets summary

Scenario: CICS and WebSphere MQ

We now describe three CICS and WebSphere MQ scenarios. The first scenario is for WebSphere MQ messages that are triggered by using the *TriggerType* EVERY event. The second scenario is for WebSphere MQ messages that are triggered by using the *TriggerType* FIRST event. The third scenario is for messages that enter CICS by using the *distributed program link (DPL) bridge*.

TriggerType EVERY

In this scenario, the client issues a non-persistent **PUT** command for a 32 KB message to a remote queue. The WebSphere MQ Manager then moves the message to the XMIT Queue. The WebSphere MQ channel then transmits this information to the server WebSphere MQ Manager over TCP/IP. The server WebSphere MQ puts this message on the Application Queue. Because this queue has a *TriggerType* EVERY event associated with it, for every message it receives, it puts a trigger message on the Initiation Queue. The long-running transaction CKTI is in a GETWAIT state on this Initiation Queue. When it receives a message, it starts the associated user transaction and tells it which Application Queue that it must read for the application data message.

The user transaction gets the message and then sends a reply by putting a 32 KB message on the specified reply-to queue. At this point, the user application ends and the transaction terminates. The reply message is transmitted back via TCP/IP to the client, which is in a GETWAIT state. When the client GETWAIT state is satisfied, the client transaction ends.

Raw data

Table 28 summarizes the performance data for this scenario.

Table 28 CICS WebSphere MQ (TriggerType EVERY) for raw data

ETR	RESP (ms)	CICS GCPU%	MQ CHINN GCPU%	MQ MSTR GCPU%	TCP/IP GCPU%	TOTAL ms CPU per trans.
624.87	1	15.11	18.12	1.38	2.41	0.592
1240.83	1	30.17	34.51	2.72	4.63	0.580
1648.86	1	40.16	45.28	3.59	6.06	0.576
1762.77	1	43.60	48.21	3.82	6.50	0.579
2071.36	1	50.68	54.62	4.53	7.46	0.566

Table 29 gives the average processor usage cost per transaction.

Table 29 Processor usage per transaction average

Average ms of CPU per Transaction	0.578
-----------------------------------	-------

Conclusions

The *TriggerType* EVERY setting causes a new transaction to be initiated for every message that is placed on the associated application queue. Although transaction initiation and termination adds pathlength that is compared to the *TriggerType* FIRST scenario that is presented next, *TriggerType* EVERY scales well naturally, without the need for mechanisms to ensure the appropriate numbers of long running server transactions.

TriggerType FIRST

In this scenario, the client issues a non-persistent **PUT** command for a 32 KB message to a remote queue. The WebSphere MQ Manager then moves the message to the XMIT Queue. The WebSphere MQ channel then transmits this information to the server WebSphere MQ Manager over TCP/IP. The server WebSphere MQ puts this message on the Application Queue, which has a *TriggerType* FIRST event that is associated with it. Therefore, when the first message is received, it puts a trigger message on the Initiation Queue. The long running transaction CKTI is in a GETWAIT state on this Initiation Queue. When it receives this first message, it starts the associated user transaction and tells it which application queue that it must read for the application data message.

The user transaction gets the message and then sends a reply by putting a 32 KB message on the reply-to queue. At this point, the user application loops back to a GETWAIT state. This user transaction is long running and never ends. The reply message is transmitted back via TCP/IP to the client, which is in a GETWAIT state. When the client GETWAIT status is satisfied, the client transaction ends.

Raw data

Table 30 summarizes the performance data for this scenario.

Table 30 CICS WebSphere MQ (TriggerType FIRST) for raw data

ETR	RESP (ms)	CICS GCPU%	MQ CHINN GCPU%	MQ MSTR GCPU%	TCP/IP GCPU%	TOTAL ms CPU per trans.
624.50	6	6.44	17.09	0.51	2.39	0.423
1242.05	7	12.89	31.81	0.98	4.42	0.403
1758.57	8	18.30	43.99	1.37	6.11	0.396
1850.00	12	19.25	46.27	1.44	6.42	0.396
2093.35	102	22.18	51.36	1.63	7.11	0.393

Table 31 shows the average processor unit cost per transaction.

Table 31 Processor usage per transaction average

Average ms of CPU per transaction	0.402
-----------------------------------	-------

One long running transaction: In the preceding scenario, there was only one long running transaction in the server region. Therefore, the ETR and response times are taken from the client address space.

Conclusions

In this scenario, we see that the processor usage per transaction is less than the *TriggerType* EVERY case that was presented earlier. This outcome is because we avoided the additional cost of transaction initialization and termination for each message that arrives in the system. There was only one long running transaction taking messages off the queue and returning the response; therefore, the throughput that was achieved was not as high as with *TriggerType* EVERY, where there were multiple tasks processing messages. Using the *TriggerType* FIRST event requires a mechanism for starting the appropriate number of server transactions that process messages for achieving high volumes in busy systems.

Distributed program link bridge

In this scenario, the client issues a non-persistent **PUT** command for a 32 KB message to a remote queue that represents a WebSphere MQ distributed program link (DPL) Bridge queue. The WebSphere MQ Manager then moves the message to the XMIT Queue. The WebSphere MQ channel then transmits this information to the server WebSphere MQ Manager over TCP/IP. The server WebSphere MQ puts this message on the Bridge Request Queue. As required, this queue has a *TriggerType* FIRST event specified; therefore, a CKBR transaction starts.

The Bridge Monitor **CKBR** task, which is now constantly browsing this queue, recognizes that a *Start unit of work* message is waiting and starts the **CKBP** transaction. This action removes the message from the request queue. It builds the COMMAREA from the relevant message and issues an **EXEC CICS LINK** command to the required user application passing the COMMAREA.

The user application returns the response in the COMMAREA and the **CKBP** transaction creates a message from it and puts the message on the specified reply-to queue.

The message is then transmitted back to the client via the WebSphere MQ channel. The client program, which is in a GETWAIT state, ends when it receives the message and the client transaction terminates.

Raw data

Table 32 summarizes the performance data for this scenario.

Table 32 CICS WebSphere MQ DPL bridge for raw data

ETR	Resp (ms)	CICS GCPU%	MQ CHINN GCPU%	MQ MSTR GCPU%	TCP/IP GCPU%	TOTAL ms CPU per trans.
624.16	1	21.99	18.05	1.90	2.50	0.711
1242.02	1	42.46	33.97	3.73	4.80	0.684
1637.66	1	56.12	43.07	4.88	6.20	0.673
1755.82	1	59.75	44.61	5.16	6.46	0.660
2147.58	1	70.92	54.90	5.87	8.07	0.650

Table 33 gives the average processor usage cost per transaction.

Table 33 Processor usage per transaction average

Average ms of CPU per transaction	0.675
--	-------

Conclusions

In this scenario, the CICS address space processor usage per transaction is higher than in the basic GET/PUT case. However, this is the cost of the initiation of the DPL bridge facility for each message. The cost in all of the other address spaces is similar to the other WebSphere MQ scenarios because the same amount of data is moved through the system.

Summary

Figure 7 depicts the CICS WebSphere MQ summary. The graph shows a comparison between the processor usage costs of using the *TriggerType* FIRST and *TriggerType* EVERY events. Although *TriggerType* FIRST has a lower processor usage per transaction, this is a single-threaded model. Therefore, to handle high throughput rates, a mechanism must be put in place to handle starting the appropriate number of long running transactions for the required message rate.

Figure 7 also shows the cost of using the WebSphere MQ DPL bridge facility to access the network application by using an `EXEC CICS LINK` command.

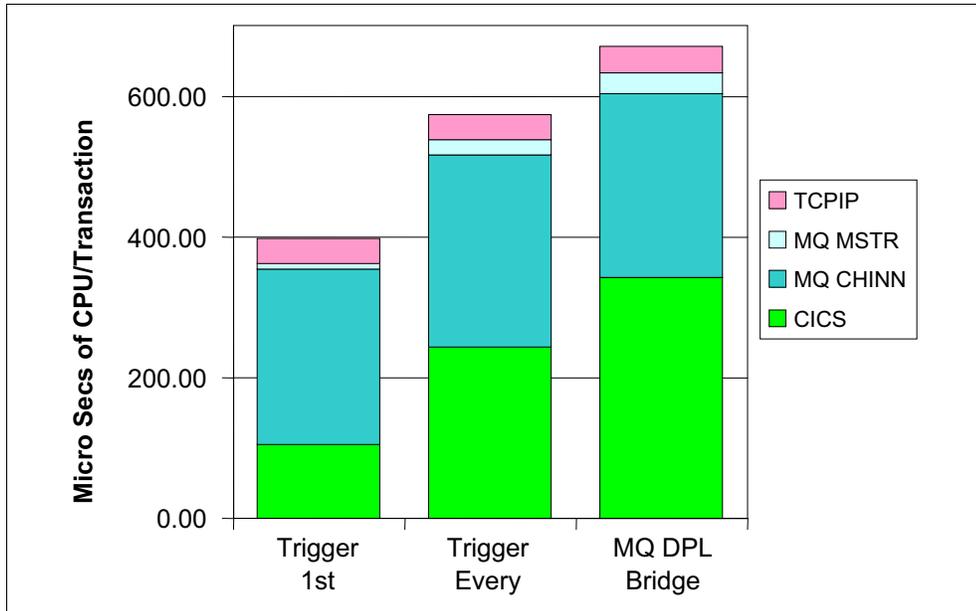


Figure 7 CICS WebSphere MQ summary

Scenario: Distributed program link

The distributed program link scenario shows the measurement data that is taken from an environment where a CICS client that is running on one MVS system issues a DPL request to a server application that is running in CICS on another MVS system. The client sends 32 KB of data with a `COMMAREA` by using the `SYNCONRETURN` command. The server application is the same server program that receives the 32 KB of data, changes it, and then returns 32 KB back to the client. The client application then terminates.

Two environments are now described. In one environment, the data flows by using an IPIC connection over TCP/IP through the OSA. In the other environment, the data flows over VTAM LU6.2 across a channel-to-channel (CTC) from the client CICS region on one MVS to a server region on another MVS system.

Over Internet Protocol interconnectivity

Table 34 summarizes the performance data for this scenario.

Table 34 DPL over IPIC for raw data

ETR	RESP (ms)	CICS GCPU%	TCP/IP GCPU%	TOTAL ms CPU per transaction
496.45	3	7.93	1.58	0.191
832.93	3	13.27	2.57	0.190
1244.97	4	19.91	3.77	0.190
2723.33	10	43.29	8.04	0.188

Table 35 gives the average processor usage cost for each transaction.

Table 35 Processor usage for each transaction average

Average ms of CPU per Transaction	0.189
-----------------------------------	-------

Over VTAM LU6.2

Table 36 summarizes the performance data for this scenario.

Table 36 DPL over VTAM LU6.2 for raw data

ETR	RESP (ms)	CICS GCPU%	TCP/IP GCPU%	TOTAL ms CPU per transaction
499.69	5	8.28	2.53	0.216
833.24	1	13.46	3.88	0.208
1249.17	1	19.77	5.50	0.202
2259.40	1	35.40	9.56	0.198
2750.13	1	42.81	11.45	0.197

Table 37 gives the average processor usage cost per transaction.

Table 37 Processor usage per transaction average

Average ms of CPU per transaction	0.204
-----------------------------------	-------

Conclusions

In this benchmark, the IPIC configuration shows a modest reduction in processor usage for each transaction when compared to VTAM LU6.2. In other benchmarks, this reduction in processor usage is greater for IPIC depending on the number of requests for each transaction and the length of the data transmitted.

The better response times for the VTAM configuration are a reflection of the CTC connection speed compared to the OSA that was used for the IPIC rather than the effect of any protocol differences.

Summary

Figure 8 shows the distributed program link summary.

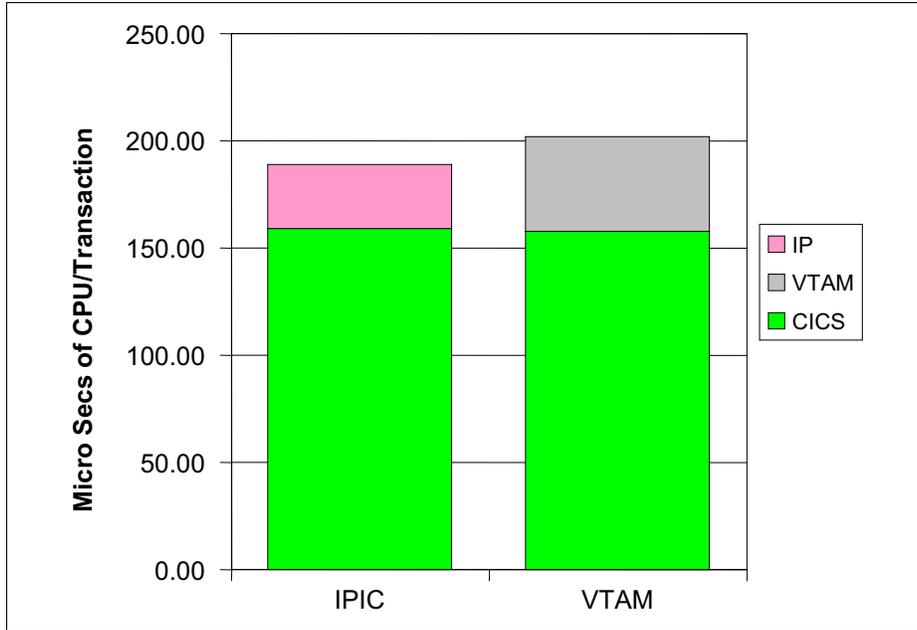


Figure 8 Distributed program link summary

Scenario: VTAM LU6.2 Distributed Transaction Processing

Example 6 consists of a CICS client application that uses the `SYNCLEVEL 2 DTP` command to send a 32 KB message into another CICS system. The server CICS system changes the data and sends 32 KB back to the client and then both the server and the client transactions terminate. The flows are as described in the next paragraph.

Three configurations were used for LU6.2:

- ▶ Where the flows were over *cross-system coupling facility* (XCF) through the coupling facility (only possible where the server and client regions are on LPARs in the same Sysplex)
- ▶ Where the flows were over IP using Enterprise Extender through the OSA
- ▶ Where the flows were over VTAM through a CTC from one LPAR to the other

Example 6 VTAM LU6.2 Distributed Transaction Processing

Client side	Server side
EXEC CICS ALLOCATE	
EXEC CICS CONNECT PROCESS	
EXEC CICS CONVERSE----->	EXEC CICS EXTRACT PROCESS
	EXEC CICS RECEIVE
	<-----EXEC SEND (wait)
EXEC CICS SYNCPOINT----->	
	EXEC CICS EXTRACT ATTRIBUTES

```

EXEC CICS RECEIVE
<-----EXEC CICS SYNCPOINT
EXEC CICS FREE ----->
EXEC CICS EXTRACT ATTRIBUTES
EXEC CICS RECEIVE
EXEC CICS FREE

```

Over cross-system coupling facility

Table 38 summarizes the performance data for this scenario.

Table 38 LU6.2 over XCF for raw data

ETR	Resp (ms)	CICS GCPU%	NET GCPU%	XCFAS GCPU%	TOTAL ms CPU per trans.
499.66	6	9.26	6.03	2.04	0.346
832.73	2	15.33	10.83	3.50	0.356
1244.52	2	22.58	16.13	5.26	0.353
2477.27	2	44.02	31.89	10.44	0.348
4691.85	3	83.33	59.38	19.31	0.345

Table 39 gives the average processor usage cost per transaction.

Table 39 Processor usage per transaction average

Average ms of CPU per transaction	0.350
--	-------

Enterprise Extender

Table 40 summarizes the performance data for this scenario.

Table 40 LU6.2 over Enterprise Extender for raw data

ETR	Resp (ms)	CICS GCPU%	NET GCPU%	TCP/IP GCPU%	TOTAL ms CPU per trans.
605.03	14	12.72	7.60	4.52	0.410
798.97	16	16.62	10.05	5.91	0.407
1170.62	18	24.80	14.74	8.36	0.409
1294.84	38	26.83	16.30	9.45	0.406
1806.80	39	35.19	22.75	12.81	0.391

Table 41 gives the average processor usage cost per transaction.

Table 41 Processor usage per transaction average

Average ms of CPU per transaction	0.404
--	-------

Over CTC

Table 42 summarizes the performance data for this scenario.

Table 42 LU6.2 VTAM over CTC for raw data

ETR	Resp (ms)	CICS GCPU%	NET GCPU%	TOTAL ms CPU per transaction
495.67	12	8.77	2.66	0.230
820.22	8	14.36	4.26	0.227
1222.77	9	21.18	6.27	0.224
2396.25	7	41.02	12.00	0.221
4449.74	9	75.25	21.55	0.217

Table 43 provides the average processor usage cost per transaction.

Table 43 Processor usage per transaction average

Average ms of CPU per Transaction	0.224
--	-------

Conclusions

The preceding measurements compared the same LU6.2 DTP application that runs in one case between two CICS regions on two different LPARs within a sysplex. In this situation, the traffic goes over the XCFAS address space and through the coupling facility. In the other case, the connection between the LPARs is configured so that the VTAM traffic flows over TCP/IP by using Enterprise Extender. In this case, the data flows over the OSA by using the same path as the other TCP/IP measurements provided in this paper. In the third case, the data flows over VTAM across a CTC.

As shown in the benchmarks, the lowest cost in terms of processor usage per transaction was achieved with the VTAM over CTC configuration. In terms of response time, the best results were achieved by using the CTC.

Summary

Figure 9 shows the LU6.2 Distributed Transaction Processing summary. The graph depicts the processor usage comparison between an LU6.2 Distributed Transaction Processing application that uses VTAM over XCF with the coupling facility. This is the same application that uses VTAM over the TCP/IP Enterprise Extender and the environment with VTAM flowing across a CTC.

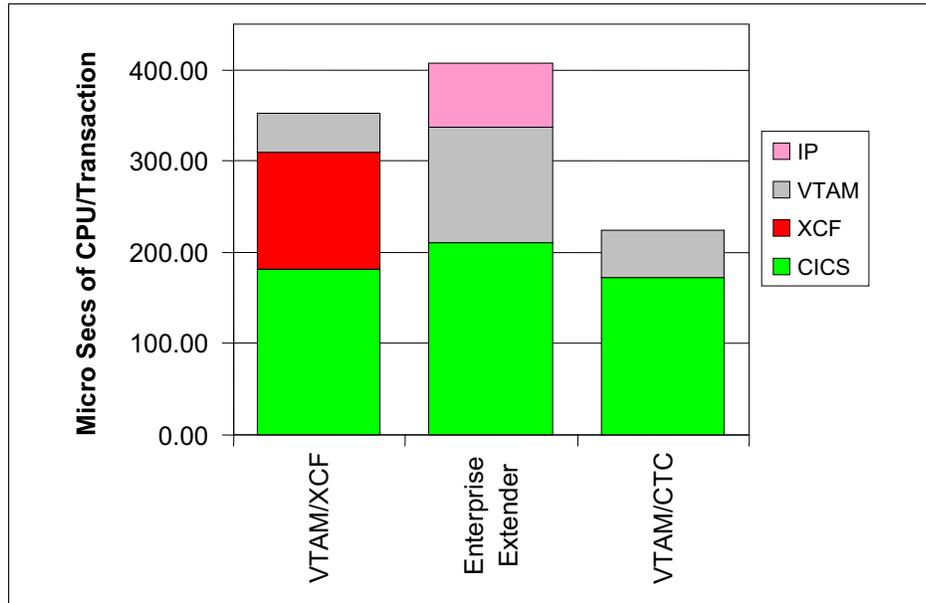


Figure 9 LU6.2 Distributed Transaction Processing summary

The team who wrote this paper

This paper was produced by a team of specialists from around the world working at the IBM International Technical Support Organization (ITSO), Rochester Center.

John Burgess is a senior software engineer working in the IBM Hursley CICS performance team. He has over 20 years experience with CICS and specializes in performance of large systems.

Arndt Eade is a senior software engineer that works in the CICS services and technology team at IBM Hursley laboratory in the United Kingdom. He has over 25 years of IT experience, mainly on IBM zSeries®, working on products such as CICS and WebSphere MQ. He joined IBM in 2003.

Thanks to the following people for their contributions to this Redpaper publication:

Trevor Clarke
Mark Cocker
Steve Day
Rob Jones
Graham Rawson
Phil Wakelin
IBM UK Software Group, Hursley

Now you can become a published author, too

Here's an opportunity to spotlight your skills, grow your career, and become a published author—all at the same time! Join an ITSO residency project and help write a book in your area of expertise, while honing your experience using leading-edge technologies. Your efforts will help to increase product acceptance and customer satisfaction, as you expand your network of technical contacts and relationships. Residencies run from two to six weeks in length, and you can participate either in person or as a remote resident working from your home base.

Find out more about the residency program, browse the residency index, and apply online at:

ibm.com/redbooks/residencies.html

Stay connected to IBM Redbooks

- ▶ Find us on Facebook:
<http://www.facebook.com/IBMRedbooks>
- ▶ Follow us on Twitter:
<http://twitter.com/ibmredbooks>
- ▶ Look for us on LinkedIn:
<http://www.linkedin.com/groups?home=&gid=2130806>
- ▶ Explore new IBM Redbooks® publications, residencies, and workshops with the IBM Redbooks weekly newsletter:
<https://www.redbooks.ibm.com/Redbooks.nsf/subscribe?OpenForm>
- ▶ Stay current on recent Redbooks publications with RSS Feeds:
<http://www.redbooks.ibm.com/rss.html>

Notices

This information was developed for products and services offered in the U.S.A.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. 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.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not grant you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing, IBM Corporation, North Castle Drive, Armonk, NY 10504-1785 U.S.A.

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.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM websites are provided for convenience only and do not in any manner serve as an endorsement of those websites. The materials at those websites are not part of the materials for this IBM product and use of those websites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Any performance data contained herein was determined in a controlled environment. Therefore, the results obtained in other operating environments may vary significantly. Some measurements may have been made on development-level systems and there is no guarantee that these measurements will be the same on generally available systems. Furthermore, some measurements may have been estimated through extrapolation. Actual results may vary. Users of this document should verify the applicable data for their specific environment.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs.

© Copyright International Business Machines Corporation 2013. All rights reserved.

Note to U.S. Government Users Restricted Rights -- Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

This document REDP-4906-00 was created or updated on January 25, 2013.



Send us your comments in one of the following ways:

- ▶ Use the online **Contact us** review Redbooks form found at:
ibm.com/redbooks
- ▶ Send your comments in an email to:
redbooks@us.ibm.com
- ▶ Mail your comments to:
IBM Corporation, International Technical Support Organization
Dept. HYTD Mail Station P099
2455 South Road
Poughkeepsie, NY 12601-5400 U.S.A.



Trademarks

IBM, the IBM logo, and ibm.com are trademarks or registered trademarks of International Business Machines Corporation in the United States, other countries, or both. These and other IBM trademarked terms are marked on their first occurrence in this information with the appropriate symbol (® or ™), indicating US registered or common law trademarks owned by IBM at the time this information was published. Such trademarks may also be registered or common law trademarks in other countries. A current list of IBM trademarks is available on the Web at <http://www.ibm.com/legal/copytrade.shtml>

The following terms are trademarks of the International Business Machines Corporation in the United States, other countries, or both:

CICS®	Redpaper™	VTAM®
HiperSockets™	Redbooks (logo)  ®	WebSphere®
IBM®	Resource Measurement Facility™	z/OS®
MVS™	RMF™	zEnterprise®
Redbooks®	System z®	zSeries®

The following terms are trademarks of other companies:

Java, and all Java-based trademarks and logos are trademarks or registered trademarks of Oracle and/or its affiliates.

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