

IBM CICS Performance Series: CICS TS for z/OS V5 Performance Report

Ian Burnett

Graham Rawson

Mike Brooks

Manuela Mandelli



z Systems



International Technical Support Organization

**IBM CICS Performance Series: CICS TS for z/OS V5
Performance Report**

August 2019

Note: Before using this information and the product it supports, read the information in “Notices” on page xi.

Third Edition (August 2019)

This edition has been updated to reflect enhancements in Version 5, Release 5 of IBM CICS Transaction Server for z/OS. The overall publication is applicable to Version 5, Release 1 of IBM CICS Transaction Server for z/OS and later (product number 5655-Y04).

© Copyright International Business Machines Corporation 2016, 2019. All rights reserved.

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

Contents

Notices	xi
Trademarks	xii
.....	xiii
Preface	xv
Authors	xvi
Now you can become a published author, too!	xvi
Comments welcome	xvii
Stay connected to IBM Redbooks	xvii
Summary of changes	xix
August 2019, Third Edition	xix
March 2018, Second Edition	xix
Part 1. CICS TS for z/OS performance concepts	1
Chapter 1. Performance terminology	3
1.1 CPU Measurement Facility	4
1.2 Relative nest intensity	4
1.2.1 Memory hierarchy and nest	5
1.2.2 Factors that can influence RNI	5
1.3 Large Systems Performance Reference	6
1.3.1 External throughput rate	7
1.3.2 Internal throughput rate	7
1.3.3 ITR and ETR relationship	8
1.3.4 LSPR ITR ratios	8
1.4 Relating LSPR values to a CICS workload	9
1.4.1 LSPR alternative	10
Chapter 2. Test methodology	11
2.1 Workloads	12
2.2 Workload design	12
2.3 Repeatable measurements	13
2.3.1 Repeatability for Java workloads	14
2.4 Driving the workload	16
2.5 Summary of performance monitoring tools	16
2.5.1 RMF Monitor I	16
2.5.2 RMF Monitor III	16
2.5.3 CICS TS statistics	16
2.5.4 CICS TS performance class monitoring	17
2.5.5 Hardware instrumentation counters and samples	17
2.6 Collecting performance data	18
2.6.1 Collecting Java performance data	19
Chapter 3. Workload descriptions	21
3.1 Regression testing	22
3.2 Data Systems Workload	22
3.2.1 DSW static routing	22
3.2.2 DSW dynamic routing	23

3.2.3	DSW dynamic routing by using IPIC	24
3.3	Relational Transactional Workload	25
3.4	WebSphere Liberty servlet with JDBC and JCICS access	26
3.5	Java OSGi workload	27
3.6	Web services.	28
3.6.1	Web services variations	29
3.7	File control workload	29
Chapter 4.	Open transaction environment	31
4.1	Introduction to the open transaction environment.	32
4.2	TCB modes	32
4.3	Changing TCB modes.	33
4.3.1	Java programs	33
4.3.2	Programs specifying JVM(NO) and API(CICSAPI)	34
4.3.3	Programs specifying JVM(NO) and API(OPENAPI)	35
4.3.4	Programs compiled with the XPLINK option	35
4.4	Understanding the effect of change mode operations	36
4.4.1	CICS Monitoring Facility	36
4.4.2	IBM CICS Performance Analyzer for z/OS	36
4.4.3	IBM CICS Interdependency Analyzer for z/OS	37
Part 2.	CICS TS performance information	39
Chapter 5.	CICS TS for z/OS V5.1	41
5.1	Introduction	42
5.2	Release-to-release comparisons	42
5.2.1	Data Systems Workload static routing	42
5.2.2	DSW dynamic routing	44
5.2.3	Relational Transactional Workload threadsafe	45
5.2.4	Java throughput	47
5.3	Improvements in threadsafety.	48
5.3.1	Threadsafe API and SPI commands.	48
5.3.2	Threadsafe program loading.	49
5.3.3	Use of T8 TCB for JDBC calls.	50
5.4	Changes to system initialization parameters.	52
5.4.1	Active keypoint frequency (AKPFREQ).	52
5.4.2	Extended dynamic storage area limit (EDSALIM).	52
5.4.3	Terminal scan delay (ICVTSD)	52
5.4.4	Maximum open TCBs (MAXOPENTCBS).	53
5.4.5	Maximum XP TCBs (MAXXPTCBS)	53
5.4.6	Maximum tasks (MXT)	53
5.4.7	Priority aging interval (PRTYAGE)	54
5.4.8	Location of terminal user areas (TCTUALOC)	54
5.5	Enhanced instrumentation	54
5.5.1	The DFHCHNL performance group	54
5.5.2	The DFHCICS performance group	55
5.5.3	The DFHDEST performance group.	55
5.5.4	The DFHFILE performance group.	55
5.5.5	The DFHRMI performance group	55
5.5.6	The DFH SOCK performance group	56
5.5.7	The DFHSTOR performance group	56
5.5.8	The DFHTASK performance group.	57
5.5.9	The DFHTERM performance group	59
5.5.10	Monitoring domain global statistics	59

5.5.11	Loader domain global statistics	60
5.6	Virtual storage constraint relief	60
5.6.1	24-bit storage	60
5.6.2	31-bit storage	61
5.6.3	64-bit storage	61
5.6.4	Mirror transactions	61
5.6.5	XCTL with a communication area	61
5.6.6	User exit global work area	62
5.6.7	Related APARs	62
5.7	64-bit application support	62
5.8	Java 7 and zEnterprise EC12	62
5.9	CICSplex System Manager dynamic routing	64
5.10	Workload consolidation	66
5.10.1	Consolidating a COBOL VSAM workload	66
5.10.2	Consolidating the GENAPP workload	69
5.11	Effect of threadsafe transient data	71
5.11.1	Maximizing time on an open TCB	71
5.11.2	Sample transient data and DB2 application	71
5.11.3	CICS monitoring data results	72
5.11.4	Throughput results	73
5.12	Transaction isolation	73
5.12.1	Unique subspaces	74
5.12.2	Common subspace	75
5.12.3	Base space	75
5.12.4	Transaction isolation performance effect	75
5.12.5	Page and extent sizes	76
Chapter 6.	CICS TS for z/OS V5.2	77
6.1	Introduction	78
6.2	Release-to-release comparisons	78
6.2.1	Data Systems Workload static routing	78
6.2.2	DSW dynamic routing	79
6.2.3	Relational Transactional Workload non-threadsafe	81
6.2.4	RTW threadsafe	82
6.2.5	Java servlet that uses JDBC and VSAM	83
6.3	Improvements in threadsafe API and SPI commands	85
6.4	Changes to system initialization parameters	86
6.4.1	Maximum tasks (MXT)	86
6.4.2	Maximum open TCBs (MAXOPENTCBS)	87
6.4.3	Maximum XP TCBs (MAXXPTCBS)	87
6.5	Enhanced instrumentation	87
6.5.1	Dispatcher global statistics	87
6.5.2	Dispatcher TCB mode statistics	88
6.5.3	Dispatcher TCB pool statistics	89
6.5.4	Monitoring domain global statistics	89
6.5.5	Transaction manager global statistics	91
6.6	Kerberos	91
6.6.1	Kerberos test configuration	93
6.6.2	Kerberos performance results	94
6.6.3	Kerberos support conclusion	94
6.7	JSON support	95
6.7.1	JSON support test configuration	95
6.7.2	Scalability as a function of response size	100

6.7.3 Scalability as a function of request rate	101
6.7.4 JSON support conclusion	102
6.8 Java applications and trace	102
6.9 Web services over HTTP improvements	104
6.9.1 Web services real storage usage	105
6.9.2 Web services CPU reduction	105
6.10 Java 7.0 and Java 7.1	106
Chapter 7. CICS TS for z/OS V5.3	109
7.1 Introduction	110
7.2 Release-to-release comparisons	110
7.2.1 Data Systems Workload dynamic routing	110
7.2.2 RTW threadsafe	110
7.3 Improvements in threadsafety	112
7.3.1 Threadsafe API and SPI commands	112
7.3.2 Optimizations for SSL support	113
7.3.3 Offloading authentication requests to open TCBs	114
7.4 Changes to system initialization parameters	114
7.4.1 Storage protection (STGPROT)	114
7.4.2 Internal trace table size (TRTABSZ)	114
7.5 Enhanced instrumentation	115
7.5.1 The DFHCICS performance group	115
7.5.2 The DFHTASK performance group	115
7.5.3 The DFHTEMP performance group	115
7.5.4 The DFHWEBB performance group	116
7.5.5 Monitoring domain global statistics	116
7.5.6 TCP/IP global statistics	117
7.5.7 URIMAP global statistics	118
7.6 Low-level CICS optimizations	118
7.6.1 Monitoring and trace enabled	119
7.6.2 Monitoring disabled, trace enabled	119
7.6.3 Monitoring enabled, trace disabled	120
7.6.4 Monitoring and trace disabled	121
7.6.5 Monitoring and trace disabled with low numbers of MRO sessions	121
7.6.6 Low-level CICS optimizations conclusions	122
7.7 Web support and web service optimization	122
7.8 Java workloads	123
7.8.1 Java workload configuration	123
7.8.2 Java servlet workload	123
7.8.3 Java OSGi workload	125
7.9 Java 8 performance	127
7.9.1 Improvements in Java 7.0, Java 7.1, and Java 8	127
7.9.2 Java performance benchmarks in CICS	127
7.9.3 Java 8 and OSGi applications	128
7.9.4 Java 8 and WebSphere Liberty servlet applications	128
7.9.5 Java 8 and z/OS Connect applications	129
7.10 Simultaneous multithreading with Java workloads	130
7.10.1 Introduction to SMT	130
7.10.2 Measuring SMT performance	131
7.10.3 CICS throughput improvement	132
7.11 Reporting of CPU time to z/OS Workload Manager	133
7.12 z/OS Connect for CICS	134
7.12.1 CICS TS V5.3 performance enhancement	134

7.12.2	Varying payload sizes by using Java parser	135
7.12.3	Comparing Java and native parsers	136
7.12.4	Comparing Java and native parsers with varying request sizes	136
7.12.5	Comparing Java and native parsers with varying response sizes	138
7.12.6	Native parser conclusion.	140
7.13	HTTP flow control	140
7.13.1	Server accept efficiency fraction	140
7.13.2	Flow control configuration	141
7.13.3	Flow control operation.	142
7.13.4	CICS statistics enhancements	142
7.13.5	Comparison of SOTUNING options	142
7.14	High transaction rate performance study	143
7.15	WebSphere Liberty zIIP eligibility	143
7.16	Link to WebSphere Liberty	144
7.16.1	Performance comparison	145
7.16.2	Link to WebSphere Liberty performance results	145
7.16.3	Link to WebSphere Liberty performance conclusion.	146
7.16.4	Comparison of CICS monitoring and RMF data	147
Chapter 8.	CICS TS for z/OS V5.4	149
8.1	Introduction	150
8.2	Release-to-release comparisons	150
8.2.1	Data Systems Workload static routing	150
8.2.2	Data Systems Workload dynamic routing	151
8.2.3	The Data Systems Workload HTTP interface	153
8.2.4	The Relational Transactional Workload threadsafe	154
8.2.5	The Java servlet that uses JDBC and VSAM	156
8.2.6	The Java OSGi workload	159
8.3	Improvements in threadsafety.	161
8.3.1	Threadsafe API commands	161
8.3.2	Use of T8 TCB for MQ calls from Java	162
8.4	Changes to system initialization parameters.	162
8.4.1	The EDSA limit (EDSALIM) parameter.	162
8.4.2	Default runaway task time (ICVR).	162
8.4.3	Maximum open TCBs (MAXOPENTCBS).	163
8.4.4	Maximum SSL TCBs (MAXSSLTCBS).	163
8.4.5	RACF synchronization (RACFSYNC).	163
8.4.6	Sign on for a preset user ID (SNPRESET).	164
8.4.7	z/OS Workload Manager Health API (WLMHEALTH).	164
8.5	Changes to resource definition attribute default values	164
8.5.1	The PROGRAM resource	165
8.5.2	The TRANSACTION resource	165
8.6	Enhanced instrumentation	165
8.6.1	The DFHCICS performance group	166
8.6.2	The DFHPROG performance group	166
8.6.3	The DFHTASK performance group.	167
8.6.4	Transaction resource class data	167
8.6.5	Identity class data	167
8.6.6	Asynchronous services global statistics	168
8.6.7	IBM MQ monitor statistics.	168
8.6.8	TCP/IP global statistics.	170
8.6.9	TCP/IP services resource statistics.	171
8.6.10	z/OS Communications Server global statistics	171

8.7	CICS tasks for WebSphere Liberty applications	172
8.8	z/OS WLM Health API	173
8.9	Asynchronous API	173
8.9.1	Enhanced transaction tracking information	174
8.9.2	Workload management	175
8.9.3	CICS monitoring enhancements	176
8.9.4	CICS statistics enhancements	178
8.9.5	Asynchronous service domain performance comparison	179
8.9.6	Asynchronous API performance summary	181
8.10	EXCI support for channels and containers	182
8.10.1	Application design (COMMAREA)	182
8.10.2	Application design (channels)	183
8.10.3	Payload comparisons	184
8.10.4	CPU time comparison	184
8.10.5	Response time comparison	186
8.10.6	EXCI support for channels and containers performance summary	188
8.11	CICS support for IBM Health Checker for z/OS	188
8.12	Web services performance	189
8.12.1	Persistent and non-persistent connections	190
8.12.2	Overhead of using SSL with persistent connections	191
8.12.3	SSL handshake overhead	192
8.12.4	CICS SSL and AT-TLS	194
Chapter 9.	CICS TS for z/OS V5.5	197
9.1	Introduction	198
9.2	Release-to-release comparisons	198
9.2.1	Data Systems Workload static routing	198
9.2.2	Java servlet that uses JDBC and VSAM	200
9.2.3	The Java OSGi workload	202
9.2.4	Relational Transactional Workload threadsafe	205
9.3	Improvements in threadsafety	206
9.3.1	QUERY SECURITY API command	206
9.3.2	Coupling Facility Data Tables	206
9.3.3	System subtasking and auxiliary temporary storage	207
9.4	Changes to system initialization parameters	207
9.4.1	High Performance Option (HPO)	207
9.4.2	Minimum TLS level (MINTLSLEVEL)	207
9.5	Enhanced instrumentation	208
9.5.1	The DFH SOCK performance group	208
9.5.2	The DFH WEBB performance group	208
9.5.3	The DFH WEBC performance group	208
9.5.4	ISC/IRC system entry resource statistics	209
9.5.5	Policy statistics	209
9.5.6	Transaction resource statistics	210
9.5.7	Transaction resource class data	210
9.6	Virtual storage constraint relief	210
9.7	z/OS WLM Health API	211
9.7.1	CICSplex SM workload routing	211
9.7.2	Throttle on number of MQGET calls issued by an MQMONITOR	211
9.8	Disabling of VSAM dynamic buffer addition	212
9.9	USS processes associated with L8, L9, X8, and X9 TCBs	212
9.10	Channels performance improvement	212
9.10.1	Containers performance comparison	213

9.10.2 Containers performance summary	216
9.11 Threadsafe Coupling Facility Data Tables	217
9.11.1 CFDT performance test configuration	217
9.11.2 Non-threadsafe CFDT application performance results	217
9.11.3 Threadsafe CFDT application performance results	218
9.11.4 CFDT performance results summary	220
9.12 CICS policy rules	221
9.12.1 Policy task rules overhead performance study	221
9.12.2 Policy task rules overhead performance summary	224
9.13 Encrypted zFS file systems	224
9.13.1 zFS file system encryption performance comparison	224
9.13.2 zFS file system encryption performance summary	225
9.14 Multiple Liberty JVM servers in a single CICS region	226
9.14.1 Shared libraries support	226
9.14.2 Multiple Liberty JVM servers performance workload configuration	226
9.14.3 Comparing CPU costs per request and maximum throughput	227
9.14.4 Comparing 31-bit memory usage	229
9.14.5 Comparing 64-bit memory usage	231
9.14.6 Multiple Liberty JVM servers performance conclusion	233
9.15 Liberty JVM server and application startup times	233
9.15.1 Startup times with shared class cache	234
9.15.2 Application startup times with multiple JVM servers	235
Related publications	237
CICS performance series	237
Other IBM Redbooks publications	237
Other publications	237
Online resources	238
Help from IBM	238

Notices

This information was developed for products and services offered in the US. This material might be available from IBM in other languages. However, you may be required to own a copy of the product or product version in that language in order to access it.

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, MD-NC119, Armonk, NY 10504-1785, US

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 jurisdictions 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 provide in any way it believes appropriate without incurring any obligation to you.

The performance data and client examples cited are presented for illustrative purposes only. Actual performance results may vary depending on specific configurations and operating conditions.

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.

Statements regarding IBM's future direction or intent are subject to change or withdrawal without notice, and represent goals and objectives only.

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 actual people or business enterprises 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. The sample programs are provided "AS IS", without warranty of any kind. IBM shall not be liable for any damages arising out of your use of the sample programs.

Trademarks

IBM, the IBM logo, and [ibm.com](http://www.ibm.com) are trademarks or registered trademarks of International Business Machines Corporation, registered in many jurisdictions worldwide. Other product and service names might be trademarks of IBM or other companies. A current list of IBM trademarks is available on the web at “Copyright and trademark information” at <http://www.ibm.com/legal/copytrade.shtml>

The following terms are trademarks or registered trademarks of International Business Machines Corporation, and might also be trademarks or registered trademarks in other countries.

CICS®	MVS™	System z9®
CICSplex®	Parallel Sysplex®	WebSphere®
DataPower®	RACF®	z Systems®
DB2®	Rational®	z/Architecture®
Db2®	Redbooks®	z/OS®
IA®	Redbooks (logo)  ®	z10™
IBM®	Resource Link®	z13®
IBM z Systems®	Resource Measurement Facility™	z9®
IBM z13®	RMF™	zEnterprise®
IBM z14™	System Storage®	
Language Environment®	System z®	

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.

UNIX is a registered trademark of The Open Group in the United States and other countries.

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

Preface

This IBM® Redbooks® publication gives a broad understanding of several important concepts that are used when describing IBM CICS® Transaction Server (TS) for IBM z/OS® (CICS TS) performance. This publication also describes many of the significant performance improvements that can be realized by upgrading your environment to the most recent release of CICS TS.

This book targets the following audience:

- ▶ Systems Architects wanting to understand the performance characteristics and capabilities of a specific CICS TS release.
- ▶ Capacity Planners and Performance Analysts wanting to understand how an upgrade to the latest release of CICS TS affects their environment.
- ▶ Application Developers wanting to design and code highly optimized applications for deployment into a CICS TS environment.

This book covers the following topics:

- ▶ A description of the factors that are involved in the interaction between IBM z Systems® hardware and a z/OS software environment.
- ▶ A definition of key terminology that is used when describing the results of CICS TS performance benchmarks.
- ▶ A presentation of how to collect the required data (and the methodology used) when applying Large Scale Performance Reference (LSPR) capacity information to a CICS workload in your environment.
- ▶ An outline of the techniques that are applied by the CICS TS performance team to achieve consistent and accurate performance benchmark results.
- ▶ High-level descriptions of several key workloads that are used to determine the performance characteristics of a CICS TS release.
- ▶ An introduction to the open transaction environment and task control block (TCB) management logic in CICS TS, including a reference that describes how several configuration attributes combine to affect the behavior of the CICS TS dispatcher.
- ▶ Detailed information that relates to changes in performance characteristics between successive CICS TS releases, covering comparisons that relate to CICS TS V4.2, V5.1, V5.2, V5.3, V5.4, and V5.5.
- ▶ The results of several small performance studies to determine the cost of using a specific CICS functional area.

Authors

This book is one in a series that is focused on CICS performance. It is written by members of the IBM Hursley CICS development community. The subject matter in this series is based on feedback from CICS customers.

Ian Burnett is the CICS Transaction Server performance team lead working for IBM United Kingdom. He has worked in IBM since 2001 as part of the development teams for several products, including IBM WebSphere® Application Server, IBM MQ, and IBM CICSplex® System Manager. Since 2009, Ian has led the performance team in Hursley, with experience in a wide range of CICS performance topics. He holds a Bachelor's degree in Computer Science from the University of Warwick in the UK.

Graham Rawson is a CICS Performance Analyst within the CICS Development group based at the IBM Laboratory in Hursley, England. Graham has worked for IBM for over 25 years in various roles supporting CICS Transaction Server and IBM Java technology. Graham holds a B.Sc. (Hons) in Chemistry from the University of East Anglia (Norwich, England) and a Certificate in Software Engineering from the University of Oxford (England).

Mike Brooks is currently a CICS Performance Analyst based at Hursley with more than 30 years experience working with CICS. He has extensive knowledge of the communication mechanisms this product uses and also has recently specialized in JSON message processing. He holds a M.Sc. in Information Systems and is an IBM Master Inventor.

Manuela Mandelli is currently a CICS Performance Analyst based at Hursley with more than 30 years experience supporting and working with CICS. She studied in Liceo Scientifico in Milano. Her areas of expertise include CICS Transaction Server, CICS Transaction Gateway, and IBM Session Manager.

The project that produced this publication was initially managed by **Marcela Adan** and then by **Martin Keen**, IBM Redbooks Project Leaders, International Technical Support Organization.

The following people contributed to previous versions of this publication:

- ▶ John Burgess

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

Comments welcome

Your comments are important to us!

We want our books to be as helpful as possible. Send us your comments about this book or other IBM Redbooks publications 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

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 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>

Summary of changes

This section describes the technical changes made in this edition of the book and in previous editions. This edition might also include minor corrections and editorial changes that are not identified.

Summary of Changes for SG24-8298-02 for IBM CICS Performance Series: CICS TS for z/OS V5 Performance Report as created or updated on August 8, 2019.

August 2019, Third Edition

This edition was updated to reflect enhancements in Version 5, Release 5 of IBM CICS Transaction Server for z/OS (CICS TS) and includes the following new and changed information.

New information

- ▶ 3.7, “File control workload” on page 29
- ▶ Chapter 9, “CICS TS for z/OS V5.5” on page 197

Changed information

- ▶ Corrections to configuration parameters in 8.2.5, “The Java servlet that uses JDBC and VSAM” on page 156 and 8.2.6, “The Java OSGi workload” on page 159.
- ▶ Updated information in 8.8, “z/OS WLM Health API” on page 173 to include new function available via APAR.

March 2018, Second Edition

This edition was updated to reflect enhancements in Version 5, Release 4 of IBM CICS Transaction Server for z/OS (CICS TS) and includes the following new and changed information.

New information

- ▶ 2.3.1, “Repeatability for Java workloads” on page 14
- ▶ 2.6.1, “Collecting Java performance data” on page 19
- ▶ 3.5, “Java OSGi workload” on page 27
- ▶ 3.6, “Web services” on page 28
- ▶ 7.15, “WebSphere Liberty zIIP eligibility” on page 143
- ▶ 7.16, “Link to WebSphere Liberty” on page 144
- ▶ Chapter 8, “CICS TS for z/OS V5.4” on page 149

Changed information

Chapters 5, 6, and 7 are updated to reflect changes in IBM CICS TS V5.4 parameter limits and defaults.



Part 1

CICS TS for z/OS performance concepts

This part describes the following important topics that are related to performance in CICS TS for z/OS:

- ▶ Definition of important terms that are used when describing performance benchmarks and their results.
- ▶ An overview of the methods and tools that are used when testing the performance of the CICS product.
- ▶ A description of several key benchmarks that are used when assessing the performance characteristics of a CICS release.
- ▶ An introduction to the open transaction environment within CICS, including reference tables for use when developing optimized CICS applications.

This part includes the following chapters:

- ▶ Chapter 1, “Performance terminology” on page 3
- ▶ Chapter 2, “Test methodology” on page 11
- ▶ Chapter 3, “Workload descriptions” on page 21
- ▶ Chapter 4, “Open transaction environment” on page 31



Performance terminology

This chapter describes several important concepts and terms that are used to help you understand the performance of IBM z Systems hardware and software. These concepts and terms are referenced extensively in this publication when the performance of a CICS environment is described.

The IBM z Systems CPU Measurement Facility (CPU MF) provides information for use with the Large Scale Performance Reference (LSPR) charts. This chapter describes how CPU MF data is collected, how the LSPR charts are used, and how the figures that are obtained from the LSPR reference tables relate to CICS transaction cost and throughput.

This chapter includes the following topics:

- ▶ 1.1, “CPU Measurement Facility” on page 4
- ▶ 1.2, “Relative nest intensity” on page 4
- ▶ 1.3, “Large Systems Performance Reference” on page 6
- ▶ 1.4, “Relating LSPR values to a CICS workload” on page 9

1.1 CPU Measurement Facility

This section provides background information about the CPU MF capability that is available in IBM z Systems z10™ (and later) hardware. Data from CPU MF is used alongside the LSPR tables as described in 1.3, “Large Systems Performance Reference” on page 6.

The CPU MF capability provides optional hardware-assisted collections of information about the logical CPUs work that is run over a specified interval in selected logical partitions (LPAR).

This powerful new capability was not available previously. However, CPU MF does not replace functions or capabilities; instead, it enriches the capabilities. CPU MF consists of the following important, but independent, functions:

- ▶ The collection of counters that maintain counts of certain activities.
- ▶ The collection of samples that provide information about what the CPU is doing at the time of the sample.

The collection of counters function is intended to be run constantly to collect long-term performance data, in a similar manner to how you collect other performance data.

The collection of samples function is a short duration, precise function that identifies where CPU resources are being used to help you improve application efficiency.

CPU MF runs at the LPAR level so that you can collect counter data in one LPAR, counter and sample data in another LPAR, and not use CPU MF at all on a third LPAR. The information that CPU MF gathers pertains to only the LPARs where you enable and start CPU MF.

The implementation of CPU MF is nondisruptive. If the prerequisite hardware and software are in place, you can start CPU MF data collection with no LPAR deactivations or activations. As a result, performing an initial program load (IPL) on the system that CPU MF is used with is not necessary.

CPU MF can run in multiple LPARs simultaneously and can be used with central processors (CPs), IBM z Systems Integrated Information Processor (zIIP), and IBM z Systems Application Assist Processor (zAAP).

For more information about the concepts, configuration, and use of CPU MF data, see *Setting Up and Using the IBM System z CPU Measurement Facility with z/OS*, REDP-4727, which is available at this website:

<http://www.redbooks.ibm.com/abstracts/redp4727.html>

1.2 Relative nest intensity

This section outlines several concepts that apply to IBM z Systems memory hierarchy and then defines the relative nest intensity (RNI) metric that quantifies the interactions between software and hardware.

Included in this book are extracts from *Large Systems Performance Reference*, SC28-1187. For more information about the IBM z Systems memory hierarchy and the LSPR workloads that were used, refer to the following LSPR for IBM z Systems resource link:

<https://www.ibm.com/servers/resourceLink/lib03060.nsf/pages/lsprindex>

1.2.1 Memory hierarchy and nest

The memory hierarchy of a processor generally refers to the caches, data buses, and memory arrays that stage the instructions and data that must be executed on the micro-processor to complete a transaction or job. Many design alternatives affect this component, such as cache size, latencies (sensitive to distance from the micro-processor), number of levels, modified, exclusive, shared, invalid (MESI) protocol, controllers, switches, and number and bandwidth of data use.

Some of the caches are *private* to the micro-processor, which means that only micro-processor can access them. Other caches are shared by multiple micro-processors. In this book, the term *memory nest* for a z Systems processor refers to the shared caches and memory along with the data buses that interconnect them.

Workload performance is sensitive to how deep into the memory hierarchy the processor must go to retrieve the instructions and data of the workload for execution. Best performance occurs when the instructions and data are found in the cache (or caches) that are nearest the processor so that little time is spent waiting before execution. Where instructions and data must be retrieved from farther out in the hierarchy, the processor spends more time waiting for their arrival.

As workloads are moved between processors with different memory hierarchy designs, performance varies because the average time to retrieve instructions and data from within the memory hierarchy varies. Also, when on a processor this component continues to vary significantly because the location of the instructions and data of a workload within the memory hierarchy is affected by many factors, including locality of reference, I/O rate, competition from other applications, and other LPARs.

The most performance-sensitive area of the memory hierarchy is the activity to the memory nest, namely, the distribution of activity to the shared caches and memory. The term, *relative nest intensity* (RNI) indicates the level of activity to this part of the memory hierarchy. By using data from CPU MF, the RNI of the workload running in an LPAR can be calculated. The higher the RNI, the deeper into the memory hierarchy the processor must go to retrieve the instructions and data for that workload.

Micro-processors do not execute instructions at a constant rate. When instructions and data must be retrieved from farther out into the memory hierarchy, the processor spends more time waiting for their arrival. Therefore, a high RNI infers that the instruction execution rate (usually measured as millions of instructions per second, or MIPS) of a processor is lower than that of a workload with a low RNI. Alternatively, a workload with a high RNI requires a higher number of cycles to complete each instruction (stated as cycles per instruction, or CPI).

1.2.2 Factors that can influence RNI

Many factors influence the performance of a workload. However, usually what these factors are influencing is the RNI of the workload. The interaction of all these factors is what results in a net RNI for the workload, which in turn directly relates to the performance of the workload.

Figure 1-1 shows the traditional factors that were used to categorize workloads in the past, along with their RNI tendency.

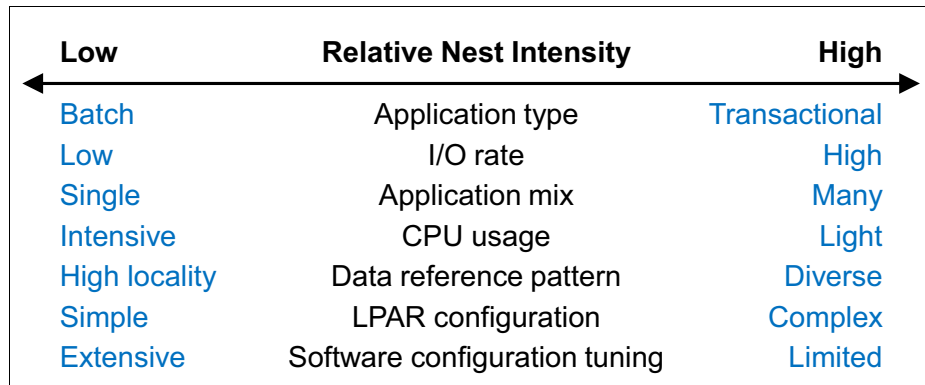


Figure 1-1 Relative nest intensity tendencies

An important aspect to emphasize is that these factors are tendencies and not absolutes. For example, a workload might have a low I/O rate, intensive CPU use, and a high locality of reference, which are all factors that suggest a low RNI. But, what if it is competing with many other applications within the same LPAR and many other LPARs on the processor, which tend to push it toward a higher RNI? The net effect of the interaction of all these factors is what determines the RNI of the workload, which in turn greatly influences its performance.

You can do little to affect most of these factors. An application type is whatever is necessary to do the job. Data reference pattern and CPU usage tend to be inherent in the nature of the application. LPAR configuration and application mix are mostly a function of what must be supported on a system. I/O rate can be influenced somewhat through buffer pool tuning.

However, one factor that can be affected (software configuration tuning) is often overlooked but can have a direct effect on RNI. In the context of a CICS workload, software configuration tuning refers to the number of address spaces, such as CICS application-owning regions (AORs), that are needed to support a workload. This factor always existed but its sensitivity is higher with today's high frequency micro-processors. Spreading the same workload over many address spaces than necessary can raise the RNI of a workload as the working set of instructions and data from each address space increases the competition for the processor caches. For more information, see 5.10, "Workload consolidation" on page 66.

Tuning to reduce the number of simultaneously active address spaces to the proper number needed to support a workload can reduce RNI and improve performance. To produce the LSPR reference tables, IBM tunes the number of address spaces for each processor type and count configuration to be consistent with what is needed to support the workload. Therefore, the LSPR workload capacity ratios reflect a presumed level of software configuration tuning. This sensitivity of RNI to the number of supporting address spaces suggests that retuning the software configuration of a production workload as it moves to a bigger or faster processor might be needed to achieve the published LSPR ratios.

1.3 Large Systems Performance Reference

The following important capacity metrics are defined in this section before the use of the LSPR tables is described:

- ▶ External throughput rate
- ▶ Internal throughput rate

1.3.1 External throughput rate

The external throughput rate (ETR) is computed by using the following equation:

$$\text{ETR} = \frac{\text{units of work}}{\text{elapsed time}}$$

For a CICS workload, *units of work* are normally expressed as the number of CICS transactions completed. To be useful, the units of work that are measured must represent a large and repeatable sample of the total workload to best represent the average. Elapsed time is normally expressed in seconds.

ETR characterizes system capacity because it is an elapsed time measurement (system capacity encompasses the performance of the processor and all of its external resources, considered together). As such, ETR lends itself to the *system comparison methodology*. This methodology requires the data processing system to be configured with all intended resources, including the processor, with appropriate amounts of central storage, expanded storage, channels, control units, I/O devices, TP network, and so on.

After the system is configured, the goal is to determine how much work the system, as a whole, can process over time. To accomplish this goal, the system is loaded with the appropriate workload until it cannot absorb work at any greater rate. The highest ETR achieved is the processing capability of the system.

When you make a system measurement of this type, all resources on the system are potential capacity inhibitors. If a resource other than the processor is, in fact, a capacity inhibitor, the processor is likely to be running at something less than optimal utilization.

This system comparison methodology is a legitimate way to measure when the intent is to assess the capacity of the system as a whole. For online systems, response time also becomes an important system-related metric because poor response times inhibit the ability of users to work. Therefore, system measurements for online work usually involve some type of response time criteria. If the response time criteria is not met, what ETR can be realized does not matter.

1.3.2 Internal throughput rate

The internal throughput rate (ITR) is computed by using the following formula:

$$\text{ITR} = \frac{\text{units of work}}{\text{processor busy}}$$

As with ETR, *units of work* are normally expressed as jobs (or job-steps) for batch workloads, and as transactions or commands for online workloads. System control programs (SCPs) and most major software products have facilities to provide this information. To be useful, the units of work that are measured must represent a large and repeatable sample of the total workload to best represent the average. Processor busy time is normally expressed in seconds.

ITR characterizes processor capacity because it is a CPU busytime measurement. As such, ITR lends itself to the *processor comparison methodology*. Because the focus of LSPR is on a single resource (the processor), you must modify the measurement approach from that used for a system comparison methodology.

To ensure that the processor is the primary point of focus, you must configure it with all necessary external resources (including central storage, expanded storage, channels, control units, and I/O devices) in adequate quantities so that they do not become constraints. You must avoid the use of processor cycles to manage external resource constraints to assure consistent and comparable measurement data across the spectrum of processors being tested.

Many acceptance criteria for LSPR measurements can help assure that external resources are adequate. For example, internal response times should be subsecond; if they are not, some type of resource constraint must be resolved. For various DASD types, expected nominal service times are known. If the measured service times are high, some type of queuing is occurring, which indicates a constrained resource. When unexpected resource constraints are detected, they are fixed and the measurement is redone.

Because the processor is also a resource that must be managed by the SCP, steps must be taken to ensure that excess queuing on it does not occur. The way to avoid this type of constraint is to make the measurements at preselected utilization levels that are less than 100%. Because the LSPR is designed to relate processor capacity, measurements must be made at reasonably high utilization, but without causing uncontrolled levels of processor queuing. Typically, LSPR measurements for online workloads are made at a utilization level of approximately 90%. Batch workloads are always measured with steady-state utilizations above 90%. Mixed workloads that contain an online and batch component are measured at utilizations near 99%.

One other point must be made about processor utilization. Whenever two processors are to be compared for capacity purposes, they should both be viewed at the same loading point, which means at equal utilization. Assessing relative capacity when one processor is running at low utilization and the other is running at high utilization is imprecise. The LSPR methodology mandates that processor comparisons be made at equivalent utilization levels.

1.3.3 ITR and ETR relationship

An ITR can be viewed as a special case ETR; that is, an ITR is the measured ETR normalized to full processor utilization. Therefore, an alternative way to compute an ITR is to use the following equation:

$$\text{ITR} = \frac{\text{ETR}}{\text{processor utilization}}$$

1.3.4 LSPR ITR ratios

LSPR capacity data is presented in the form of ITR ratios for IBM processors where each model is configured with multiple z/OS images that are based on an average LPAR profile of client systems. All capacity numbers are relative to the IBM 2094-701 running multi-image z/OS image.

Comparing ITR ratios for two processor configurations allows a capacity planner to predict the effects of modifying hardware configuration at a high level. However, the most accurate sizings require the use of the LPAR Configuration Capacity Planning function of the zPCR tool, which can be customized to match a specific multi-image configuration rather than the average configurations that are reflected in the multi-image LSPR table.

1.4 Relating LSPR values to a CICS workload

By using data that is obtained from CPU MF and the reference information that is found in LSPR, you can understand how a CICS workload is expected to perform when moving between hardware configurations.

The example in this section outlines the steps to help you understand the effects of a hardware upgrade. For this example, assume that the workload has an “average” RNI as determined by the CPU MF.

This example examines the expected effects of adding CPs to an IBM z Systems z13[®]. Table 1-1 lists an extract of ITR ratios for the two processor configurations. This extract was taken from the z/OS V2.1 LSPR ITR ratios reference.

Table 1-1 Extract of LSPR table for selected processors

Processor	# CP	Low	Average	High
2964-703	3	9.08	8.30	7.28
2964-705	5	14.72	13.21	11.45

To calculate the potential throughput improvements that are obtained by upgrading the configuration from 3 CPs to 5 CPs, calculate the ratios of the relevant ITR columns. So, the average throughput scaling is equal to $13.21 \div 8.30 = 1.59$.

Therefore, in the absence of software constraints, you might expect the throughput of the system to increase by 59%.

To calculate the change in CPU cost per transaction, first calculate the CPU cost of each LSPR transaction, as shown in the following formula:

$$\text{CPU cost} = \frac{\text{number of CPs}}{\text{ITR}}$$

Therefore, the LSPR *Average RNI* transaction for the 2964-703 processor costs 0.361s of the CPU, as shown in the following formula:

$$\frac{3}{8.30} = 0.361\text{s}$$

The same transaction on the 2964-705 processor costs 0.379s of the CPU, as shown in the following formula:

$$\frac{5}{13.21} = 0.379\text{s}$$

These values show that the CPU cost per transaction increases from 0.361s to 0.379s, which is an increase of 5%.

This increase in CPU per transaction is an expected result because increasing concurrency through the addition of CPUs increases contention for common cache lines. As described in 1.2.2, “Factors that can influence RNI” on page 5, workload performance is sensitive to how deep into the memory hierarchy the processor must go to retrieve instructions and data.

Increasing concurrency decreases the probability of a cache line to be available for exclusive use by a processor at any specific time.

Note: This increase in CPU per transaction is important for non-threadsafe CICS transactions. Non-threadsafe applications run on the CICS QR TCB; therefore, non-threadsafe applications in CICS are limited by the capacity of the single QR TCB within a CICS region.

1.4.1 LSPR alternative

The LSPR shows relative capacity ratios that are sensitive to workload type. However, LPAR configuration is also a sensitive factor in capacity relationships. IBM offers the Processor Capacity Reference (zPCR) tool for customer use. The tool takes the LSPR to the next level by estimating capacity relationships that are sensitive to workload type and LPAR configuration, processor configuration, and specialty engine configuration. All of these factors can be customized to match your configuration. The LSPR data is contained in the tool.

For the most accurate capacity sizings, zPCR should be used. By using CPU-MF data that is collected in your environment, the zPCR tool can calculate the overall RNI value of a workload and determine the most appropriate LSPR workload to model the environment.

For more information about the zPCR tooling, see the IBM Techdoc *Getting Started with zPCR*, which is available at this website:

<http://www.ibm.com/support/techdocs/atmastr.nsf/WebIndex/PRS1381>



Test methodology

This chapter provides an insight into the approach that was taken by the CICS performance team when producing performance benchmark results. The concept of a CICS workload is defined, along with a description of how workloads are designed and coded.

Performance testing requires the combination of several techniques to provide accurate, repeatable measurements. These techniques are presented here, while demonstrating some of the tools that were used when collecting performance data.

This chapter includes the following topics:

- ▶ 2.1, “Workloads” on page 12
- ▶ 2.2, “Workload design” on page 12
- ▶ 2.3, “Repeatable measurements” on page 13
- ▶ 2.4, “Driving the workload” on page 16
- ▶ 2.5, “Summary of performance monitoring tools” on page 16
- ▶ 2.6, “Collecting performance data” on page 18

2.1 Workloads

This book uses the term *workload* extensively. The term refers to the combination of the following key components of the environment that are used when producing performance figures for a specific CICS configuration:

- ▶ **Application code**

Application code can be written in any language that is supported by the CICS environment. The number, sequence, and ordering of **EXEC CICS**, **EXEC SQL**, or **EXEC DLI** commands dictate the flow of control between the application and the IBM CICS Transaction Server (TS) for IBM z/OS (CICS TS) environment under test and is known as the *workload logic*.
- ▶ **Data that is required by the application**

The data that is required by the application can be stored in VSAM files or in an IBM DB2® database, provided by the simulated clients, or supplied by some other external system. The data that is used corresponds to the data that is exchanged between components of CICS as part of a customer's application.
- ▶ **Topology of connected address spaces**

The number of CICS regions, the methods that are used to connect these CICS regions, and the logical partition (LPAR) in which the CICS region is executed all form part of the workload.
- ▶ **Configuration of the CICS region**

There are many configuration parameters for CICS and the value for each can be modified to achieve a specific effect.
- ▶ **Simulated clients**

The number of simulated clients, their method of communication with the CICS regions under test, and the rate at which requests are sent to the CICS regions can be varied to affect the behavior of a workload.

2.2 Workload design

Performance test workloads that are developed by the CICS TS performance team are deliberately *lightweight*; that is, workloads have little business logic. The phrase *business logic* refers to language constructs that serve only to manipulate data according to business rules, rather than the workload logic that is used to control program flow between the application and the CICS TS environment.

The CICS TS performance team specifically target the discovery of performance problems in the CICS TS runtime code, and having lightweight applications maximizes the visibility of any potential problems at the time of development.

The use of a transaction from the data system workload (DSW) as described in 3.2, “Data Systems Workload” on page 22, helps you understand why minimizing business logic is important. Consider the following coding scenarios for the application:

- ▶ A minimal business logic case with a total transaction CPU cost of 0.337 ms and consisting of the following values:
 - 0.322 ms of CPU for calls into CICS
 - 0.015 ms of CPU for business logic

- ▶ A more heavyweight business logic case with a total transaction CPU cost of 1.500 ms and consisting of the following values:
 - 0.322 ms of CPU for calls into CICS
 - 1.178 ms of CPU for business logic

In both cases, the amount of CPU consumed by the CICS TS code to complete the CICS operations that are required for the workload is equal to 0.322 ms.

Now consider the example where a change in the CICS TS product during the development phase inadvertently introduces a CPU overhead of 5 μ s for each transaction. With the workload in the first scenario (which contains a minimal amount of business logic), the total transaction cost increased from 0.337 ms to 0.342 ms of CPU, an increase of 1.5%. With the workload in the second scenario (which contains significant business logic), the total transaction cost increased from 1.500 ms to 1.505 ms of CPU, an increase of 0.3%.

Although techniques that are used to minimize variability in performance test results are described in 2.3, “Repeatable measurements” on page 13 and 2.6, “Collecting performance data” on page 18, you should note that only a finite level of accuracy in terms of performance test results are achievable. By following leading practices in the CICS TS performance test environment, experience indicates that an accuracy of approximately $\pm 1\%$ can be achieved. The use of coding in the first scenario resulted in a relative performance change (1.5%), which is greater than the measurement accuracy. The small performance degradation was detected and the defect can be corrected.

Minimizing the amount of business logic in the test application maximizes the relative change in performance for the whole workload for any specific modification to the CICS TS runtime code. By using this *worst-case* test scenario approach, the performance test team can be confident that real-world applications do not observe any change in performance behavior.

Observation: For the DSW, an IBM zEnterprise® EC12 model HA1 executes at a rate of approximately 1,270 million instructions per second, per central processor (CP). An inadvertent change that added 5 μ s to the total transaction cost represents approximately 6,350 instructions.

2.3 Repeatable measurements

Before describing how performance data is collected, it is important to understand that unless you totally dedicate hardware for a benchmark, the CPU that is used can vary each time that the benchmark is run. Achieving repeatable results can be difficult. This statement is true for benchmark comparisons and also for CPU usage comparisons after a CICS upgrade.

For more information about how CPU time can be affected by other address spaces in the LPAR and other LPARs on the central processor complex (CPC), see *IBM CICS Performance Series: Effective Monitoring for CICS Performance Benchmarks*, REDP-5170, which is available at this website:

<http://www.redbooks.ibm.com/abstracts/redp5170.html>

The LPARs that support the CICS regions in all performance benchmarks that are described in this publication include dedicated CPs. Although the CPs are dedicated, the L3 and L4 caches remain shared with other CPs that are used by other LPARs. So, this situation is not perfect; it can lead to CPU variation because those caches can have their data invalidated by those CPs that are used by the other LPARs. Clearly, minimizing the magnitude of these external influences is a high priority when producing reliable performance benchmark results.

An automated measurement system is used to execute the benchmarks and collect the performance data. This automated system executes overnight during a period when no human users are permitted to access the LPAR. The use of an automation system reduces variation in results by ending unnecessary address spaces that can potentially disrupt the measurements. The use of *overnight* automation also minimizes disruption because that is the time frame during which other LPARs on the CPC are least busy.

2.3.1 Repeatability for Java workloads

Java programs consist of classes, which contain Java *bytecode* that is *platform-neutral*, meaning that it is not specific to any hardware or operating system platform. At run time, the Java virtual machine (JVM) compiles Java bytecode into IBM z/Architecture® instructions, using the just-in-time compiler (JIT) component.

Producing highly-optimized z/Architecture instructions from Java bytecode requires processor time and memory. If all Java methods were compiled to the most aggressive level of optimization on first execution, this process results in long application initialization times, along with wasting significant quantities of CPU time optimizing methods that are used only during startup.

To provide a balance between application startup times and long-term performance, the JIT compiler optimizes the bytecode using an iterative process. The JIT compiler maintains a count of the number of times each Java method is called. When the call count of a method exceeds a JIT recompilation threshold, the JIT recompiles the method to a more aggressive level of optimization and resets the method invocation count. This process is repeated until the maximum optimization level is reached. Therefore, often-used methods are compiled soon after the JVM has started, and less-used methods are compiled much later or not at all. The JIT compilation threshold helps the JVM start quickly and still have good long-term performance.

For more information about the operation of the JIT compiler on z/OS, see the topic “The JIT compiler” in IBM Knowledge Center at the following website:

<https://ibm.biz/BdjxNR>

This process of progressively optimizing Java methods leads to a change over time in the amount of CPU consumed by otherwise identical transactions. The first time a transaction is executed in Java, the z/Architecture instructions that are produced by the JIT compiler are at a low optimization level, which results in a relatively high CPU cost to execute the Java methods.

As more transactions are executed, the Java method invocation counts are increased. Therefore, the JIT recompiles a Java method to a more aggressive level of optimization. This greater level of optimization results in a Java method requiring less CPU to execute than before the recompilation took place. As a result, the CPU that is required to execute the transaction reduces. This process is repeated several times during the lifetime of the JVM.

Figure 2-1 illustrates this process for a complex servlet workload in the plot.

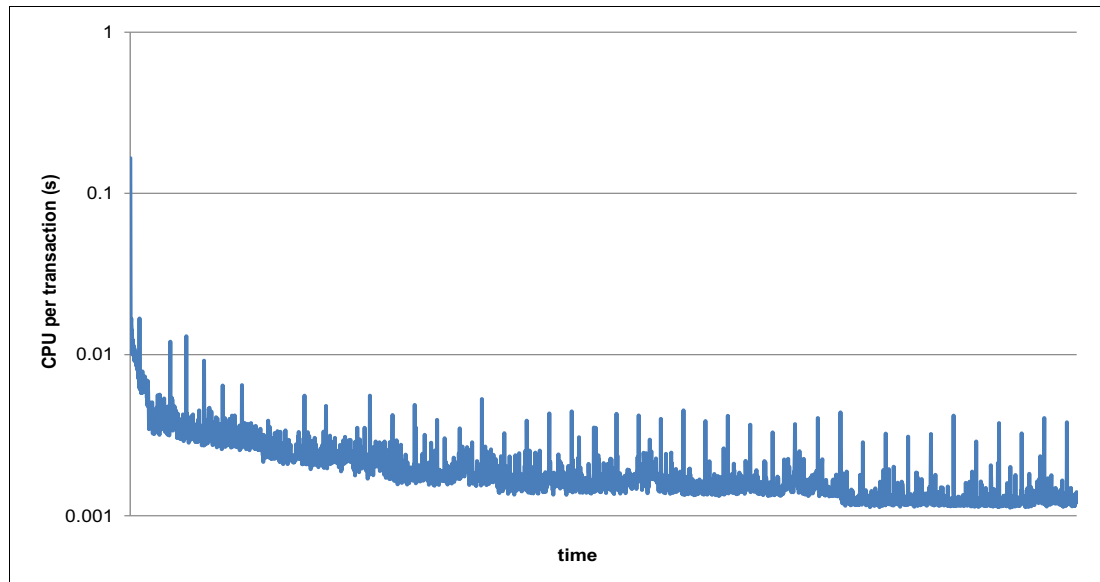


Figure 2-1 Plot of CPU cost per transaction over time for a Java workload

Noting that the vertical axis in Figure 2-1 is a logarithmic scale, the first few invocations of the transaction show a relatively high CPU usage. As the transaction is executed multiple times, the JIT compiler optimizes the workload more aggressively. Thus, the CPU cost per transaction reduces over time. Steps can be observed in the CPU cost per transaction value, which are events where high-use methods are further optimized. The frequent spikes in the CPU cost per transaction are due to garbage collection events.

When executing benchmarks that use JVMs, ensure that the JIT compiler has fully optimized the most important Java methods in the workload before starting CPU measurements. To minimize variability introduced by the JIT compiler, run the CICS Java workload at a constant transaction rate for a period of time, known as the *warm-up* time. After the workload is running in a steady-state for the warm-up period, it is assumed that the JIT compiler will not optimize the workload further, and CPU measurements can be taken.

The warm-up period for a workload is determined by producing a chart, such as the one in Figure 2-1. The warm-up time is the point at which the CPU cost per transaction ceases to show any improvements.

Shutting down a JVM discards the JIT-compiled native code; therefore, the iterative process of optimization begins again when the JVM is restarted. The ahead-of-time (AOT) compiler provides the ability to persist generated native code across subsequent executions of the same program, with the primary goal of improving startup times. The AOT compiler generates native code dynamically while an application runs and caches any generated AOT code in the shared data cache. Subsequent JVMs that execute the method can load and use the AOT code from the shared data cache without incurring the performance decrease experienced with JIT-compiled native code.

Because AOT code must persist over different program executions, AOT-generated code does not perform as well as JIT-generated code. AOT code usually performs better than interpreted code. For more information about the AOT compiler, see the topic “The AOT compiler” in IBM Knowledge Center at the following website:

<https://ibm.biz/BdjxNX>

2.4 Driving the workload

The IBM Workload Simulator for z/OS (Workload Simulator) tool is used to send work into the CICS regions from multiple simulated clients concurrently. For more information about Workload Simulator, refer to the following product web page:

<http://www.ibm.com/software/products/en/workloadsimulator>

The process of sending work into the CICS regions is commonly referred to as *driving* the workload. The system under test is on a separate LPAR in the same sysplex. All network traffic is routed by way of a coupling facility from one LPAR to the other.

2.5 Summary of performance monitoring tools

During the benchmark measurement periods, the following tools are used:

- ▶ RMF Monitor I
- ▶ RMF Monitor III
- ▶ CICS TS statistics
- ▶ CICS TS performance class monitoring
- ▶ Hardware instrumentation counters and samples

2.5.1 RMF Monitor I

IBM RMF™ Monitor I records system resource usage, including CPU, DASD, and storage. It is also used with the workload manager (WLM) configuration to record the CPU, transaction rates, and response times for CICS service classes and report classes.

SMF records 70 - 79 are written on an interval basis. They can be post-processed by using the ERBRMFPP RMF utility program.

2.5.2 RMF Monitor III

RMF Monitor III records the coupling facility activity for the logger and temporary storage structures.

SMF records 70 - 79 are written on an interval basis. Also, the records can be post-processed by using the ERBRMFPP RMF utility program. RMF Monitor III can be used on an interactive basis and the data can be written to VSAM data sets for later review.

2.5.3 CICS TS statistics

CICS statistics are used to monitor and report CICS resource usage, including CPU, storage, file accesses, and the number of requests that were transaction-routed.

With CICS interval statistics, most of the counters are reset at the start of the interval so that any resource consumption that is reported relates only to the observed measurement period. Interval statistics can be activated by using the **CEMT SET STATISTICS** command. However, when you set this interval, the first interval can be adjusted to a shorter time so that all the intervals are synchronized to the STATE0D parameter. For example, if you use CEMT to set the interval to 15 minutes at 10 past the hour, the first interval expires in 5 minutes so that all future intervals line up on 15-minute wall clock boundaries. The values in this first report also can be associated with a much longer period, depending on the time of the last reset.

Another alternative to the use of interval statistics is to use CEMT to reset the counters and then at the end of the measurement period, use CEMT to record all the statistics. Resetting the statistics requires a change of state from ON to OFF or from OFF to ON. To ensure that this change happens, the following commands provide an example of resetting the statistics in one CICS region:

```
F CICS001,CEMT SET STAT OFF RESET
```

```
F CICS001,CEMT SET STAT ON RESET
```

The measurement period is between the RESET and the RECORD, as shown in the following example:

```
F CICS001,CEMT PERFORM STAT ALL RECORD
```

Regardless of whether the statistics are ON or OFF, when a **PERFORM STAT ALL RECORD** command is issued, a statistics record is written.

CICS statistics are written as SMF 110 subtype 2 records. They can be post-processed by using the CICS statistics utility program, DFHSTUP, or CICS Performance Analyzer (CICS PA).

2.5.4 CICS TS performance class monitoring

When CICS Performance Class Monitoring is turned on by using MNPER=ON in the CICS startup parameters or CEMT or CEMN transactions to turn it on dynamically, a Performance Class Monitoring record is generated for every executed transaction when the transaction ends.

The following command is an example of turning on CICS Performance Class Monitoring and Resource Class Monitoring in one CICS region:

```
F CICS001,CEMT SET MON ON PER RESRCE
```

Monitoring can then be turned off by using the following command:

```
F CICS001,CEMT SET MON ON NOPER NORESRCE
```

The performance class record of each transaction contains information about the resources that were used by that transaction, how much CPU was used on all the various task control blocks (TCBs), and information about how long it waited for different resources. Resource Class Monitoring records contain information about the individual files, temporary storage queues, and distributed program links (DPLs) that were used by transactions.

Monitoring records are written as SMF 110 subtype 1 records that can be analyzed by using CICS PA.

2.5.5 Hardware instrumentation counters and samples

The CPU Measurement Facility (CPU MF) is described in 1.1, “CPU Measurement Facility” on page 4. The CPU MF capability is built into the hardware, and a z/OS component called *hardware instrumentation services* (HIS) sets up buffers that the hardware then uses to store the sampling data. When a number of buffers are filled, the hardware generates an interrupt. This interrupt enables HIS to asynchronously collect the sampling information and save it to a file in the z/OS UNIX file system. It also provides the ability for the samples to be gathered without the software responsible for collecting the data, having to run at the highest Workload Manager priority level.

HIS can be used to collect the following types of data:

- ▶ Counters
- ▶ Instruction samples

HIS counters are written as System Management Facilities (SMF) 113 records and to the z/OS UNIX file system. These counters contain information about key hardware events, such as the number of instructions that are executed, the number of cycles that were used, and the amount of instruction cache and data cache misses. Counters are used to provide a high-level understanding of how the address spaces interact with the hardware.

HIS instruction samples are written only to the z/OS UNIX file system. The samples are used to provide a view of CPU activity for individual instructions or groups of instructions. Tooling enables the inspection of this data to help the CICS performance team understand where hot spots exist in the CICS runtime code. *Hot spots* are short sequences of one or two machine instructions that consume a disproportionately large fraction of the total CPU cost. These hot spots are frequently caused by data access patterns that do not make optimal use of the hardware cache subsystem. Tooling that is written to consume HIS instruction samples also permits the comparison of two benchmark runs, where differences in performance can be analyzed at the instruction level.

For more information about configuring and the use of HIS, refer to *Setting Up and Using the IBM System z CPU Measurement Facility with z/OS*, REDP-4727, which is available at this website:

<http://www.redbooks.ibm.com/abstracts/redp4727.html>

2.6 Collecting performance data

Performance data often is collected for five measurement intervals. The rate at which work is driven into CICS is varied by adjusting the Workload Simulator *user think time interval* (UTI). The UTI value represents the delay between a simulated client that is receiving a response, and then sending the next request into CICS. A large think time results in a low rate of transactions in CICS. Reducing the UTI increases the rate at which work is driven into the CICS environment.

The initial measurement period begins by adjusting the UTI to achieve the required transaction rate in the CICS regions. The workload can run for a period to ensure that all programs were loaded and the local shared resource (LSR) pools are populated. After the stabilization period is complete, the performance data collection is started.

No specific changes to any default CICS parameters are needed to support the data that is collected during performance benchmarks. Data is collected for a 5-minute period, which is relatively short but adequate in our environment when running in a steady-state.

RMF, CICS Performance Class Monitoring, CICS statistics, and HIS are all synchronized and started and ended together. An automation tool is used that enters commands on the IBM MVS™ console on a time-based interval.

To generate the RMF interval, start and stop RMF at the appropriate times, which creates an interval report for that period rather than trying to synchronize on a time basis.

When the workload is running at its stabilized state, the CICS statistics are reset by using the commands that are described in 2.5.3, “CICS TS statistics” on page 16. CICS Performance Class Monitoring is turned on by using the commands that are shown in 2.5.4, “CICS TS

performance class monitoring” on page 17. RMF Monitor I was started by using the following MVS command:

```
S RMF.R
```

Monitor III is then started by using the following command:

```
F R,START III
```

HIS is also started to collect counter data only.

After 5 minutes elapses, RMF and HIS are stopped, and the command that is shown in 2.5.3, “CICS TS statistics” on page 16 is issued to request that CICS statistics are recorded.

After the performance data collection period ends, the UTI is reduced, which increases the transaction rate in CICS. Again, the workload is allowed to run for a period to ensure that the system reaches a steady-state. After this stabilization period is complete, the performance data collection is restarted.

After five cycles of UTI adjustment and data collection, a set of data is produced which represents the performance of the CICS regions at several transaction rates. The SMF data set that contains the collected RMF, CICS, and HIS performance data is copied for later post-processing and analysis to examine the performance characteristics of the workload.

2.6.1 Collecting Java performance data

CICS performance class monitoring data does not account for all the CPU time that is consumed by a CICS region. Areas where the time spent is not included in the monitoring data can include the following examples:

- ▶ Non-CICS TCBs
- ▶ Service request blocks (SRBs) for networking or system calls
- ▶ Request initialization (that is, before a CICS task is established)
- ▶ Request termination (that is, after the CICS task monitoring data is written)

When running a Java workload, this *uncaptured* time is larger than that observed for more traditional workloads. This increased discrepancy happens for the following reasons:

- ▶ A running JVM has several non-CICS TCBs executing to perform critical functions. The most significant of these functions are garbage collection (GC) and JIT compilation. GC and JIT TCBs can use non-trivial amounts of CPU in the JVM.
- ▶ For applications using an IBM WebSphere Application Server Liberty (WebSphere Liberty) JVM server, the initial HTTP or HTTPS request is accepted in Java code. Therefore, a non-trivial amount of Java code is executed before CICS is notified of the request and, thus, before a CICS task is established.

For Java applications running in an OSGi JVM server, the discrepancy is lower than a WebSphere Liberty JVM server, because a CICS task is always established before invoking the OSGi JVM. The uncaptured time, therefore, is mostly due to the GC and JIT TCBs identified previously. This discrepancy is studied for two Java workloads in 7.16.4, “Comparison of CICS monitoring and RMF data” on page 147.

Given the potential for large amounts of uncaptured CPU time, it is important to use CPU time information measured at an address space level when analyzing the performance of CICS applications that use Java.



Workload descriptions

As described in 2.1, “Workloads” on page 12, a *workload* is a collection of application programs (started by a simulated operator) where the application accesses data from one or more data stores.

This chapter describes several general-purpose workloads that are used by IBM to report performance results in every release of CICS TS for z/OS. It also highlights some of the functional areas that are covered by the test cases.

This chapter includes the following topics:

- ▶ 3.1, “Regression testing” on page 22
- ▶ 3.2, “Data Systems Workload” on page 22
- ▶ 3.3, “Relational Transactional Workload” on page 25
- ▶ 3.4, “WebSphere Liberty servlet with JDBC and JCICS access” on page 26
- ▶ 3.5, “Java OSGi workload” on page 27
- ▶ 3.6, “Web services” on page 28
- ▶ 3.7, “File control workload” on page 29

3.1 Regression testing

One of the primary objectives of testing the performance of the CICS TS product is to ensure that customer workloads do not observe a degradation in performance when upgrading to a new release. The process of comparing one release of CICS TS with another by using identical workloads and identical configurations is known as *regression testing*.

Each chapter in Part 2, “CICS TS performance information” on page 39 provides information that relates to a specific CICS TS release. A significant component of all the release-specific chapters are the results of running regression test workloads. For more information about the details of workload results, see Chapter 5, “CICS TS for z/OS V5.1” on page 41, Chapter 6, “CICS TS for z/OS V5.2” on page 77, and Chapter 7, “CICS TS for z/OS V5.3” on page 109.

Note: Not all results of every workload that is executed during the development phase of a CICS TS release are presented in the performance report.

Many workloads are used during regression test. The remainder of this chapter outlines some of the workloads that are available to the CICS TS performance team in the IBM development organization.

3.2 Data Systems Workload

The Data Systems Workload (DSW) is a non-threadsafe COBOL application that accesses VSAM files. Transactions use Basic Mapping Support (BMS) maps to interface with 3270 terminals. The amount of files in use varies depending on configuration, but can be in the range of 16 - 320.

The DSW workload is composed of a number of transactions, where 50% of CICS transactions issue at least one file control (FC) request. On average, six FC requests are issued per CICS task. FC requests are distributed in the following percentages:

- ▶ 69% read
- ▶ 10% read for update
- ▶ 9% update
- ▶ 11% add
- ▶ 1% delete

To simulate users of the application in a controlled manner, IBM Workload Simulator for z/OS is configured to emulate many 3270 terminals. Depending on the configuration, the amount of simulated users can be in the range of 1,000 - 4,000.

Several configuration options are available for DSW. Some of these variants are described in the following sections.

3.2.1 DSW static routing

Five CICS regions are configured for the workload. Two terminal-owning regions (TOR) connect to two application-owning regions (AOR). These two AORs then connect to a file-owning region (FOR). Files are accessed in the FOR by using VSAM local shared resources (LSR).

Work enters the system in a TOR and is then transaction-routed to the corresponding AOR. The business logic of the workload then accesses the VSAM data by using CICS function shipping to the FOR. Temporary storage (TS) requests are fulfilled by using local, unrecoverable, auxiliary temporary storage.

All connections use the multi-region operation and cross-memory (MRO/XM) protocol. CICSplex System Manager is not used to provide dynamic workload routing in this scenario. Figure 3-1 shows the topology of DSW in a static routing configuration.

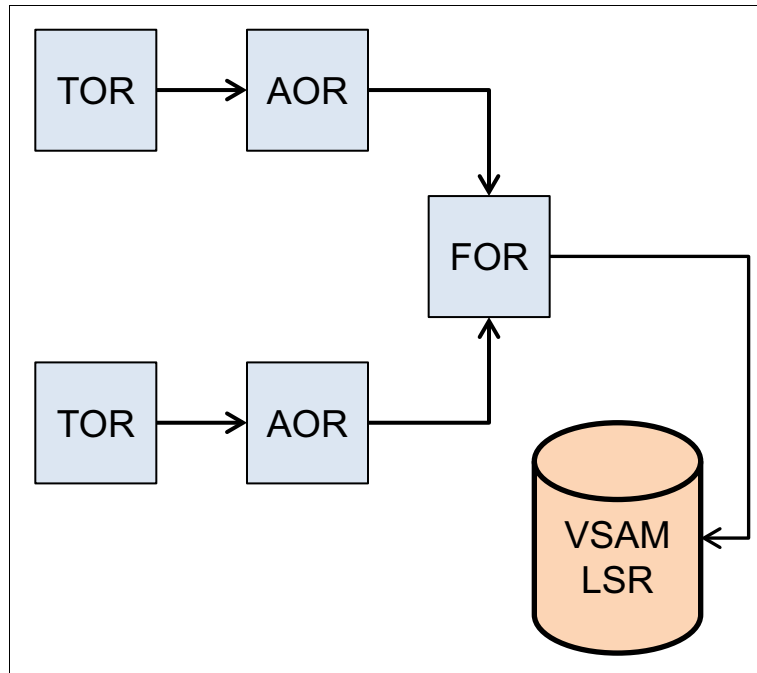


Figure 3-1 Topology of the DSW performance workload in static routing configuration

3.2.2 DSW dynamic routing

By using the same business logic and file structure as the static routing variant of the DSW, the application is extended to include the use of VSAM record-level sharing (RLS) and CICSplex System Manager (CICSplex SM) dynamic transaction routing. The use of CICSplex SM introduces the requirement for the following two CICS regions:

- ▶ CICSplex SM address space (CMAS) region

The CMAS region is the component of the CICSplex SM topology responsible for most of the work that is involved in managing and reporting on CICS regions and their resources. Each CICSplex must have at least one CMAS.

- ▶ Web user interface (WUI) server

The WUI server is a CICS region that acts as a CICSplex SM application, which uses the API to view and manage objects in the data repositories of CICSplex SM address spaces.

To remove application affinities and enable dynamic workload distribution, temporary storage requests are fulfilled by using shared temporary storage, which is held in the coupling facility (CF).

As with the static routing configuration, all connections between CICS regions use the MRO/XM protocol.

Transactions enter the system through TORs by using the BMS maps interface, and are then transaction-routed to an AOR. Although the static routing variant that is described in 3.2.1, “DSW static routing” on page 22 is connected to a single AOR only, all TORs are connected to all AORs in this scenario. A CICSplex SM workload is defined and installed to route transactions dynamically from the routing regions (the TORs) into the target regions (the AORs). File commands from the business logic use the support for VSAM RLS in CICS TS to access the required data.

Typically, a dynamic routing configuration uses four TORs and 30 AORs, although not all regions are highlighted in topology diagrams. Figure 3-2 shows the topology of the DSW workload when it is configured to use dynamic routing with CICSplex System Manager.

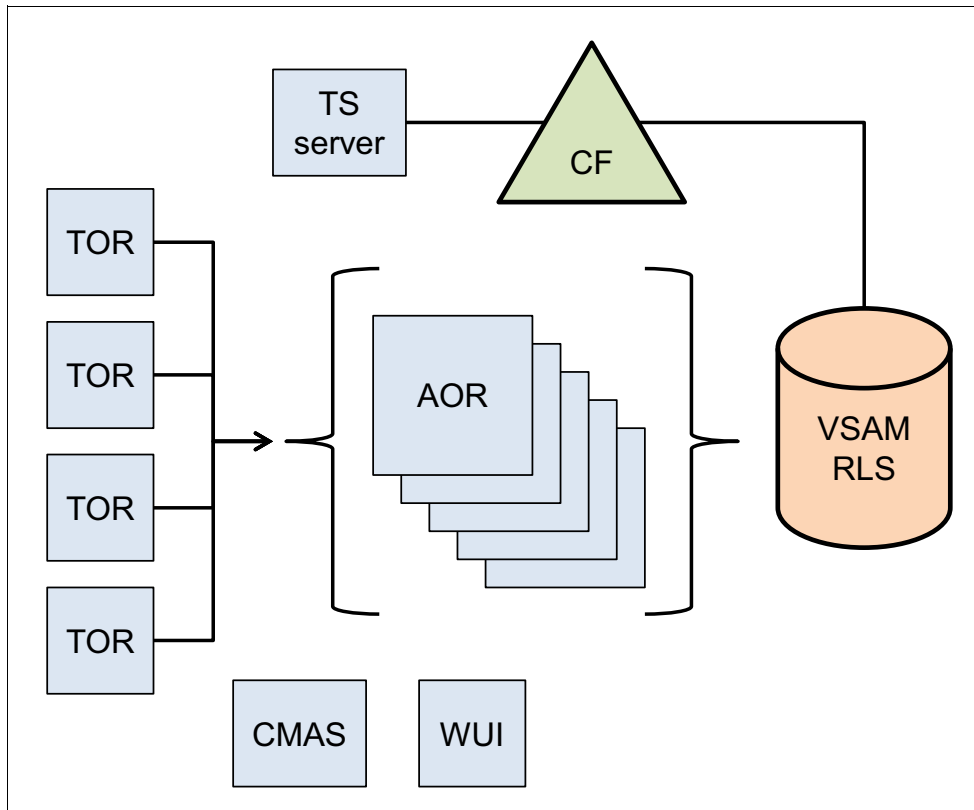


Figure 3-2 Topology of the DSW performance workload in dynamic routing configuration

3.2.3 DSW dynamic routing by using IPIC

The topology of this workload is the same as described in 3.2.2, “DSW dynamic routing” on page 23. The only difference between these variants is that IP interconnectivity (IPIC) is used to facilitate communication between CICS regions, rather than the MRO/XM protocol.

As described in 2.2, “Workload design” on page 12, workloads are designed to minimize any unnecessary overhead or variations in runtime performance. The DSW IPIC workload uses the TCP/IP *home* address (127.0.0.1) to avoid testing physical networking hardware.

3.3 Relational Transactional Workload

The Relational Transactional Workload (RTW) is a COBOL application that accesses a DB2 database. Transactions use BMS maps to interface with 3270 terminals.

Seven transaction types access 16 DB2 tables by using EXEC SQL commands. In total, slightly more than 30 million rows of data are defined in the database. On average, each CICS task issues 200 DB2 SQL calls. These SQL requests are distributed in the following percentages:

- ▶ 54% SELECT
- ▶ 1% INSERT
- ▶ 1% UPDATE
- ▶ 1% DELETE
- ▶ 8% OPEN CURSOR
- ▶ 27% FETCH CURSOR
- ▶ 8% CLOSE CURSOR

The two main variants of the RTW workload are non-threadsafe and threadsafe.

The non-threadsafe variant of RTW includes all **PROGRAM** resources defined with the **CONCURRENCY** attribute set to the value of **QUASIRENT**.

The threadsafe variant specifies the value of **REQUIRED**. For more information about the effects of this variation, see Chapter 4, “Open transaction environment” on page 31.

Typically, the RTW workload is executed as a stand-alone CICS region, but some test scenarios require many transactions. To achieve high transaction rates, a variant of the RTW workload is available and uses TORs and AORs in a similar manner that was described for DSW in 3.2.2, “DSW dynamic routing” on page 23. Figure 3-3 shows the topology of the RTW workload in a high-volume configuration.

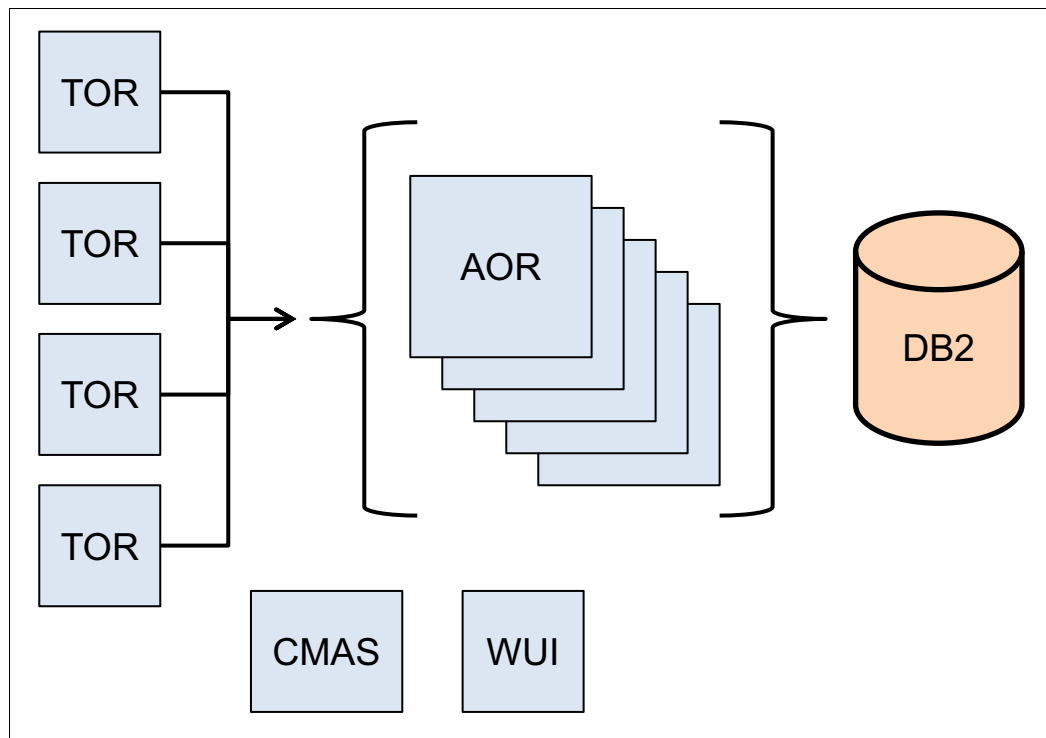


Figure 3-3 Topology of high-volume RTW configuration

3.4 WebSphere Liberty servlet with JDBC and JCICS access

The workload that is presented in this section is a pure Java application, based on the CICS-supplied JDBC sample application. For more information about the sample JDBC servlet application, see the “About the servlet examples” topic in IBM Knowledge Center, which is available at this website:

<https://ibm.biz/Bdi6nd>

The sample application is composed of a Java servlet, which accesses an SQL database by using the Java JDBC API and the IBM Data Server Driver for JDBC with type 2 connectivity. The values that are retrieved from the database are then rendered as an HTML page and sent to the client as the response. For this workload, the sample application was expanded to also access a VSAM file by using the JCICS interface. The VSAM file contains a copy of the data that is held in the sample database.

The supplied sample application uses the `CICSDB2DynamicSQLExceptionExample.getData(String)` method to read 42 rows from the sample DB2 table EMP. This method was modified to also read 42 records from a VSAM file by using JCICS `KeyedFileBrowse.next()` calls and display the data as extra entries in the HTML table that is returned to the client.

The application is deployed into a WebSphere Liberty environment inside a CICS JVM server. Simulated browser requests are made to the HTTP port that is specified in the `server.xml` configuration file. Figure 3-4 shows an overview of the application topology.

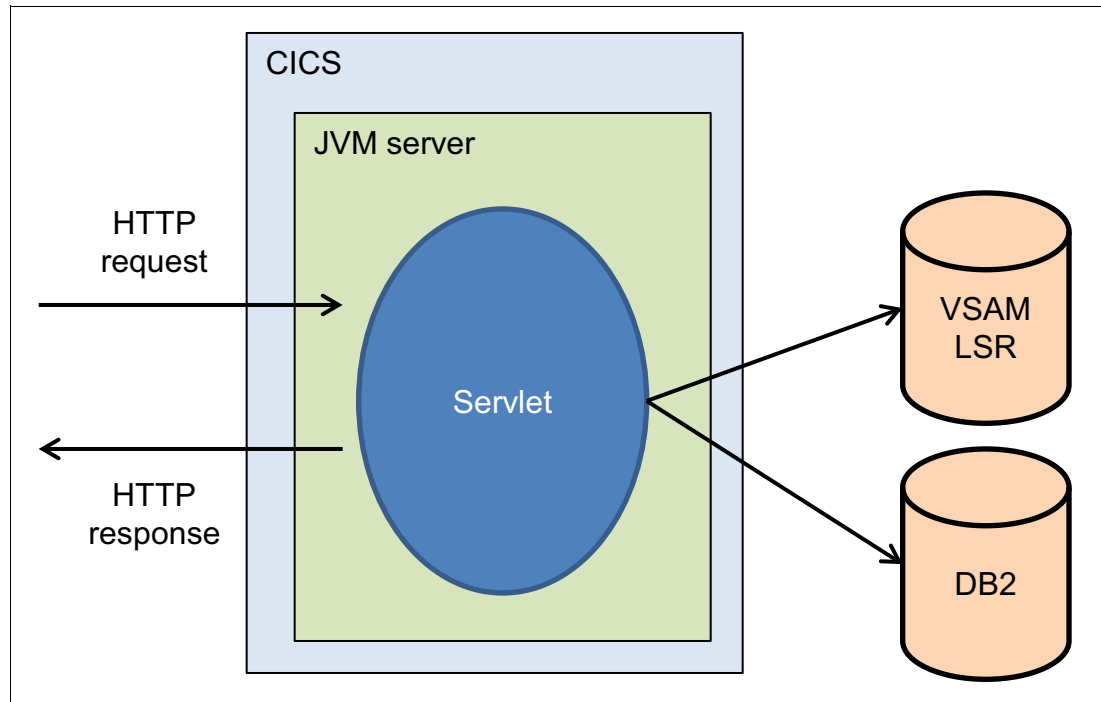


Figure 3-4 Topology of JDBC and VSAM servlet workload

The WebSphere Liberty server configuration file `server.xml` was automatically generated by specifying the following options in the JVM profile file for the JVM server:

- ▶ `-Dcom.ibm.cics.jvmserver.wlp.autoconfigure=true`
- ▶ `-Dcom.ibm.cics.jvmserver.wlp.server.host=hostname`
- ▶ `-Dcom.ibm.cics.jvmserver.wlp.server.name=serverName`
- ▶ `-Dcom.ibm.cics.jvmserver.wlp.server.http.port=httpPort`

The following entry was also added to the JVM profile configuration file to optimize retrieval of static resources from the web application:

```
-Dcom.ibm.cics.jvmserver.wlp.optimize.static.resources=true
```

By using IBM Workload Simulator for z/OS, 200 HTTP clients were simulated to provide a controlled rate of transactions in the CICS region under test. The workload measured overall central processor usage and zIIP usage of the address space.

3.5 Java OSGi workload

The Java OSGi workload is composed of several applications. This mixture includes some of the JCICS sample applications as described in the “The JCICS example programs” topic in IBM Knowledge Center at this website:

<https://ibm.biz/Bd4zcX>

The CICS BUNDLE JDBC example, “Hello World,” and temporary storage queue (TSQ) example were modified to include Java programming to simulate extra business logic, such as creating and manipulating strings, generating random numbers, and performing mathematical operations on these numbers.

The workload is driven by running CICS transactions at a simulated console by using IBM Workload Simulator for z/OS, as described in 2.4, “Driving the workload” on page 16.

Figure 3-5 shows an overview of the workload.

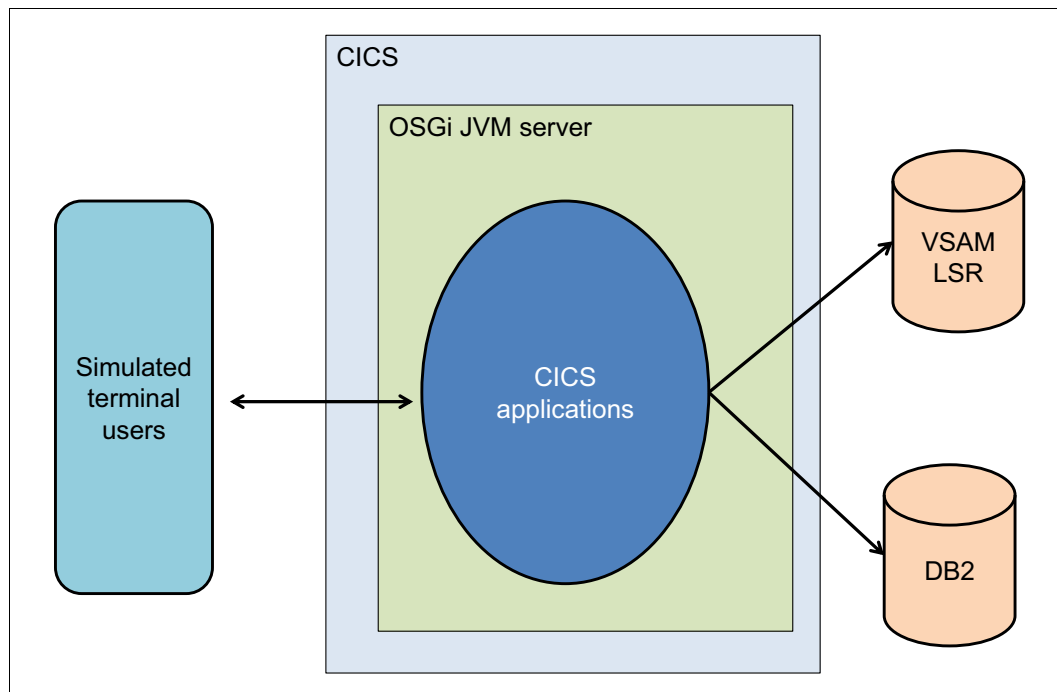


Figure 3-5 Overview of OSGi Java workload

3.6 Web services

The web services set of workloads measure the performance of CICS as a SOAP web service provider. These workloads contain a range of configuration options for testing, including variations of:

- ▶ Connection persistence
- ▶ SSL usage
- ▶ SSL provider
- ▶ SSL handshake type (none, partial, or full)

All variations follow the same application topology, as shown in Figure 3-6. Notice that other resources are required to configure CICS to use web services, but these have been omitted for clarity. For a detailed setup guide, see the “Configuring web services in CICS” topic in IBM Knowledge Center at this website:

<https://ibm.biz/BdZYwd>

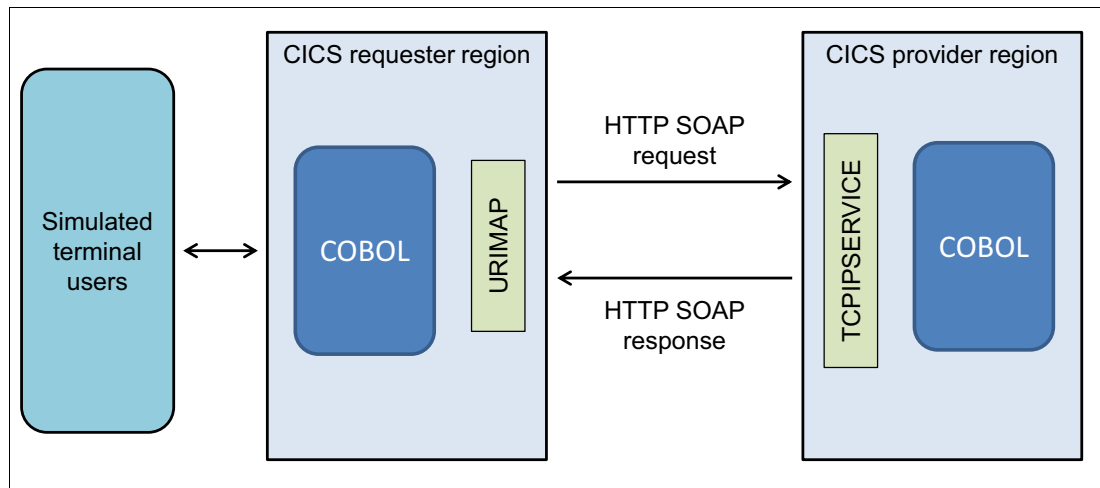


Figure 3-6 Topology of CICS web services workload

The DFHLS2WS utility is used to produce a CICS wsbind file. This file is installed into the CICS *provider* region and configures CICS to invoke a COBOL application using a channel interface.

Two copybooks are used as input to the utility:

- ▶ The first copybook defines the input data to the SOAP web service.
- ▶ The second copybook defines the output data from the SOAP web service.

Both copybooks are similar, with the input data copybook shown in Example 3-1. This configuration results in a web service that accepts a SOAP message that contains 10 80-byte fields as an input and returns a SOAP message that contains 10 80-byte fields as an output.

Example 3-1 Copybook used for input data to SOAP web service

```
01 RECEIVED-DATASTRUCTURE.
   02 JB1          PIC X(80).
   02 JB2          PIC X(80).
   02 JB3          PIC X(80).
   02 JB4          PIC X(80).
   02 JB5          PIC X(80).
   02 JB6          PIC X(80).
```

02 JB7	PIC X(80).
02 JB8	PIC X(80).
02 JB9	PIC X(80).
02 JB10	PIC X(80).

The workload is driven using a second CICS region (the *requester* region) using a URIMAP resource to send web service requests into the CICS provider region under test. The requester region runs on a separate LPAR and is driven by running CICS transactions at a simulated console by using IBM Workload Simulator for z/OS, as described in 2.4, “Driving the workload” on page 16.

The CPU consumption of the CICS provider region is measured during execution of the benchmark. When combined with the known arrival rate of the SOAP messages, a CPU per request value can be obtained.

3.6.1 Web services variations

Test variations are achieved by modification of resources in both the provider and requester regions, with the following available options:

- ▶ Connection persistence

The choice between persistent and non-persistent connections is configured by modifying the SOCKETCLOSE attribute in the URIMAP of the requester region. When set to zero, the connection closes after the response message arrives.

- ▶ Use of SSL

SSL is enabled by specifying the SCHEME(HTTPS) parameter in the URIMAP of the requester region.

- ▶ Use of CICS SSL or AT-TLS

The use of CICS SSL is configured by specifying SSL(YES) for the TCPIPSERVICE parameter in the provider region. The use of Application Transparent Transport Layer Security (AT-TLS) is achieved by specifying SSL(NO) for the TCPIPSERVICE parameter in the provider region and configuring AT-TLS in IBM Communication Server.

- ▶ SSL handshake type

No SSL handshakes are performed during the measurement period when persistent connections are used; the configuration of this is described previously. When non-persistent connections are used, the CICS SSLDELAY SIT parameter in the requester region controls whether a full or a partial SSL handshake is performed.

A setting of zero for SSLDELAY specifies that the provider region will not cache SSL session tokens, and therefore a full SSL handshake will take place for every new connection.

A non-zero setting for SSLDELAY specifies the amount of time in seconds for which the provider region will cache the SSL session token. A value of 600 is used to guarantee the session token is always cached between successive requests, regardless of the transaction rate used.

3.7 File control workload

The DSW application described in 3.2, “Data Systems Workload” on page 22 predominantly exercises the CICS File Control interface, however the application is not threadsafe. As described in 9.3, “Improvements in threadsafety” on page 206 the CICS TS V5.5 release

introduced the ability to access Coupling Facility Data Tables (CFDTs) from an Open TCB. Testing this functionality required a new threadsafe workload to exercise the CICS File Control API. This section describes the generic threadsafe File Control workload that can be configured for several different scenarios.

All programs in the application are written using threadsafe programming practices. Three main programs are used, with the following purpose:

- ▶ Terminal handling program

This program reads and validates input from the 3270 terminal. The input is used to specify several configuration options. These configuration options are stored in a channel and passed to the *TCB switch program* using **EXEC CICS LINK**.

Specifies **QUASIRENT** for the **CONCURRENCY** attribute.

- ▶ TCB switch program

This program accepts configuration from the terminal handling program and then passes this configuration unchanged to the *main application logic program* using **EXEC CICS LINK**. The TCB switch program is used only to ensure the main application logic program begins execution on the correct TCB.

When binding the TCB switch program, the binder produces two aliases and both aliases are defined as **PROGRAM** resources in the CICS region. Each program definition specifies a different value for the **CONCURRENCY** attribute. The terminal handling program can select which TCB that the main application program should use, simply by selecting the correct program alias. For more details on TCB switching in CICS, see Chapter 4, “Open transaction environment” on page 31.

Specifies **QUASIRENT** (non-threadsafe configuration) or **REQUIRED** (threadsafe configuration) for the **CONCURRENCY** attribute.

- ▶ Main application logic program

This program accepts the supplied configuration and translates this into the requested combination of CICS File Control calls.

Specifies **THREADSAFE** for the **CONCURRENCY** attribute.

A minimum of two and a maximum of 1,000 VSAM KSDS files can be defined in the application. Each file can be defined to have one of three record lengths: 64 bytes, 100 bytes, or 256 bytes.

Each transaction accesses a number of records, which is specified by the data supplied at the terminal. A *read-only* transaction performs one **EXEC CICS READ** command per record. An *update* transaction performs one **EXEC CICS READ UPDATE** and one **EXEC CICS REWRITE** command per record. All file reads compute a hash value to validate data integrity.



Open transaction environment

This chapter introduces the open transaction environment and provides a reference for the task control block (TCB) modes that are used when an application is dispatched within CICS.

This chapter includes the following topics:

- ▶ 4.1, “Introduction to the open transaction environment” on page 32
- ▶ 4.2, “TCB modes” on page 32
- ▶ 4.3, “Changing TCB modes” on page 33
- ▶ 4.4, “Understanding the effect of change mode operations” on page 36

4.1 Introduction to the open transaction environment

The open transaction environment (OTE) is an environment where threadsafe application code can run on its own TCB inside the CICS address space without interference from other transactions. Applications that use the OTE can run on a class of TCB called an open TCB, or on the quasi-reentrant TCB (QR TCB).

The use of an open TCB provides the following advantages over the use of the QR TCB:

- ▶ No dispatching of other CICS tasks occurs on an open TCB.
- ▶ Multiple open TCBs can run concurrently.
- ▶ An application that runs under an open TCB can issue non CICS API requests that can involve the TCB being blocked. If blocking occurs, only this open TCB is halted, and not the whole of CICS.
- ▶ When the CICS task ends, the open TCB can be reused by another CICS task.

CICS manages open TCBs in separate pools, with each pool containing a different type (or mode) of open TCB. When applications are dispatched onto a TCB, the type of TCB depends on the combination of several CICS and resource configuration parameters:

- ▶ The storage protection setting for the CICS region
- ▶ Whether the application executes in a JVM server
- ▶ The parameters that are used during the compilation and binding process of the application
- ▶ The value of the following PROGRAM resource attributes:
 - API
 - CONCURRENCY
 - EXECKEY

For more information about the open transaction environment, see the “Threadsafe learning path” topic in IBM Knowledge Center at this website:

<https://ibm.biz/Bdi6G7>

4.2 TCB modes

CICS manages open TCBs in separate pools, with each pool containing a different type (or mode) of open TCB. Each mode has a two-character identifier to indicate its specific purpose, and is handled by CICS in a different way. The following open TCB modes are used by CICS when running user applications:

- ▶ L8 and L9 mode TCBs

These TCBs are used to run threadsafe programs; that is, programs that are defined as CONCURRENCY (THREADSAFE) or CONCURRENCY (REQUIRED) in the PROGRAM resource definition. Consider the following points:

- L8 mode TCBs are used by CONCURRENCY (THREADSAFE) and CONCURRENCY (REQUIRED) application programs that specify API (CICSAPI) because CICS services do not require TCB key matching.

L8 mode TCBs are used for application programs that specify API (OPENAPI) with EXECKEY (CICSKEY).

- L9 mode TCBs are used for application programs that specify API (OPENAPI) with EXECKEY (USERKEY).
- ▶ T8 mode TCBs
 - These TCBs are used to run Java programs in a JVMSERVER resource.
- ▶ X8 and X9 mode TCBs
 - These TCBs are used to run C and C++ programs compiled with the XPLINK option. Consider the following points:
 - X8 mode TCBs are used for XPLink application programs that specify EXECKEY (CICSKEY).
 - X9 mode TCBs are used for XPLink application programs that specify EXECKEY (USERKEY).

4.3 Changing TCB modes

A CICS task is not restricted to execution on a single TCB. During the lifetime of a task, the CICS dispatcher can perform multiple TCB change mode operations to provide the user application with the correct environment.

Although the change mode operation is not apparent to the user application at run time, it incurs a small performance penalty. A change mode operation has a path length of approximately 3,000 instructions, which equates to a small CPU overhead. The application can also be suspended if a suitable TCB is not available immediately. An excessive number of change mode operations can result in poor application performance. Minimizing TCB change modes can reduce the CPU used by an application, while improving throughput.

The remainder of this section provides a reference to the type of TCB that is used when executing application code and the required change mode operations based on the combination of CICS region and resource parameter values. The following categories of applications are described:

- ▶ Java programs
- ▶ Programs specifying JVM(N0) and API (CICSAPI)
- ▶ Programs specifying JVM(N0) and API (OPENAPI)
- ▶ Programs compiled with the XPLINK option

Note: Task termination occurs on the QR TCB. A switch from an open TCB to the QR TCB is always necessary.

4.3.1 Java programs

CICS programs that reference a Java application must specify the following attribute values on the resource definition:

- ▶ JVM (YES)
- ▶ EXECKEY (CICS)
- ▶ CONCURRENCY (REQUIRED)

Java applications run on a T8 mode open TCB and never on the QR TCB.

Note: For compatibility with previous releases, CONCURRENCY (THREADSAFE) is the default value for Java programs, but the preferred option to use is CONCURRENCY (REQUIRED).

4.3.2 Programs specifying JVM(NO) and API(CICSAPI)

Where a CICS PROGRAM definition specifies the attribute values JVM(NO) and API(CICSAPI), the application was written in a non Java language and the program is restricted to the use of only the CICS permitted application programming interfaces.

If the program is defined with CONCURRENCY(QUASIRENT), it always runs on the QR TCB. If the program is defined with CONCURRENCY(THREADSAFE), it runs on whichever TCB is in use by CICS at the time that is determined as suitable. If the program is defined with CONCURRENCY(REQUIRED), it always runs on an open TCB.

Table 4-1 lists the TCB modes that are used, depending on the value of the CONCURRENCY attribute.

Table 4-1 TCB mode switch table for programs specifying JVM(NO) and API(CICSAPI)

CONCURRENCY	Initial TCB	DB2 or IBM MQ command	Non-threadsafe CICS command
QUASIRENT	QR	QR →L8 →QR	No change
THREADSAFE	QR	L8	QR
REQUIRED	L8	No change	L8 →QR →L8

Note: Executing a threadsafe CICS API command does not cause a TCB switch for any value of the CONCURRENCY attribute.

The following example uses a configuration that is listed in the last row of Table 4-1 and corresponds to a program that is defined with the attribute values:

- ▶ JVM(NO)
- ▶ API(CICSAPI)
- ▶ CONCURRENCY(REQUIRED)

When the program is run, the following sequence of TCB modes is used:

1. The program begins execution. The program is initially dispatched on an L8 TCB.
2. The program executes a threadsafe CICS command. No TCB change mode is required.
3. The program executes a DB2 command. No TCB change mode is required.
4. The program executes a non-threadsafe CICS command, which results in the following TCB change mode processing:
 - a. Execution switches from an L8 TCB to the QR TCB.
 - b. The CICS command is started and completes.
 - c. Execution switches from the QR TCB back to an L8 TCB.
 - d. Control is returned to the user program.
5. The program completes. A TCB change mode operation occurs, switching from an L8 TCB to the QR TCB for task termination processing.

This approach to understanding Table 4-1 can be applied to Table 4-2 on page 35 and Table 4-3 on page 36.

4.3.3 Programs specifying JVM(NO) and API(OPENAPI)

Where a CICS PROGRAM definition specifies the attribute values JVM(NO) and API(OPENAPI), the application was written in a non Java language and the program is not restricted to use only the CICS application programming interfaces. To specify the OPENAPI value, your program must be coded to threadsafe standards and defined with CONCURRENCY(REQUIRED).

Note: The combination of CONCURRENCY(THREADSAFE) with API(OPENAPI) that was supported in previous releases is deprecated but is kept for compatibility, and produces the same behavior as CONCURRENCY(REQUIRED) with API(OPENAPI).

Table 4-2 lists the TCB on which a CICS program runs when various combinations of the STGPROT SIT parameter and the EXECKEY attribute of the PROGRAM resource are used.

Table 4-2 TCB mode switch table for programs specifying JVM(NO) and API(OPENAPI)

STGPROT	EXECKEY	Initial TCB	DB2 or IBM MQ command	Non-threadsafe CICS command
NO	CICS	L8	No change	L8 →QR →L8
	USER			
YES	CICS	L8	No change	L8 →QR →L8
	USER	L9	L9 →L8 →L9	L9 →QR →L9

Note: Executing a threadsafe CICS API command does not cause a TCB switch for programs specifying OPENAPI.

4.3.4 Programs compiled with the XPLINK option

Programs that are compiled to use XPLink must be coded to threadsafe standards and be defined as API(OPENAPI) and CONCURRENCY(REQUIRED). The presence of the XPLink signature in a load module takes precedence over the API attribute on the PROGRAM resource definition.

Note: The combination of CONCURRENCY(THREADSAFE) with API(OPENAPI) that was supported in previous releases is deprecated but is kept for compatibility, and produces the same behavior as CONCURRENCY(REQUIRED) with API(OPENAPI).

Table 4-3 lists the TCB on which an XPLink program runs when various combinations of the STGPROT SIT parameter and the EXECKEY attribute of the PROGRAM resource are used.

Table 4-3 TCB mode switch table for programs that use the XPLink feature

STGPROT	EXECKEY	Initial TCB	DB2 or IBM MQ command	Non-threadsafe CICS command
NO	CICS	X8	X8 →L8 →X8	X8 →QR →X8
	USER			
YES	CICS	X8	X8 →L8 →X8	X8 →QR →X8
	USER	X9	X9 →L8 →X9	X9 →QR →X9

Note: Executing a threadsafe CICS API command does not cause a TCB switch for XPLink programs.

4.4 Understanding the effect of change mode operations

There are several tools available that can help you understand the effect of change mode operations.

4.4.1 CICS Monitoring Facility

CICS Monitoring Facility (CMF) data provides several fields that are related to the performance of applications and their interaction with the CICS dispatcher. The DSCMDLY field in the DFHTASK group provides several change mode operations. It also provides and the total elapsed time the task was suspended awaiting redispach following a change mode operation. For more information about the DFHTASK group, see “Performance data in group DFHTASK” in IBM Knowledge Center at this website:

<https://ibm.biz/Bdi6G5>

4.4.2 IBM CICS Performance Analyzer for z/OS

IBM CICS Performance Analyzer for z/OS (CICS PA) is a powerful offline reporting tool to help you tune and manage your CICS systems. By using the CMF and CICS statistics data, CICS PA provides comprehensive performance reporting and analysis capabilities. CICS PA is supplied with many sample reports. Several of these sample reports enable the analysis of TCB change mode operations. For more information about the features and use of CICS PA, see the following website:

<http://www.ibm.com/software/products/en/cics-panaly>

4.4.3 IBM CICS Interdependency Analyzer for z/OS

IBM CICS Interdependency Analyzer for z/OS (CICS IBM IA®) is a dynamic discovery tool that helps you understand the relationships, dependencies, and flows of CICS applications. The CICS IA product provides threadsafe analysis tooling that enables application developers and systems programmers to optimize the execution of program code. For more information about the features and use of CICS IA, see *IBM CICS Interdependency Analyzer*, SG24-6458, and the following website:

<http://www.ibm.com/software/products/en/cics-ianaly>

CICS TS performance information

This part features one chapter per CICS release. Several topics are covered in each chapter, where each topic can follow one of the following themes:

- ▶ Performance results for regression workloads, comparing the documented release and its immediate predecessor.
- ▶ Improvements in threadsafe capabilities of the CICS runtime code.
- ▶ Changes to SIT parameters that can have a significant effect on CICS performance.
- ▶ Updates to CICS statistics and monitoring data.
- ▶ Improvements in the CICS release that provide virtual storage constraint relief.
- ▶ Special performance studies that document the performance implications of enabling specific CICS features.
- ▶ Other small performance studies that are created in response to client feedback.

This part includes the following chapters:

- ▶ Chapter 5, “CICS TS for z/OS V5.1” on page 41
- ▶ Chapter 6, “CICS TS for z/OS V5.2” on page 77
- ▶ Chapter 7, “CICS TS for z/OS V5.3” on page 109
- ▶ Chapter 8, “CICS TS for z/OS V5.4” on page 149
- ▶ Chapter 9, “CICS TS for z/OS V5.5” on page 197



CICS TS for z/OS V5.1

The IBM CICS Transaction Server (TS) for IBM z/OS (CICS TS) V5.1 release introduces various technical and operational capabilities. Included in these updates are many improvements that provide performance benefits over previous CICS releases.

Included in the CICS V5.1 performance report are the following subject areas:

- ▶ Key performance benchmarks that are presented as a comparison with the CICS TS V4.2 release.
- ▶ An outline of improvements made regarding the threadsafe characteristics of the CICS run time.
- ▶ Details of the changes made to performance-critical CICS initialization parameters, and the effect of these updates.
- ▶ Description of all the updated monitoring fields, including examples where necessary.
- ▶ The extent and effect of the reduction in 24-bit and 31-bit virtual storage usage.
- ▶ High-level views of new functions that were introduced in the CICS V5.1 release, including performance benchmark results where appropriate.
- ▶ A description of transaction isolation and how changes that were introduced in this release might affect workloads with this feature enabled.

This chapter includes the following topics:

- ▶ 5.1, “Introduction” on page 42
- ▶ 5.2, “Release-to-release comparisons” on page 42
- ▶ 5.3, “Improvements in threadsafety” on page 48
- ▶ 5.4, “Changes to system initialization parameters” on page 52
- ▶ 5.5, “Enhanced instrumentation” on page 54
- ▶ 5.6, “Virtual storage constraint relief” on page 60
- ▶ 5.7, “64-bit application support” on page 62
- ▶ 5.8, “Java 7 and zEnterprise EC12” on page 62
- ▶ 5.9, “CICSplex System Manager dynamic routing” on page 64
- ▶ 5.10, “Workload consolidation” on page 66
- ▶ 5.11, “Effect of threadsafe transient data” on page 71
- ▶ 5.12, “Transaction isolation” on page 73

5.1 Introduction

When compiling the results for this chapter, the workloads were run on an IBM zEnterprise 196 model M80 (machine type 2817). A maximum of 16 dedicated central processors (CPs) were available on the measured logical partition (LPAR), with a maximum of four dedicated CPs available to the LPAR used to simulate users. These LPARs are configured as part of an IBM Parallel Sysplex®. An internal coupling facility (CF) was co-located on the same central processor complex (CPC) as the measurement and driving LPARs, connected through internal coupling peer (ICP) links. An IBM System Storage® DS8800 unit was used to provide external storage.

This chapter presents the results of several performance benchmarks when run in a CICS TS for z/OS V5.1 environment. Unless otherwise stated in the results, the CICS V5.1 environment was the code that was available on the general availability date of 14 December 2012. Several of the performance benchmarks are presented in the context of a comparison with CICS TS V4.2. All LPARs used z/OS V1.13.

For more information about performance terms that are used in this chapter, see Chapter 1, “Performance terminology” on page 3. For more information about the test methodology that was used, see Chapter 2, “Test methodology” on page 11. For more information about the workloads that were used, see Chapter 3, “Workload descriptions” on page 21.

Where reference is made to an *LSPR processor equivalent*, the indicated machine type and model can be found in the large systems performance reference (LSPR) document. For more information about obtaining and using LSPR data, see 1.3, “Large Systems Performance Reference” on page 6.

5.2 Release-to-release comparisons

This section describes some of the results from a selection of regression workloads that are used to benchmark development releases of CICS TS. For more information about the use of regression workloads, see Chapter 3, “Workload descriptions” on page 21.

5.2.1 Data Systems Workload static routing

The static routing variant of the Data Systems Workload (DSW) is described in 3.2.1, “DSW static routing”. This section presents the performance figures that were obtained by running this workload. The LPAR used for measurement was configured with 16 CPs online, which resulted in an LSPR processor equivalent of 2817-716.

Table 5-1 lists the results of the DSW static routing workload that used the CICS TS V4.2 release.

Table 5-1 CICS TS V4.2 results for DSW static routing workload

ETR	CICS CPU	CPU per transaction (ms)	LPAR busy
2498.52	75.86%	0.304	6.78%
2928.69	88.35%	0.302	7.79%
3543.47	104.08%	0.294	9.09%
4428.34	129.16%	0.292	11.13%
5944.91	168.58%	0.284	14.34%

Table 5-2 lists the same figures for the CICS TS V5.1 release.

Table 5-2 CICS TS V5.1 results for DSW static routing workload

ETR	CICS CPU	CPU per transaction (ms)	LPAR busy
2496.35	77.55%	0.311	6.89%
2939.62	87.18%	0.297	7.65%
3532.10	102.29%	0.290	8.86%
4425.48	126.17%	0.285	10.80%
5948.50	166.52%	0.280	14.07%

The average CPU per transaction figure for CICS TS V4.2 is calculated to be 0.295 ms, and the CICS TS V5.1 figure is also calculated to be 0.292 ms. The performance of this workload is considered to be equivalent across the two releases.

These performance results are also shown in Figure 5-1.

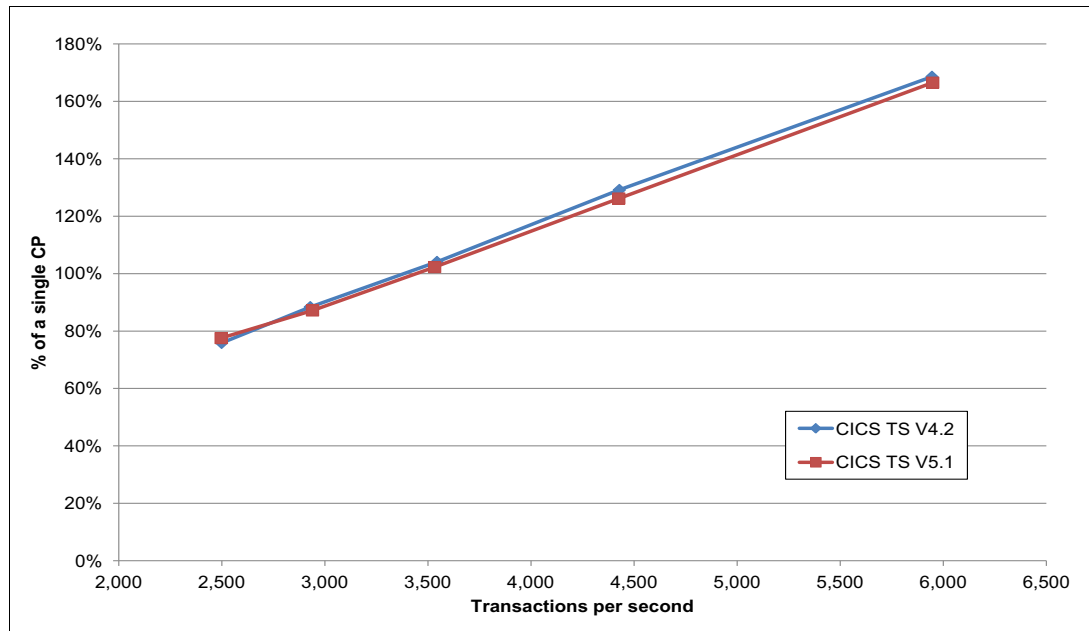


Figure 5-1 Plot of CICS TS V4.2 and V5.1 performance figures for DSW static routing workload

The measured CPU cost for each transaction rate is similar for CICS TS V4.2 and V5.1, with the CICS TS V5.1 release showing a marginal improvement. CPU cost scales linearly in accordance with the transaction rate.

5.2.2 DSW dynamic routing

The dynamic routing variant of the DSW workload is described in 3.2.2, “DSW dynamic routing”. This section presents the performance figures that were obtained by running this workload. The workload was configured with four terminal-owning regions (TORs) dynamically routing transactions to 30 application-owning regions (AORs). The LPAR that was used for measurement was configured with eight CPs online, which resulted in an LSPR processor equivalent of 2817-708.

Table 5-3 lists the results of the DSW dynamic routing workload that used the CICS TS V4.2 release.

Table 5-3 CICS TS V4.2 results for DSW dynamic routing workload

ETR	CICS CPU	CPU per transaction (ms)	LPAR busy
2071.61	141.20%	0.682	21.05%
2842.02	189.11%	0.665	27.85%
4128.25	270.70%	0.656	39.41%
5047.36	326.08%	0.646	47.24%
6493.98	417.16%	0.642	60.21%

Table 5-4 lists the same figures for the CICS TS V5.1 release.

Table 5-4 CICS TS V5.1 results for DSW dynamic routing workload

ETR	CICS CPU	CPU per transaction (ms)	LPAR busy
2074.87	139.91%	0.674	20.87%
2846.00	188.55%	0.663	27.78%
4133.39	269.54%	0.652	39.32%
5053.15	326.22%	0.646	47.33%
6501.18	416.92%	0.641	60.25%

The average CPU per transaction figure for CICS TS V4.2 is calculated to be 0.658 ms, and the CICS TS V5.1 figure is also calculated to be 0.655 ms. The performance of this workload is considered to be equivalent across the two releases. Figure 5-2 shows the results from Table 5-3 on page 44 and Table 5-4.

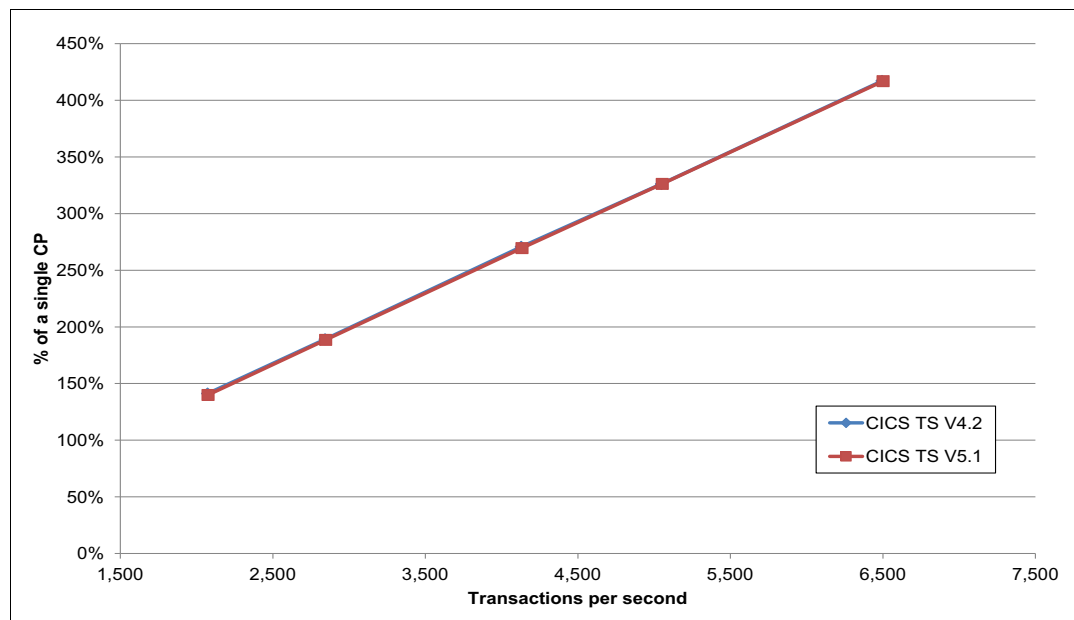


Figure 5-2 Plot of CICS TS V4.2 and V5.1 performance figures for DSW dynamic routing workload

You can see the V4.2 and V5.1 lines are overlaid, which indicates near-identical CPU cost per transaction. The plot lines are also straight, which indicates linear scaling as transaction throughput increases.

5.2.3 Relational Transactional Workload threadsafe

This section presents the performance figures for the threadsafe variant of the Relational Transactional Workload (RTW), as described in 3.3, “Relational Transactional Workload” on page 25.

Table 5-5 on page 46 lists the results of the RTW threadsafe workload that used the CICS TS V4.2 release.

Table 5-5 CICS TS V4.2 results for the RTW threadsafe workload

ETR	CICS CPU	CPU per transaction (ms)	LPAR busy
249.69	53.59%	2.146	21.33%
361.55	77.65%	2.148	30.93%
474.66	101.46%	2.138	39.85%
592.37	125.40%	2.117	48.89%
730.20	153.82%	2.107	59.51%

Table 5-6 lists the same figures for the CICS TS V5.1 release.

Table 5-6 CICS TS V5.1 results for the RTW threadsafe workload

ETR	CICS CPU	CPU per transaction (ms)	LPAR busy
249.98	54.19%	2.168	21.63%
361.88	78.35%	2.165	31.26%
474.86	101.42%	2.136	39.74%
592.74	126.14%	2.128	49.20%
729.98	155.06%	2.124	59.98%

The average CPU per transaction figure for CICS TS V4.2 is calculated to be 2.131 ms, and the CICS TS V5.1 figure is also calculated to be 2.144 ms. The difference between these figures is less than 1%, so the performance of this workload is considered to be equivalent across the two releases. Figure 5-3 shows these performance results.

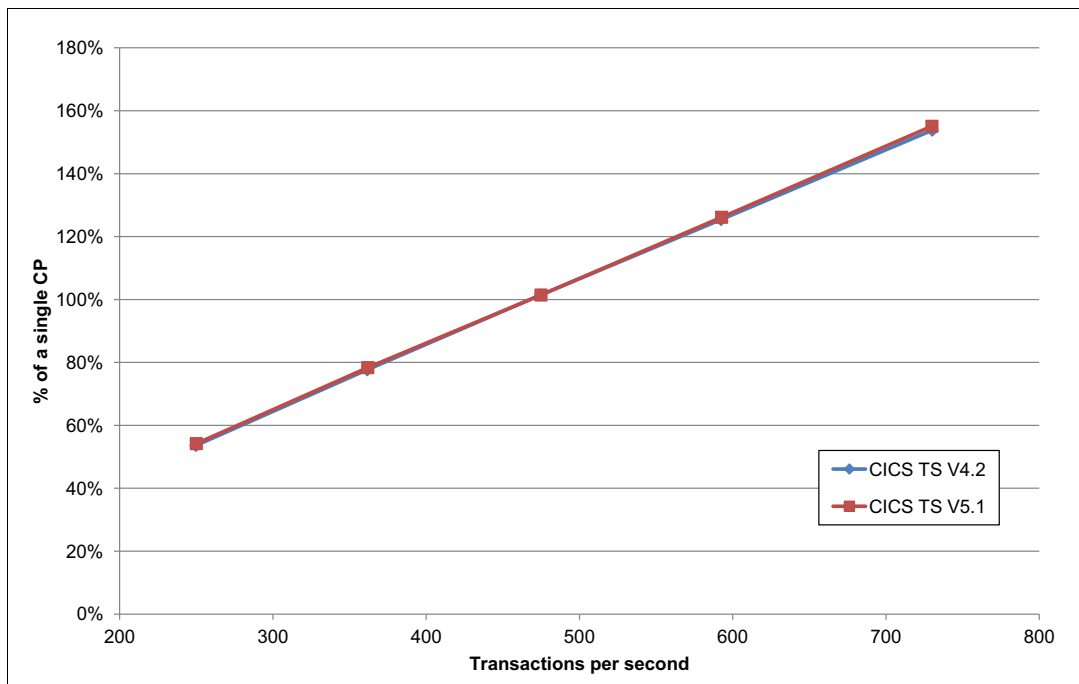


Figure 5-3 Plot of CICS TS V4.2 and V5.1 performance figures for RTW threadsafe workload

As previously observed, the straight line indicates linear scaling as throughput increases, and the overlaid lines demonstrate equivalent performance between the two CICS releases.

5.2.4 Java throughput

CICS TS V4.2 supports Java V6.0.1 only, whereas CICS TS V5.1 supports Java 7.0 only. This section compares the throughput of a Java workload that is running in CICS TS V4.2 that uses Java 6.0.1 with the same workload that is running in CICS TS V5.1 that uses Java 7.0.

Note: APAR PI30532 enables support for Java 7.1 in CICS TS V5.1. APAR PI52819 enables support for Java 8 in CICS TS V5.1.

The workload was an intensive Java application that performed some JCICS calls. The workload ran in a single CICS region that contained one JVMSERVER resource. A high transaction injection rate was maintained to drive a zEnterprise 196 model M80 with eight GCPs and one zIIP to maximum utilization. Data was collected in 1-minute intervals, and the THREADLIMIT attribute of the CICS JVM server was increased every 5 minutes. Each benchmark started with two JVM server threads.

The benchmarks used Java V6.0.1 SR3 and Java V7.0 SR3. The following configuration parameters were used in both cases:

- ▶ -Xms600M
- ▶ -Xmx600M
- ▶ -Xmns500M
- ▶ -Xmos100M
- ▶ -Xgcpolicy:gencon

The chart that is shown in Figure 5-4 plots the throughput in transactions per second as the THREADLIMIT attribute of the JVM server was increased.

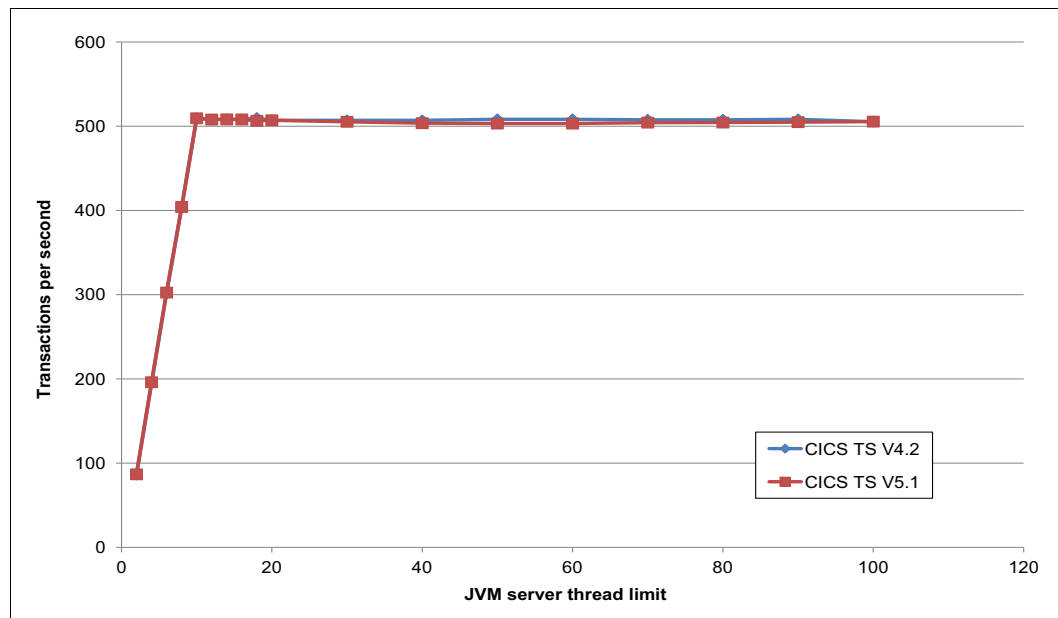


Figure 5-4 Throughput comparison for a Java workload in CICS TS V4.2 and CICS TS V5.1

For both configurations, the CPU utilization reached 99.9% when the JVM server reached nine concurrent threads. It can be seen from Figure 5-4 on page 47 that similar performance is observed in both configurations, and both configurations scale well.

For more information about the benefits of Java 7.0 in CICS TS V5.1, see 5.8, “Java 7 and zEnterprise EC12” on page 62.

5.3 Improvements in threadsafety

Most new CICS API and SPI commands in CICS V5.1 are threadsafe. Also, some commands were made threadsafe in this release. Specific functional areas also were improved to reduce task control block (TCB) switches.

5.3.1 Threadsafe API and SPI commands

The following new CICS API commands are threadsafe:

- ▶ GETMAIN64
- ▶ FREEMAIN64
- ▶ GET64 CONTAINER
- ▶ PUT64 CONTAINER

The following CICS API transient data commands were made threadsafe:

- ▶ READQ TD
- ▶ WRITEQ TD
- ▶ DELETEQ TD

These transient data API commands are threadsafe when used with a queue in a local CICS region, and when the request is function shipped to a remote CICS region over an Internet Protocol interconnectivity (IPIC) connection only. For other types of connections to remote CICS regions, the command is not threadsafe.

For more information about CICS API commands, see the topic “CICS command summary” in IBM Knowledge Center at this website:

<https://ibm.biz/Bdi68Q>

The following new CICS system programming interface (SPI) commands are threadsafe:

- ▶ INQUIRE EPADAPTINSET
- ▶ EPADAPTERSET commands:
 - INQUIRE EPADAPTERSET
 - SET EPADAPTERSET

The following CICS SPI commands were made threadsafe:

- ▶ SET TASK
- ▶ TRACEDEST commands:
 - INQUIRE TRACEDEST
 - SET TRACEDEST
- ▶ TRACEFLAG commands:
 - INQUIRE TRACEFLAG
 - SET TRACEFLAG

- ▶ TRACETYPE commands:
 - INQUIRE TRACETYPE
 - SET TRACETYPE

For more information about CICS SPI commands, see the topic “System commands” in IBM Knowledge Center at this website:

<https://ibm.biz/Bd4Yx3>

5.3.2 Threadsafe program loading

When running on an open TCB and a CICS program load is requested, there is no longer a TCB change mode to the resource-owning (RO) TCB. The RO TCB is still used when an application is not running on an open TCB, or for CICS DFHRPL and LIBRARY data set management operations.

The ability to load programs on an open TCB can result in path length reductions for applications that frequently issue LOAD requests because of the removal of the TCB switch. The use of an open TCB can also result in reduced contention for the single RO TCB and potentially offers significantly increased CICS program LOAD capability. Reducing the effect of program load on the RO TCB also reduces any contention on the RO TCB that is caused by competing security calls that also share the RO TCB.

A simple threadsafe application was created that issued an EXEC CICS LOAD command while running on an open TCB. The chart that is shown in Figure 5-5 plots the response time against the program LOAD request rate.

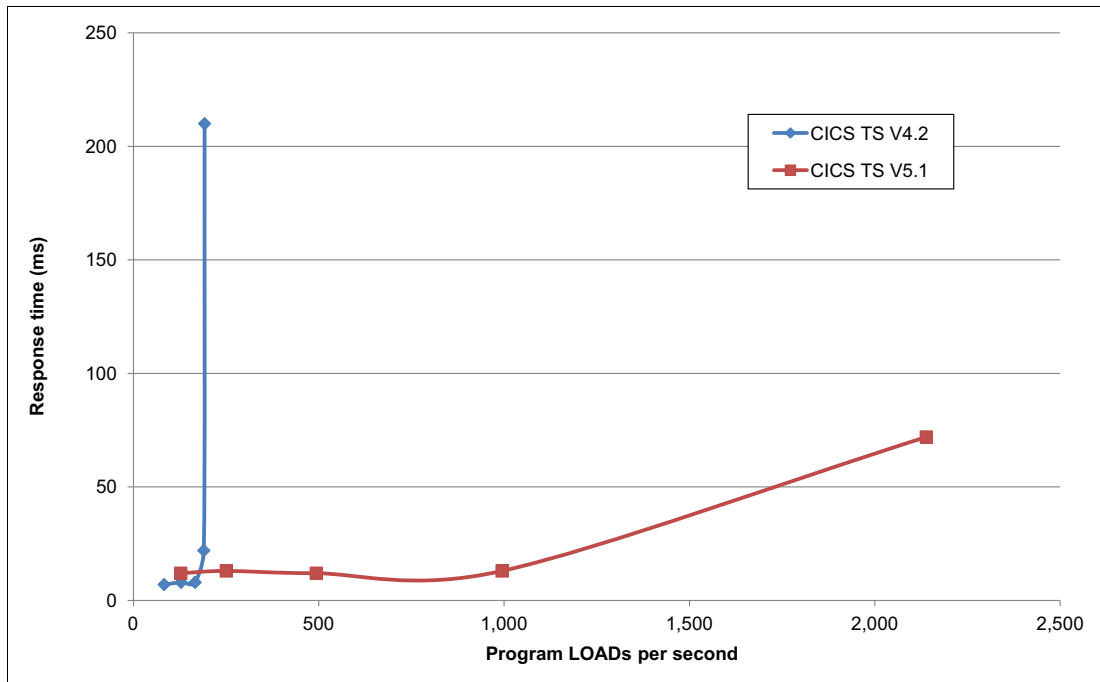


Figure 5-5 Response time against program LOAD rate

With CICS TS V4.2, CICS must switch to the RO TCB to complete the physical load when the application issues an EXEC CICS LOAD command even though the application is running on an open TCB.

In this environment, a load operation is taking an average of 5 ms elapsed time to complete. Although the command does not use 5 ms of CPU time during this period, it does mean that the RO TCB is dispatched for a total of 5 ms and so the LOAD operations are effectively run serially. At a rate of around 200 requests per second, the RO TCB reached its limit in terms of available dispatch time.

When the same application is run in CICS TS V5.1, the LOADs are run concurrently on open TCBs. The chart in Figure 5-5 on page 49 shows that the throughput capability increased approximately tenfold. The limiting factor for this application became the I/O subsystem in CICS TS V5.1 instead of the RO TCB.

Updates to the CICS monitoring and statistics data also are associated with the threadsafe program load enhancements. For more information about updates to CICS monitoring, see 5.5.8, “The DFHTASK performance group” on page 57. For more information about for updates to CICS statistics, see 5.5.11, “Loader domain global statistics” on page 60.

5.3.3 Use of T8 TCB for JDBC calls

A JDBC call from a Java application that uses the type 2 JDBC driver in CICS no longer requires a TCB change mode operation. As described in Chapter 4, “Open transaction environment” on page 31, TCB change mode operations can add CPU overhead to an application. In a CICS JVM server, Java applications run on T8 TCBs. In CICS TS V4.2, DB2 calls from a Java environment required a switch to an L8 TCB. In CICS TS V5.1, this switch is removed.

To demonstrate the improvement, a modified version of the CICS DB2 Dynamic SQL example was used. The application reads 43 rows from a DB2 table and writes the results to a CICS terminal. This combination of DB2 accesses and terminal writes ensures that the application has a mix of JDBC and JCICS calls.

The application was tested by using a single CICS region with one JVM server defined. Both configurations used DB2 V10. The JVM server was defined with a limit of 20 concurrent threads and the following JVM parameters:

- ▶ -Xgcpolicy:gencon
- ▶ -Xmx600M
- ▶ -Xms600M
- ▶ -Xmnx500M
- ▶ -Xmns500M
- ▶ -Xmox100M
- ▶ -Xmos100M

CICS TS V4.2 used Java V6.0.1 SR3, and CICS TS V5.1 used Java V7.0 SR3.

By using the methodology that is described in 2.6, “Collecting performance data” on page 18, CPU usage was recorded by using RMF for five different transaction rates. Figure 5-6 shows the measured CPU utilization as the transaction rate is increased.

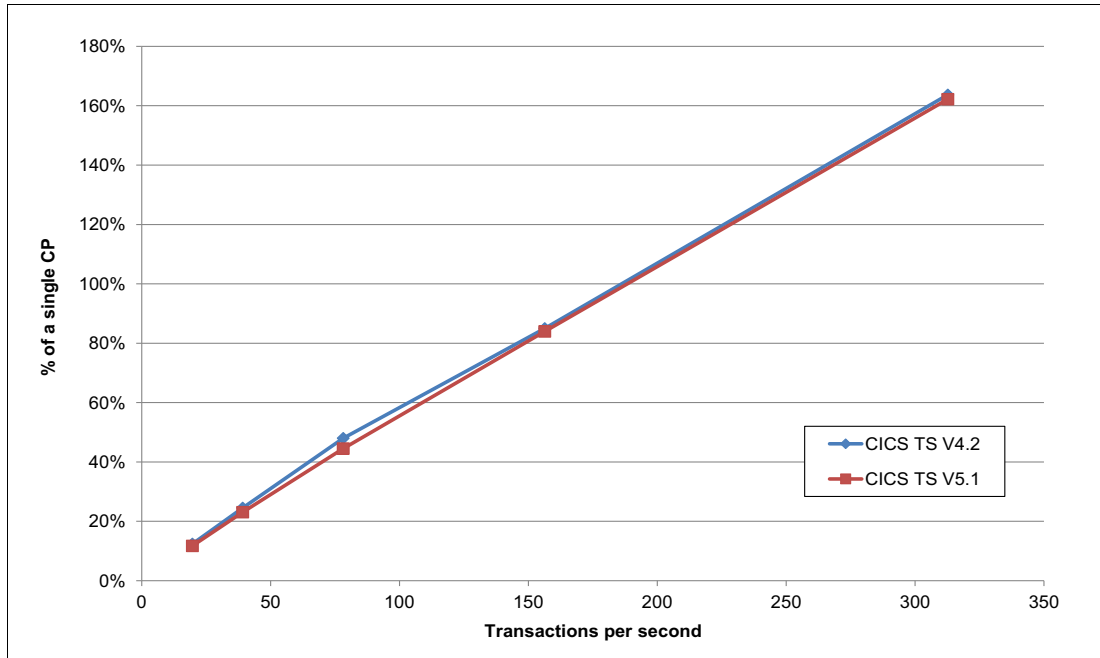


Figure 5-6 Plot of CPU utilization against transaction rate for a Java workload in CICS V4.2 and V5.1

The CICS TS V5.1 configuration uses slightly less CPU overall, which benefits from the reduction in TCB change mode operations. Both configurations scale well, as indicated by the linear increase in CPU utilization as the transaction rate increased.

By using the same JDBC application, a second benchmark was run and CICS monitoring facility (CMF) data was collected. Several key performance metrics were extracted from the CMF data and are listed in Table 5-7.

Table 5-7 Average CPU time and TCB breakdown for Java DB2 workload

CICS release	Average CPU time (ms)				Average TCB change mode count
	Total user	QR TCB	T8 TCB	L8 TCB	
V4.2	4.374	0.310	2.907	1.157	300
V5.1	4.230	0.322	3.844	0.064	202

Table 5-7 shows that the number of TCB change mode operations for each transaction decreased because the workload now completes JDBC calls into DB2 on a T8 TCB, rather than an L8 TCB. This reduction in TCB change modes has the following effects:

- ▶ The overall CPU per transaction is reduced from 4.374 ms to 4.230 ms.
- ▶ Where CPU time for DB2 calls was previously accumulated on the L8 TCB, this time is now accumulated on the T8 TCB. A small amount of CPU time is still accumulated on an L8 TCB for sync point processing at transaction completion.

5.4 Changes to system initialization parameters

Several performance-related CICS system initialization table (SIT) parameters were changed in the CICS TS V5.1 release. This section describes changes to the SIT parameters that have the most affect on CICS performance. All comparisons to previous limits or default values refer to CICS TS V4.2.

5.4.1 Active keypoint frequency (AKPFREQ)

The minimum value for the AKPFREQ parameter was decreased from 200 to 50. This value means that completed log task records can be deleted more frequently, which reduces the DASD data space usage. The value that is specified for the AKPFREQ parameter can be zero, or 50 - 65535.

If you specify AKPFREQ=0, no activity keypoints are written. Therefore, replication support is affected because without activity keypointing, tie-up records are not written to replication logs.

For more information, see the topic “AKPFREQ” in IBM Knowledge Center at this website:

<https://ibm.biz/Bd4YDc>

5.4.2 Extended dynamic storage area limit (EDSALIM)

The default value for the EDSALIM parameter was increased from 48 MB to 800 MB. This new default value enables a CICS region that was started with the default value to process a reasonable workload. The value that is specified for the EDSALIM parameter can be 48 MB - 2047 MB in multiples of 1 MB.

For more information, see the topic “EDSALIM” in IBM Knowledge Center at this website:

<https://ibm.biz/Bd4YDx>

Note: The minimum value for the EDSALIM parameter was increased to 64 MB in CICS TS V5.4.

5.4.3 Terminal scan delay (ICVTSD)

The default value for the ICVTSD parameter was decreased from 500 to 0. The value that is specified for the ICVTSD parameter can be 0 - 5000 milliseconds.

The terminal scan delay facility was used in earlier releases to limit how quickly CICS dealt with some types of terminal output requests that were made by applications to spread the overhead of dealing with the requests. Specifying a nonzero value was sometimes appropriate where the CICS system used non-SNA networks. However, with SNA and IPIC networks, setting ICVTSD to 0 is appropriate to provide a better response time and the best virtual storage usage.

For more information, see the topic “ICVTSD” in IBM Knowledge Center at this website:

<https://ibm.biz/Bd4YDD>

5.4.4 Maximum open TCBs (MAXOPENTCBS)

The MAXOPENTCBS parameter was used to specify the maximum number of open TCBs in the pool of L8 and L9 mode TCBs. These TCBs are used for OPENAPI application programs and task-related user exits that are enabled with the OPENAPI option.

CICS now manages the number of TCBs in this pool automatically by using the following formula based on the current value of the maximum tasks (MXT) system parameter:

$$\text{MAXOPENTCBS} = (2 \times \text{MXT}) + 32$$

For more information about open TCBs, see Chapter 4, “Open transaction environment” on page 31.

Note: The MAXOPENTCBS parameter was reintroduced in CICS TS V5.2. The minimum value was increased to 32 in CICS TS V5.4.

5.4.5 Maximum XP TCBs (MAXXPTCBS)

MAXXPTCBS was used to specify the maximum number of open TCBs in the pool of X8 and X9 mode TCBs. These TCBs are used for C and C++ programs compiled with the XPLINK option.

CICS now manages the number of TCBs in this pool automatically by using the following formula based on the current value of the maximum tasks (MXT) system parameter:

$$\text{MAXXPTCBS} = \text{MXT}$$

For more information about open TCBs, see Chapter 4, “Open transaction environment” on page 31.

Note: The MAXXPTCBS parameter was reintroduced in CICS TS V5.2.

5.4.6 Maximum tasks (MXT)

The maximum number of user tasks that can exist in a CICS region concurrently was increased in the CICS TS V5.1 release. The maximum value that can be specified for the MXT parameter was increased from 999 to 2000. The minimum value was increased from 1 to 10, and the default value was increased from 5 to 500.

The changes mean that a CICS region operates more efficiently with the default setting and can process more workload, so the need to increase the number of CICS regions is reduced.

Note: The default value for MXT was changed to 250 in CICS V5.2.

These changes apply to the MXT system initialization parameter, the MAXTASKS option of the SET SYSTEM and CEMT SET SYSTEM commands, and the MAXTASKS value in CICSplex SM.

You must ensure that enough storage is available to support the maximum number of tasks value. For more information about setting the maximum task specification, see the “Setting the maximum task specification (MXT)” topic in IBM Knowledge Center at this website:

<https://ibm.biz/Bd4YDM>

When you increase the maximum number of tasks for a CICS region, measure performance to ensure that the response time and other time components (such as dispatch time and suspend time) for your transactions remain acceptable. In some systems, an increase in concurrent tasks might increase resource contention to a level that causes more delays for transactions.

In the performance class data for a transaction, the new MAXTASKS field records the current setting for the maximum number of tasks for the CICS region. The CURTASKS field records the current number of active user transactions in the system at the time the user task was attached. This data helps you to assess the relationship between the task load during the life of a transaction, and the performance of the transaction. For more information about these new performance class data fields, see 5.5, “Enhanced instrumentation” on page 54.

5.4.7 Priority aging interval (PRTYAGE)

The default value for the PRTYAGE parameter was decreased from 32768 (32.768 seconds) to 1000 (1 second). This lower value means that the priority of long-running tasks that are on the ready queue increases more rapidly.

For more information, see the topic “PRTYAGE” in IBM Knowledge Center at this website:

<https://ibm.biz/Bd4YDa>

5.4.8 Location of terminal user areas (TCTUALOC)

The default value of the TCTUALOC parameter was changed from BELOW to ANY. The specification of TCTUALOC=ANY means that terminal user areas can be stored in 24-bit or 31-bit storage, and CICS uses 31-bit storage to store them if possible.

If you require the terminal user area to be in 24-bit storage because you have application programs that are not capable of 31-bit addressing, specify the system initialization parameter TCTUALOC=BELOW for the CICS region.

For more information, see the topic “TCTUALOC” in IBM Knowledge Center at this website:

<https://ibm.biz/Bd4YDe>

5.5 Enhanced instrumentation

Significant enhancements to monitoring and statistics were made in the CICS TS V5.1 release. This section details the extra and changed fields that are now available in the CMF and statistics SMF records.

5.5.1 The DFHCHNL performance group

The following fields were updated to also include request counts from the new EXEC CICS GET64 CONTAINER and EXEC CICS PUT64 CONTAINER API commands:

- ▶ PGGETCCT
- ▶ PGPUTCCT
- ▶ PGMOVCCT
- ▶ PGGETCDL
- ▶ PGPUTCDL
- ▶ PGCRECCT

For more information about counters that are available in the DFHCHNL performance group, see the topic “Performance data in group DFHCHNL” in IBM Knowledge Center at this website:

<https://ibm.biz/Bd4YDp>

5.5.2 The DFHCICS performance group

One new field was added to the DFHCICS performance group.

- ▶ Managed Platform - policy rule thresholds exceeded (field MPPRTXCD)
The number of policy rule thresholds that this task exceeded.

For more information about counters that are available in the DFHCICS performance group, see the topic “Performance data in group DFHCICS” in IBM Knowledge Center at this website:

<https://ibm.biz/Bd4YDV>

5.5.3 The DFHDEST performance group

The following new fields were added to the DFHDEST performance group:

- ▶ Transient Data intrapartition lock wait time (field TDILWTT)
The elapsed time for which the user task waited for an intrapartition transient data lock.
- ▶ Transient Data extrapartition lock wait time (field TDELWTT)
The elapsed time for which the user task waited for an extrapartition transient data lock.

For more information about counters that are available in the DFHDEST performance group, see the topic “Performance data in group DFHDEST” in IBM Knowledge Center at this website:

<https://ibm.biz/Bd4YDA>

5.5.4 The DFHFILE performance group

The following new fields were added to the DFHFILE performance group:

- ▶ File control wait time for exclusive control of a VSAM control interval (field FCXCWTT)
The elapsed time in which the user task waited for exclusive control of a VSAM control interval.
- ▶ File control wait time for a VSAM string (field FCVSWTT)
The elapsed time in which the user task waited for a VSAM string.

For more information about counters that are available in the DFHFILE performance group, see the topic “Performance data in group DFHFILE” in IBM Knowledge Center at this website:

<https://ibm.biz/Bd4YDu>

5.5.5 The DFHRMI performance group

The DFHRMI group is present in the performance class record only if RMI=YES is specified on the DFHMCT TYPE=INITIAL macro.

In CICS TS V5.1, the default value for the RMI parameter changed from NO to YES.

5.5.6 The DFHSOCK performance group

The following new fields were added to the DFHSOCK performance group:

- ▶ IPIC session allocation wait time (field ISALWTT)
The elapsed time for which a user task waited for an allocate request for an IPIC session.
- ▶ Cipher selected (field SOCIPHER)
Identifies the code for the cipher suite that was selected during the SSL handshake for use on the inbound connection.

For more information about counters that are available in the DFHSOCK performance group, see the topic “Performance data in group DFHSOCK” in IBM Knowledge Center at this website:

<https://ibm.biz/Bd4YD3>

5.5.7 The DFHSTOR performance group

The following new fields were added to the DFHSTOR performance group:

- ▶ Number of GCDSA storage getmains (field SC64CGCT)
Number of user-storage GETMAIN requests that are issued by the user task for storage above the bar in the CICS dynamic storage area (GCDSA).
- ▶ GCDSA storage high water mark above 2 GB (field SC64CHWM)
Maximum amount (high-water mark) of user storage, rounded up to the next 4 KB, allocated to the user task above the bar in the CICS dynamic storage area (GCDSA).
- ▶ Number of GUDSA storage getmains (field SC64UGCT)
Number of user-storage GETMAIN requests that are issued by the user task for storage above the bar in the user dynamic storage area (GUDSA).
- ▶ GUDSA storage high water mark above 2 GB (field SC64UHWM)
Maximum amount (high-water mark) of user storage, rounded up to the next 4 KB, allocated to the user task above the bar in the user dynamic storage area (GUDSA).
- ▶ Number of shared storage getmains above 2 GB (field SC64SGCT)
Number of storage GETMAIN requests that are issued by the user task for shared storage above the bar in the GCDSA or GSDSA.
- ▶ Shared storage bytes obtained (field SC64GSHR)
Amount of shared storage obtained by the user task by using a GETMAIN request above the bar in the GCDSA or GSDSA. The total number of bytes that are obtained is rounded up to the next 4 KB and the resulting number of 4 KB pages is reported.
- ▶ Shared storage bytes released (field SC64FSHR)
Amount of shared storage that is released by the user task by using a FREEMAIN request above the bar in the GCDSA or GSDSA. The total number of bytes that are obtained is rounded up to the next 4 KB and the resulting number of 4 KB pages is displayed.

For more information about counters that are available in the DFHSTOR performance group, see the topic “Performance data in group DFHSTOR” in IBM Knowledge Center at this website:

<https://ibm.biz/Bd4YDk>

5.5.8 The DFHTASK performance group

The following new fields were added to the DFHTASK performance group that are related to the CICS RO, SO, and T8 mode TCBs:

- ▶ User task RO TCB dispatch time (field RODISPT)
The elapsed time during which the user task was dispatched by the CICS dispatcher on the CICS RO mode TCB.
- ▶ User task RO TCB CPU time (field ROCPUT)
The processor time during which the user task was dispatched by the CICS dispatcher on the CICS RO mode TCB.
- ▶ User task RO TCB wait-for-dispatch time (field ROMODDLY)
The elapsed time for which the user task waited for redispach on the CICS RO TCB. This time is the aggregate of the wait times between each event completion and user-task redispach.
- ▶ User task SO TCB wait-for-dispatch (field SOMODDLY)
The elapsed time for which the user task waited for redispach on the CICS SO TCB. This time is the aggregate of the wait times between each event completion and user-task redispach.
- ▶ JVM server thread TCB delay time (field MAXTTDLY)
The elapsed time for which the user task waited to obtain a T8 TCB because the CICS system reached the limit of available threads.

The RO mode TCB is used for loading programs, unless the API command to load the program (EXEC CICS LOAD, EXEC CICS XCTL, or EXEC CICS LINK) is issued by an application that is running on an open TCB. In that situation, the open TCB is used to load the program instead of the RO TCB. The CICS RO mode TCB is also used for opening and closing CICS data sets, issuing IBM RACF® calls, and similar tasks.

The SO mode TCB is used to make calls to the socket interface of TCP/IP.

The T8 mode open TCBs are used by a JVM server to perform multi-threaded processing. Each T8 TCB runs under one thread. The thread limit is 2,000 for each CICS region, and each JVM server in a CICS region can have up to 256 threads.

The following new fields were added to the DFHTASK performance group that are related to the hardware environment on which the CICS region is running:

- ▶ CEC machine type (field CECMCHTP)
The central electronics complex (CEC) machine type, in EBCDIC, for the physical hardware environment where the CICS region is running. CEC is a commonly used synonym for central processing complex (CPC).
- ▶ CEC model number (field CECMDLID)
The CEC model number, in EBCDIC, for the physical hardware environment where the CICS region is running.

The following new fields were added to the DFHTASK performance group that are related to region load status:

- ▶ Maximum tasks value (field MAXTASKS)
The maximum task limit (MXT), expressed as a number of tasks, for the CICS region at the time the user task was attached.

- ▶ Current tasks in CICS region (field CURTASKS)

The current number of active user transactions in the system at the time the user task was attached.

The following new fields were added to the DFHTASK performance group that are related to specialty processor offload rates:

- ▶ Processor time on a standard processor (field CPUTONCP)

The total task processor time on a standard processor for which the user task was dispatched on each CICS TCB under which the task ran.

- ▶ Processor time eligible for offload to a specialty processor (field OFFLCPUT)

The total task processor time that was spent on a standard processor, but was eligible for offload to a specialty processor (zIIP or zAAP).

The following derived metrics can be obtained by combining the USRCPUT field with the CPUTONCP and OFFLCPUT fields:

- ▶ Total CPU time on specialty processor:

$USRCPUT - CPUTONCP$

- ▶ Total CPU time on standard processor that was not offload-eligible:

$CPUTONCP - OFFLCPUT$

- ▶ Total CPU time that was offload eligible:

$OFFLCPUT + USRCPUT - CPUTONCP$

Note: The times that are shown in the CPUTONCP and OFFLCPUT fields are available only when running on a system that supports the Extract CPU Time instruction service that is available on IBM System z9® or later hardware. For z/OS V1R13, the PTF for APAR OA38409 must also be applied.

The following new fields were added to the DFHTASK performance group that are related to CICS application context support:

- ▶ Application name (field ACAPPLNM)

The 64-character name of the application in the application context data.

- ▶ Platform name (field ACPLATNM)

The 64-character name of the platform in the application context data.

- ▶ Application major version (field ACMAJVER)

The major version of the application in the application context data, expressed as a 4-byte binary value.

- ▶ Application minor version (field ACMINVER)

The minor version of the application in the application context data, expressed as a 4-byte binary value.

- ▶ Application micro version (field ACMICVER)

The micro version of the application in the application context data, expressed as a 4-byte binary value.

- ▶ Operation name (field ACOPERNM)

The 64-character name of the operation in the application context data.

For more information about counters that are available in the DFHTASK performance group, see the topic “Performance data in group DFHTASK” in IBM Knowledge Center at this website:

<https://ibm.biz/Bd4YRP>

5.5.9 The DFHTERM performance group

The MRO, LU6.1, and LU6.2 session allocation wait time (field TCALWTT) field was added to the DFHTERM performance group. This field is the elapsed time for which a user task waited for an allocate request for a multiregion operation (MRO), LU6.1, or LU6.2 session.

For more information about counters that are available in the DFHTERM performance group, see the topic “Performance data in group DFHTERM” in IBM Knowledge Center at this website:

<https://ibm.biz/Bd4YRm>

5.5.10 Monitoring domain global statistics

The following new fields were added to the collected monitoring domain global statistics:

- ▶ CEC Machine Type and Model Number (fields MNGMCHTP and MNGMDLID)
The CEC machine type and model number for the physical hardware environment where the CICS region is running.
- ▶ WLM Address Space Goal Management (field MNGWLMGM)
Whether z/OS Workload Manager manages the CICS address space by using region goals, transaction goals, or both.

A fragment of a sample DFHSTUP report containing the new fields is shown in Example 5-1.

Example 5-1 Fragment of a sample monitoring statistics report produced by CICS TS V5.1 DFHSTUP

```
CEC Machine Type and Model Number . . . : 2817-779
Exception records . . . . . : 0
Exception records suppressed. . . . . : 0
Performance records . . . . . : 0
Performance records suppressed. . . . . : 0
...
MVS WLM Mode. . . . . : Goal
MVS WLM Server. . . . . : Yes
MVS WLM Workload Name . . . . . : CICS CPU
MVS WLM Service Class . . . . . : CICS BTCH
MVS WLM Report Class. . . . . : CICS2A31
MVS WLM Resource Group. . . . . :
WLM Manage Regions Using Goals of . . . : Transaction
MVS WLM Goal Type . . . . . : Velocity
MVS WLM Goal Value. . . . . : 80
MVS WLM Goal Importance . . . . . : 1
MVS WLM CPU Critical. . . . . : No
MVS WLM Storage Critical. . . . . : No
```

For more information about monitoring domain statistics, see the topic “Monitoring domain: global statistics” in IBM Knowledge Center at this website:

<https://ibm.biz/Bd4YRG>

5.5.11 Loader domain global statistics

The following new fields were added to the collected loader domain global statistics:

- ▶ Library load requests on the RO TCB (field LDGLLR0)
The number of times that the loader issued a program load request that used the RO TCB. This value is a subset of the number of library loads shown by LDGLLR. To calculate the number of program load requests that ran on open TCBs, subtract this value from the value shown by LDGLLR.
- ▶ Total loading time on the RO TCB (field LDGLLTR0)
The time taken for the number of library loads shown by LDGLLR0. This value is a subset of the time shown by LDGLLT. To calculate the time taken for program load requests that ran on open TCBs, subtract this value from the value shown by LDGLLT.

A fragment of a sample DFHSTUP report that contains the new fields is shown in Example 5-2. The Average loading time on the RO TCB field is calculated by the DFHSTUP program and is not included directly in the SMF data.

Example 5-2 Sample CICS TS V5.1 DFHSTUP loader domain global statistics report fragment

LIBRARY load requests	:	0
LIBRARY load requests on the RO TCB	:	0
Total loading time.	:	00:00:00.0000
Total loading time on the RO TCB.	:	00:00:00.0000
Average loading time.	:	00:00.000000
Average loading time on the RO TCB.	:	00:00.000000

For more information about loader domain global statistics, see the topic “Loader domain: Global statistics” in IBM Knowledge Center at this website:

<https://ibm.biz/Bd4YRb>

5.6 Virtual storage constraint relief

Virtual storage constraint relief (VSCR) is the reduction of virtual storage usage, which helps avoid short-on-storage conditions and can reduce the need for more CICS regions. CICS TS V5.1 provides VSCR in a number of areas, which reduces pressure on 24-bit and 31-bit virtual storage.

This section describes virtual storage improvements for the following areas:

- ▶ 24-bit virtual (*below the line*)
- ▶ 31-bit virtual (*above the line but below the bar*)
- ▶ 64-bit virtual (*above the bar*)

5.6.1 24-bit storage

The following CICS infrastructure items now use 31-bit storage in place of all, or some, of the 24-bit storage that was used in previous releases:

- ▶ Sync point and back out processing.
- ▶ Processing for transient data **EXEC CICS** application programming commands, wherever possible.

- ▶ CICS execution diagnostic facility (CEDF).
- ▶ CICS command-language tables for command interpreter (CECI) and other functions.
- ▶ Processing for journaling **EXEC CICS** application programming commands.
- ▶ Processing for function-shipped DL/I calls.
- ▶ Mirror transactions. For more information, see 5.6.4, “Mirror transactions” on page 61.
- ▶ The COMMAREA on an **EXEC CICS XCTL** call. For more information, see 5.6.5, “XCTL with a communication area” on page 61.

For more information about items that are improved in the CICS TS V5.1, see the topic “Reduced use of 24-bit storage by CICS” in IBM Knowledge Center at this website:

<https://ibm.biz/Bd4YRL>

5.6.2 31-bit storage

The following CICS infrastructure items now use 64-bit storage in place of all, or some, of the 31-bit storage that was used in previous releases:

- ▶ Console queue processing
- ▶ Storage allocation control blocks
- ▶ Loader control blocks

For more information about changes in the use of 31-bit storage, see the topic “Changes in CICS storage use from 31-bit to 64-bit storage” in IBM Knowledge Center at this website:

<https://ibm.biz/Bd4YR9>

5.6.3 64-bit storage

In CICS TS V5.1, the new *managed platform* and *application context* CICS facilities use 64-bit virtual storage.

User applications can now directly access 64-bit storage. For more information, see 5.7, “64-bit application support” on page 62.

5.6.4 Mirror transactions

The mirror transactions (CEHP, CEHS, CPMI, CSHR, CSMI, CSM1, CSM2, CSM3, CSM5, and CVMI) are now defined as TASKDATALOC (ANY).

5.6.5 XCTL with a communication area

When a communication area (COMMAREA) is used with an **EXEC CICS XCTL** command, CICS can now create the COMMAREA in 31-bit storage, rather than 24-bit storage, where appropriate.

When **EXEC CICS XCTL** is used, CICS ensures that any COMMAREA is addressable by the program that receives it by creating the COMMAREA in an area that conforms to the addressing mode of the receiving program. If the receiver is `AMODE(24)`, the COMMAREA is created in 24-bit storage. If the receiver is `AMODE(31)`, the COMMAREA is created in 31-bit storage.

In earlier releases of CICS, the COMMAREA was always copied into 24-bit storage.

5.6.6 User exit global work area

The **EXEC CICS ENABLE PROGRAM** system programming command has the new attribute **GALLOCATION** that specifies the location of the storage that CICS provides as a global work area for this exit program. The **GALLOCATION** attribute can have one of the following values:

- ▶ **LOC24**

The global work area is in 24-bit storage. This location is the default location.

- ▶ **LOC31**

The global work area is in 31-bit storage.

For more information about the **EXEC CICS ENABLE PROGRAM** system programming command, see the “ENABLE PROGRAM” topic in IBM Knowledge Center at this website:

<https://ibm.biz/Bd4zLc>

5.6.7 Related APARs

The IBM Language Environment® APAR PM57053 reduces the amount of virtual storage that is consumed by applications in 24-bit storage. For more information, see “PM57053: CEECPINI GETS TOO MUCH BELOW-THE-LINE STORAGE IN CICS” in IBM Support Portal at this website:

<http://www.ibm.com/support/docview.wss?uid=isg1PM57053>

5.7 64-bit application support

CICS TS V5.1 supports non-Language Environment assembly language programs that run in 64-bit addressing mode, which provides 64-bit application support to access large data objects.

New API commands, a new CICS-supplied procedure, and new CICS executable modules are supplied to provide 64-bit application support. CICS storage manager, program manager, loader domain, CICS-supplied macros, and the **CECI** and **CEDF** transactions are changed to provide 64-bit application support. New dynamic storage areas (DSAs) are available in 64-bit storage.

For more information about developing 64-bit assembly language programs, see the “Developing AMODE(64) assembler language programs” topic in IBM Knowledge Center at this website:

<https://ibm.biz/Bd4YFy>

5.8 Java 7 and zEnterprise EC12

As described in 5.2.4, “Java throughput” on page 47, CICS TS V4.2 supports Java V6.0.1 only, and CICS TS V5.1 supports Java 7. The IBM Java 7 SDK for z/OS provides greater exploitation of the IBM zEnterprise EC12 hardware than previous releases of the SDK. The improved hardware exploitation by the Java virtual machine (JVM) can provide significant performance improvements for CICS workloads that use Java.

Note: APAR PI30532 enables support for Java 7.1 in CICS TS V5.1. APAR PI52819 enables support for Java 8 in CICS TS V5.1.

This section compares the internal throughput rate (ITR) of a Java workload when running in various combinations of hardware and software configurations. For more information about ITR, see 1.3.2, “Internal throughput rate” on page 7.

The workload was an intensive Java application that processes an inbound web service by using the Java pipeline implementation.

The IBM zEnterprise 196 (z196) was a model M80 that was running on an LPAR configured with four dedicated CPs, which resulted in an LSPR processor equivalent of 2817-704.

The IBM zEnterprise EC12 (zEC12) was a model HA1, running on an LPAR configured with four dedicated CPs, which resulted in an LSPR processor equivalent of 2827-704.

Both configurations used z/OS V1R13. The following hardware and software configuration combinations were studied:

- ▶ CICS TS V4.2 using Java V6.0.1 SR3 on a z196
- ▶ CICS TS V5.1 using Java V7.0 SR3 on an z196
- ▶ CICS TS V5.1 using Java V7.0 SR3 on a zEC12
- ▶ CICS TS V5.1 using Java V7.0 SR3 with aggressive hardware exploitation on a zEC12

Aggressive hardware exploitation was enabled in Java V7.0 by using the `-Xaggressive` and the `-Xjit:noResumableExceptionHandler` runtime options.

Figure 5-7 shows the relative ITR values that was recorded for these configurations.

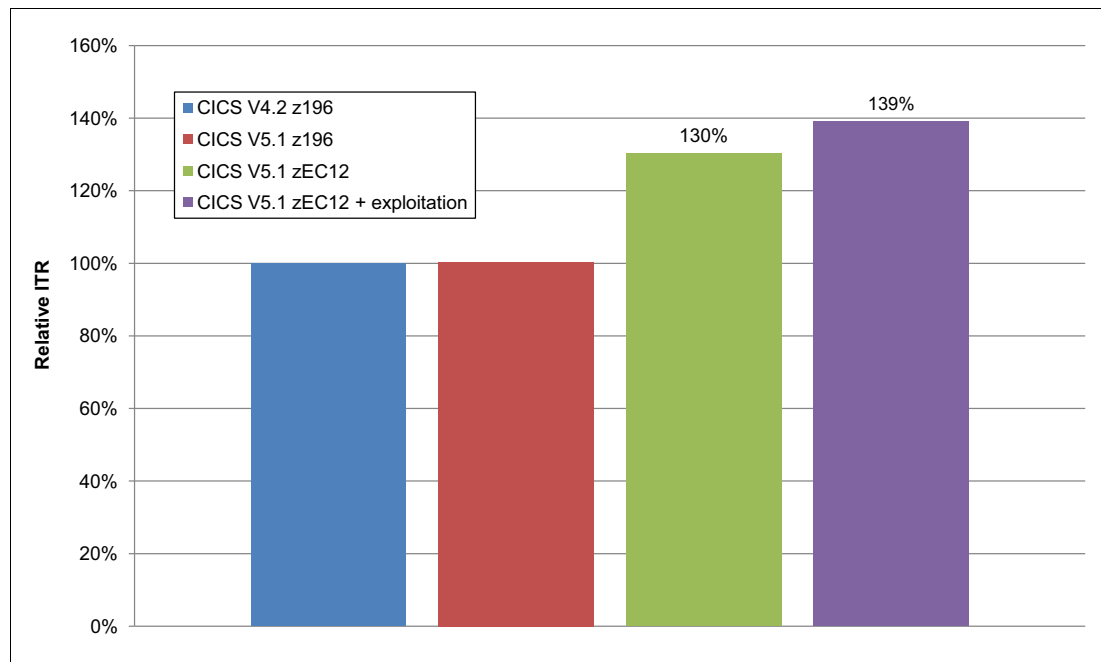


Figure 5-7 Comparison of hardware and software configurations for a Java workload

As shown in Figure 5-7, the use of Java V7.0 in CICS TS V5.1 provided equivalent throughput when compared with Java V6.0.1 in CICS TS V4.2.

Upgrading the hardware to a zEC12 improved ITR by 30% and enabled the `-Xaggressive` option that further increased this value to 39%.

Another Java workload was also tested that had only a small amount of Java logic. This simple workload demonstrated a 24% improvement when moving from a z196 to a zEC12. The use of the `-Xaggressive` JVM option increased this improvement slightly to 25%. This improvement of 25% when moving from a z196 to a zEC12 is in line with the LSPR expectations.

For more information about interpreting LSPR tables, see 1.3, “Large Systems Performance Reference” on page 6. The LSPR tables used can be found in IBM Resource Link® at this website:

<https://www.ibm.com/servers/resourceLink/lib03060.nsf/pages/lSprITRz0Sv1r13>

From the LSPR tables, the relative ITR values were seen when running a workload with an average relative nest intensity:

- ▶ 2817-704 = 7.72
- ▶ 2827-704 = 9.66

Therefore, the expected ITR improvement equals $9.66 \div 7.72 = 1.25$, or an increase of 25%.

5.9 CICSplex System Manager dynamic routing

By using a variant of the Data Systems Workflow (DSW) workload as described in 3.2, “Data Systems Workload” on page 22, the cost per transaction was measured to understand the overhead that was involved with the use of dynamic transaction routing by using CICSplex System Manager (CICSplex SM). The following scenarios were used to provide the performance results:

- ▶ Connections from 2500 simulated LU2 clients were processed directly by each of the AORs with no TORs involved.
- ▶ A single TOR routed transaction to the AORs by using a simple *round robin* routing algorithm.
- ▶ Transactions were routed to the AORs with CICSplex SM dynamic routing by using the CICSplex SM sysplex optimized routing algorithm.

In all cases, four AORs were used and accessed the data files by using VSAM RLS. TORs were connected to AORs through MRO/XM connections. Eight CPs were online during the measurements, which provided an LSPR processor equivalent of 2817-708. The results of the performance study are listed in Table 5-8, Table 5-9 on page 65, and Table 5-10 on page 65. Table 5-8 presents the scenario in which no TOR was used; therefore, the TOR ETR and TOR CPU columns are not applicable.

Table 5-8 Performance results with no transaction routing

TOR ETR	TOR CPU	AOR ETR	AOR CPU	CPU per transaction (ms)
n/a	n/a	2072.61	178.94%	0.863
n/a	n/a	2842.46	230.46%	0.810
n/a	n/a	4120.62	324.47%	0.787
n/a	n/a	5035.52	387.15%	0.768
n/a	n/a	5617.90	427.88%	0.761

For this scenario, the average CPU cost per transaction was 0.797 ms.

Table 5-9 presents the scenario where a simple round-robin routing algorithm is configured by specifying the SIT parameter DTRPGM. This table demonstrates the migration costs when moving from single local access to MRO transaction routing.

Table 5-9 Performance results with TOR using a roundrobin algorithm

TOR ETR	TOR CPU	AOR ETR	AOR CPU	CPU per transaction (ms)
1982.71	21.71%	2072.91	179.85%	0.976
2716.34	28.32%	2841.31	229.76%	0.912
3947.17	40.11%	4127.12	320.02%	0.876
4782.32	49.27%	5002.40	380.17%	0.862
5357.32	56.98%	5602.40	414.15%	0.843

The TOR and AOR transaction rates are shown separately because some of the transactions that are routed from the TOR to the AOR issue a local **EXEC CICS START** command for more transactions when running in the AORs.

For this second scenario, the average CPU cost per transaction was 0.893 ms.

Table 5-10 presents the scenario in which dynamic transaction routing uses CICSplex SM sysplex that is optimized workload routing.

Table 5-10 Performance results with TOR using CICSplex SM dynamic transaction routing

TOR ETR	TOR CPU	AOR ETR	AOR CPU	CPU per transaction (ms)
1982.55	26.31%	2071.68	178.87%	0.999
2716.04	34.55%	2840.08	229.68%	0.935
3946.17	48.99%	4125.28	321.11%	0.902
4813.04	59.46%	5033.92	380.43%	0.878
5394.49	67.92%	5640.63	418.77%	0.867

As in Table 5-9 on page 65, the TOR and AOR transaction rates are shown separately to differentiate where local **EXEC CICS START** commands were issued.

For this final scenario, the average CPU cost per transaction was 0.916 ms.

Comparing scenarios one and two represents the CPU cost that was incurred by introducing transaction routing into the environment. The average CPU per transaction increased by approximately 0.1 ms.

By using CICSplex SM Sysplex Optimized Routing, the average cost per transaction increases only slightly (0.026 ms). For this workload, the transaction rate and loading on individual regions is stable and does not frequently cross predefined thresholds. When a workload is running and is not crossing thresholds, updates to the Coupling Facility are made infrequently.

For more information about CICSplex SM Sysplex Optimized Routing, see the “Sysplex optimized workload management learning path” topic in IBM Knowledge Center at this website:

<https://ibm.biz/Bd4YFn>

5.10 Workload consolidation

As described in 1.2.2, “Factors that can influence RNI” on page 5, tuning to reduce the number of simultaneously active address spaces to the proper number that are needed to support a workload can reduce relative nest intensity (RNI) and improve performance.

Combined with improvements in earlier releases, the following areas in CICS TS V5.1 enable you to run more work through fewer CICS regions than ever before:

- ▶ Increase in the concurrent task limit
Changes to the MXT system initialization parameter, which now permits a maximum of 2,000 tasks in a CICS region concurrently, are described in 5.4.6, “Maximum tasks (MXT)” on page 53.
- ▶ Virtual storage constraint relief
The reduction in CICS 24-bit and 31-bit storage usage, as described in 5.6, “Virtual storage constraint relief” on page 60, for CICS TS V5.1 enables more concurrent tasks in a single CICS region.
- ▶ Threadsafe support
Changes in CICS, as described in 5.3, “Improvements in threadsafety” on page 48, can reduce contention for the QR TCB. Workloads that were formerly constrained by the single QR TCB can now be able to run concurrently on an open TCB.

The process of *workload consolidation* involves reducing the number of CICS regions while maintaining the same level of availability, reliability, and throughput. The remainder of this section describes two performance studies that demonstrate how workload consolidation can reduce CPU usage, real storage usage, and operational costs because of requiring management of fewer CICS address spaces while maintaining the same transaction throughput.

5.10.1 Consolidating a COBOL VSAM workload

The first consolidation scenario used a variant of the DSW workload, as described in 3.2, “Data Systems Workload” on page 22. Four TORs used CICSplex SM dynamic routing to distribute work to 30 AORs.

The benchmark followed the standard CICS performance test approach of collecting five measurement intervals at increasing transaction rates. For more information about the methodology used, see 2.6, “Collecting performance data” on page 18. Also, CPU measurement facility (CPU MF) data was collected for the final measurement interval to aid analysis of the workload.

A measurement was created to determine the transaction cost and resource usage before the workload consolidation exercise. The results of this measurement are listed in Table 5-11.

Table 5-11 CPU cost and storage used for DSW workload with 30 AORs

ETR	CICS CPU (% of single CP)	CPU per transaction (ms)	Frames of real storage used
4983.60	253.74%	0.640	736,961
6385.12	325.48%	0.635	737,319
10135.28	510.46%	0.619	738,387
13969.74	704.09%	0.616	739,682
15898.14	821.69%	0.629	740,917

The average CPU cost per transaction is calculated as 0.627 ms. During the highest transaction rate measurement interval, the real storage usage peaked at 740,917 frames, or 2.87 GB.

The workload was then reconfigured to use only 10 AORs. This reconfiguration was starting 10 CICS AOR address spaces, rather than 30 as in the original setup. No other configuration changes were made, with CICSplex SM dynamically recognizing only 10 AORs were available.

By using the same data collection methodology, the benchmark was run again and the results are listed in Table 5-12.

Table 5-12 CPU cost and storage used for DSW workload with 10 AORs

ETR	CICS CPU (% of single CP)	CPU per transaction (ms)	Frames of real storage used
4969.95	232.11%	0.582	342,299
6390.11	293.22%	0.568	342,460
10137.49	456.27%	0.551	342,893
13969.68	620.51%	0.540	343,470
15867.72	725.80%	0.557	343,775

In this configuration, the average CPU cost per transaction is calculated as 0.560 ms. The peak real storage usage was recorded as 343,775 frames, or 1.31 GB.

Although both configurations sustained a throughput of over 15,800 transactions per second, the 10 AOR setup reduced the CPU cost per transaction by 11%. Real storage usage also decreased by over 50%.

It is clear that reducing the number of CICS regions reduces the overall real storage usage. By using the CPU MF data, it also can be seen how the 10 AOR configuration achieves the observed performance improvement.

Table 5-13 on page 68 presents some of the most significant metrics that were obtained from the CPU MF data that was recorded as part of the measurement process. This data was obtained through a sampling process. Therefore, all counters are not absolute, but are values that present a statistical view of the workload behavior when run for the workload over a sufficiently long time.

Table 5-13 Comparison of 30 AOR and 10 AOR CPU MF data for DSW workload

Performance metric	30 AORs	10 AORs	Delta
Execution samples	2,487,298	2,201,099	-11%
Instruction first cycle (IFC)	379,000	371,470	-2%
Microseconds per transaction	628.34	556.43	-11%
Cycles per instruction (CPI)	6.53	5.90	-10%
MIPS per CP	797	882	+10%
Data cache misses	744,894	608,550	-18%
Instruction cache miss (includes TLB miss)	90,483	66,626	-26%
% cycles used by TLB misses	6.82%	5.94%	-13%
Relative nest intensity	0.48	0.34	

The following data points are extracted from the CPU MF samples:

► Execution samples

Represents the number of CPU MF samples that were taken while code was run in a CICS address space used in the workload.

► Instruction first cycle (IFC)

Provides a relative indication as to the number of instructions run in the workload.

► Microseconds per transaction

Post-processing the CPU MF data requires the counter data and an input of the sustained transaction rate. The post-processing tools can calculate the total CPU that was used from the sample data. Therefore, the tools also can calculate a CPU cost per transaction value.

► Cycles per instruction (CPI)

This value represents the average number of CPU clock cycles an instruction took to run. Some instructions can be run in a single clock cycle (such as a register to register operation). Other instructions can take hundreds or thousands of cycles if the operation is moving data within real storage and must wait because data is not available in the cache.

► MIPS per CP

As described in 1.2.1, “Memory hierarchy and nest” on page 5, the instruction execution rate of a processor can vary. This metric uses the collected data to determine the millions of instructions per second (MIPS) that each CP managed to achieve during the sample period.

► Data cache misses

This counter represents the number of cycles for which a processor was waiting for data to become available from the cache.

► Instruction cache miss

This counter provides an indication of the number of cycles for which a processor was waiting for instructions to become available from the cache. It also includes time spent while the storage subsystem performs the translation from a virtual to a real storage address when the dynamic-address-translation (DAT) mechanism does not have an entry in the translation lookaside buffer (TLB). For more information about the DAT process and the hardware TLB, see Chapter 3, “Storage”, in *z/Architecture Principles of Operation*, SA22-7832.

- ▶ Percent of cycles used by TLB misses

Represents the fraction of cycles that were spent waiting for a TLB miss, where a TLB miss is defined in the *Instruction cache miss* metric.

- ▶ Relative nest intensity

RNI is defined in 1.2, “Relative nest intensity” on page 4. The RNI metric is calculated by the post-processing tools by using a formula that is specific to the hardware on which the workload is running. This formula uses a weighted count of cache misses from each of the cache levels in the memory hierarchy. For more information about the formulas that are used, see *LSPR workload categories*, which is available at this website:

<https://www.ibm.com/servers/resourceLink/lib03060.nsf/pages/lsprwork>

The microseconds per transaction value is reduced by approximately 11% in the second configuration. This delta matches the relative change as measured by RMF. The sampling data CPU cost per transaction does not exactly match the value as reported by RMF because of the manner in which CPU is accounted in instructions that operate across address spaces.

The number of instructions run (represented by IFC) for both environments changes only by approximately 2%, which is expected given that they are running the same number of CICS transactions. The reduction in CPU cost per transaction is because of the smaller number of data caches, instruction cache, and TLB misses. As predicted by 1.2.1, “Memory hierarchy and nest” on page 5, the MIPS rate increases because there are fewer cycles that are wasted while the processor is waiting for data to be retrieved from deep in the memory hierarchy.

As described in 1.2, “Relative nest intensity” on page 4, the RNI of the workload is also reduced because the processor does not need to go as deep into the memory hierarchy to find the required data.

5.10.2 Consolidating the GENAPP workload

The second consolidation scenario used a variant of the General Insurance Application (GENAPP). GENAPP is available for download as SupportPac CB12, which is available at this website:

<http://www.ibm.com/support/docview.wss?uid=swg24031760>

The workload was driven by using the web services extensions that are included in the SupportPac. For performance purposes, the supplied VSAM files and DB2 database definitions were extended to include a larger working set of data.

The baseline performance measurement that uses 30 AORs was run and the results are listed in Table 5-14.

Table 5-14 CPU cost and storage used for GENAPP workload with 30 AORs

ETR	CICS CPU (% of single CP)	CPU per transaction (ms)	Frames of real storage used
828.31	94.85%	1.145	862,739
992.14	114.24%	1.151	873,593
1237.67	139.43%	1.126	880,690
1633.98	185.24%	1.133	897,041
1883.25	233.38%	1.239	959,291

The average CPU cost per transaction is calculated as 1.159 ms. During the highest transaction rate measurement interval, the real storage usage peaked at 959,291 frames, or 3.66 GB.

As with the DSW scenario, the workload was then reconfigured to use only 10 AORs with no other changes to the workload. By using the same data collection methodology, the benchmark was run again and the results are listed in Table 5-15.

Table 5-15 CPU cost and storage used for GENAPP workload with 10 AORs

ETR	CICS CPU (% of single CP)	CPU per transaction (ms)	Frames of real storage used
827.72	86.42%	1.044	381,422
986.51	104.35%	1.057	389,384
1231.89	129.67%	1.052	394,495
1629.05	166.94%	1.024	399,247
1916.36	209.88%	1.095	464,827

In this configuration, the average CPU cost per transaction is calculated as 1.054 ms. The peak real storage usage was recorded as 464,827 frames, or 1.77 GB.

The sustained transaction rate for each configuration was similar (approximately 1,900 transactions per second), but the 10 AOR setup reduced the CPU cost per transaction by around 9%. Again, peak real storage usage decreased by over 50%.

By using the metrics as defined for Table 5-13 on page 68, the CPU MF data is listed for the GENAPP workload in Table 5-16.

Table 5-16 Comparison of 30 AOR and 10 AOR CPU MF data for GENAPP workload

Performance metric	30 AORs	10 AORs	Delta
Execution samples	3,517,830	3,188,565	-9%
Instruction first cycle (IFC)	589,236	590,667	+2%
Microseconds per transaction	1240	1095	-11%
Cycles per instruction (CPI)	5.97	5.39	-10%
MIPS per CP	898	1003	+12%
Data cache misses	1,145,876	932,896	-19%
Instruction cache miss (includes TLB miss)	149,468	115,015	-23%
% cycles used by TLB misses	9.95	9.23	-7%
Relative nest intensity	0.75	0.51	

The microseconds per transaction value is again reduced by approximately 11% in the second configuration. This reduction in CPU cost per transaction is the result of a drop in data cache, instruction cache, and TLB misses. The MIPS rate for each processor is increased for the same reasons described in the DSW scenario. The RNI metric also decreases because the formula used accounts for the number of cache misses for the workload.

5.11 Effect of threadsafe transient data

CICS TS V5.1 makes the transient data commands threadsafe when used with a queue in a local CICS region, or function shipped to a remote CICS region over an IPIC connection.

This section describes the benefits of threadsafe transient data commands when combined with other improvements in CICS threadsafe support.

5.11.1 Maximizing time on an open TCB

An example threadsafe application is defined to run in user key, make calls to DB2, and is required to maximize the amount of time spent running on an open TCB. Before CICS TS V4.2, the following possible changes are made to the PROGRAM definition:

- ▶ Specify API (CICSAPI)

As described in 4.3.2, “Programs specifying JVM(NO) and API(CICSAPI)” on page 34, the program begins execution on the QR TCB, and then switches to an L8 TCB when making a DB2 call. Execution continues on the L8 TCB until a non-threadsafe CICS command is run, at which time it switches back to the QR TCB and stays there until the next DB2 call.

- ▶ Specify API (OPENAPI)

As described in 4.3.3, “Programs specifying JVM(NO) and API(OPENAPI)” on page 35, the program begins execution on an L9 open TCB. The drawback of specifying OPENAPI is that execution switches to an L8 TCB for each DB2 call, and switches back to an L9 TCB on completion.

CICS TS V4.2 introduced the value REQUIRED for the CONCURRENCY attribute of a CICS PROGRAM definition as a method of maximizing time on an open TCB. As shown in Table 4-1 on page 34, execution begins on an L8 TCB and remains on the L8 TCB when running a DB2 call. When a non-threadsafe CICS command is run, the application switches execution to the QR TCB, but then switches back to the L8 TCB on completion of the command.

5.11.2 Sample transient data and DB2 application

A test application was written that reads a row from a DB2 table and then writes a record to a transient data queue. This DB2 read and then TD write process was repeated 150 times. Three configurations were tested, with the following CICS releases and program attributes:

- ▶ CICS TS V4.1 - CONCURRENCY (THREADSAFE)
- ▶ CICS TS V4.2 - CONCURRENCY (REQUIRED)
- ▶ CICS TS V5.1 - CONCURRENCY (REQUIRED)

In all cases, the program was defined as API (CICSAPI).

In the CICS TS V4.1 configuration, the program starts on the QR TCB until it makes a DB2 call, and then switches to an L8 TCB. The program stays on the L8 until it makes a non-threadsafe call, such as a transient data read. The non-threadsafe call causes execution to switch back to the QR. Program execution now stays on the QR until the next DB2 call when it switches back to the L8. This process continues until the end of the program. Although all of this application code is coded to threadsafe standards and can run on open TCBs, a large portion of it is running on the QR.

In CICS TS V4.2, the program can be defined as `CONCURRENCY (REQUIRED)`. Now the application starts on an L8 TCB and stays there until it makes a non-threadsafe call, such as the transient data write. When the transient data command completes, execution switches back to the L8 immediately and continues running on the open TCB. Now only a small portion of the application runs on the QR TCB and most of it runs on the L8 TCB reducing contention on the QR TCB.

When the application is moved to CICS TS V5.1, the transient data write, which caused TCB change mode operations in earlier releases, is now threadsafe. The entire application can now run on open TCBs with a reduction in CPU per transaction because of the removal of the switches.

5.11.3 CICS monitoring data results

By using the application as described in 5.11.2, “Sample transient data and DB2 application” on page 71, a workload was run and CMF data was collected.

The CMF data was post-processed by using the CICS Performance Analyzer tool. Extracts from key metrics are listed in Table 5-17.

Table 5-17 CICS monitoring data for transient data and DB2 application

CICS	Average response time (ms)	Average user CPU time (ms)	Average QR CPU time (ms)	Average L8 CPU time (ms)	Average change mode	Average TD cmd count	Average RMI DB2 time (ms)
V4.1	11.942	6.967	4.597	2.370	302	150	1.626
V4.2	11.393	6.875	0.212	6.663	306	150	1.420
V5.1	6.805	6.195	0.026	6.169	8	150	1.147

The CICS V4.1 configuration shows an average QR CPU usage of 4.597 ms per transaction and an L8 TCB average of 2.370 ms per transaction. As described in 5.11.2, “Sample transient data and DB2 application” on page 71, the application is coded to threadsafe standards, but most of the application runs on the QR TCB. Average CPU per transaction is reported as 6.967 ms.

When moving the application to CICS V4.2 and `CONCURRENCY (REQUIRED)`, most of the code now runs on an open TCB. The average QR CPU usage is reduced to 0.212 ms per transaction, while the L8 TCB average is increased to 6.663 ms per transaction. There are still a high number of TCB change mode operations because the transient data write command is non-threadsafe in CICS V4.2. Average CPU per transaction remains effectively unchanged at 6.875 ms.

Moving the application to CICS V5.1 introduces threadsafe transient data, which removes the requirement for most of the TCB change mode operations. The number of change modes is reduced from 306 to 8. Only start-of-task and end-of-task processing are now run on the QR TCB, which reduces the average QR CPU usage down to 0.026 ms per transaction. The reduction in change mode operations reduces the CPU usage overall, with the total CPU per transaction reduced from 6.967 ms to 6.195 ms, which is a reduction of over 10% for this application.

Moving to `CONCURRENCY (REQUIRED)` and CICS V5.1 also reduces the average response time significantly, from 11.942 ms to 6.805 ms.

5.11.4 Throughput results

Figure 5-8 shows the CPU usage of the CICS region as the transaction rate increases for the workload, in all of the CICS TS V4.1, V4.2, and V5.1 configurations.

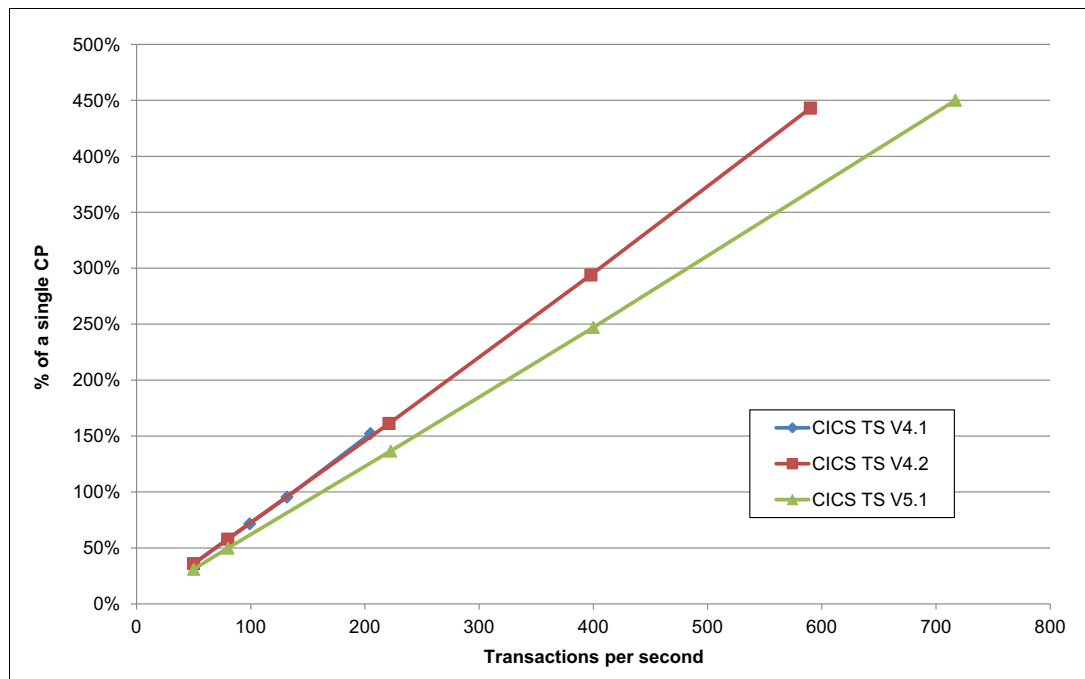


Figure 5-8 CPU comparison for TD and DB2 workload on CICS TS V4.1, V4.2, and V5.1

Figure 5-8 also shows that the CICS V4.1 configuration becomes constrained by the QR TCB at around 200 transactions per second. This result is in line with expectations based on the figures in Table 5-17 on page 72. In the CICS V4.1 results, each transaction required 4.597 ms of CPU time on the QR TCB. Therefore, the QR TCB can sustain only a maximum of $1000 / 4.597 = 217$ transactions per second.

Specifying a value of REQUIRED for the CONCURRENCY attribute in CICS V4.2, the constraint on the QR TCB is eliminated. In this configuration, the CPU cost per transaction is approximately the same as for CICS V4.1 (as shown by the overlapping lines) but the maximum throughput increases to around 570 transactions per second. In this configuration, the constraint is now the total CPU available on the LPAR.

The elimination of the TCB switching in CICS V5.1 reduces total CPU per transaction and enables even higher throughput to be achieved before being limited by the available CPU on the LPAR.

5.12 Transaction isolation

CICS transaction isolation builds on CICS storage protection, which enables user transactions to be protected from one another. Transaction isolation uses the MVS subspace group facility to offer protection between transactions. This configuration ensures that an application program that is associated with one transaction cannot accidentally overwrite the data of another transaction.

Transaction isolation is enabled globally for a CICS region by specifying `TRANISO=YES` in the CICS SIT parameter at startup. Transaction isolation requires storage protection to be enabled, which is achieved by specifying `STGPROT=YES` as a CICS SIT parameter.

In addition to specifying the storage and execution key individually for each user transaction, you can specify that CICS is to isolate the user-key task-lifetime storage of a transaction to provide transaction-to-transaction protection. You complete this task by using the `ISOLATE` option of the `TRANSACTION` resource definition.

Transaction isolation does not apply to 64-bit storage.

For more information about the use of transaction isolation in a CICS environment, see the topic “CICS storage protection and transaction isolation” in IBM Knowledge Center at this website:

<https://ibm.biz/Bd4YEH>

Transaction isolation is not a new feature in CICS TS V5.1, although changes to default values in this release can affect how storage is allocated when compared to previous CICS TS releases. The remainder of this section meets the following goals:

- ▶ Describes the concepts of transaction isolation
- ▶ Provides a study that demonstrates the overhead of enabling transaction isolation
- ▶ Highlights how changes to default values in CICS TS V5.1 can affect your environment when running with transaction isolation enabled

5.12.1 Unique subspaces

In general, transaction isolation ensures that user-key programs are allocated to separate (unique) *subspaces*, and have the following characteristics:

- ▶ `ISOLATE(YES)` and `KEY(USER)` on the transaction definition
- ▶ Read and write access to the user-key task-lifetime storage of their own tasks, which is allocated from one of the user dynamic storage areas (UDSA or EUDSA)
- ▶ Read and write access to shared storage, which is storage obtained by `GETMAIN` commands with the `SHARED` option (SDSA or ESDSA)
- ▶ Read access to the CICS-key task-lifetime storage of other tasks (CDSA or ECDSA)
- ▶ Read access to CICS code
- ▶ Read access to CICS control blocks that are accessible by the CICS API

User-key programs do not have any access to user-key task-lifetime storage of other tasks.

You might have some transactions where the application programs access one another's storage in a valid way. One such case is when a task waits on one or more event control blocks (ECBs) that are later posted by an `MVS POST` or *hand posting* by another task.

For example, a task can pass the address of a piece of its own storage to another task (by a temporary storage queue or some other method) and then `WAIT` for the other task to post an ECB to say that it includes updated the storage. If the original task is running in a unique subspace, the posting task fails when attempting the update and to post the ECB, unless the posting task is running in CICS-key.

CICS supports the following methods to ensure that transactions that must share storage can continue to work in the subspace group environment:

- ▶ Use of the common subspace
- ▶ Use of the base space
- ▶ Use of common storage by obtaining the storage with the SHARED option

5.12.2 Common subspace

You can specify that all the related transactions are to run in the *common subspace*. The common subspace allows tasks that must share storage to coexist, while isolating them from other transactions in the system. Transactions that are assigned to the common subspace have the following characteristics:

- ▶ They specify ISOLATE(NO) on the transaction definition.
- ▶ They have read and write access to each other's task-lifetime storage.
- ▶ They have no access of any kind to storage of transactions that run in unique subspaces.
- ▶ They have read only access to CICS storage.

5.12.3 Base space

Programs that are defined with EXECKEY(CICS) run in the *base space*.

You can ensure that the application programs of the transactions that are sharing storage are all defined with EXECKEY(CICS). This setting ensures that their programs run in base space, where they have read and write access to all storage. However, this method is not recommended because it does not give any storage protection.

5.12.4 Transaction isolation performance effect

The performance figures in this section were obtained when running the DSW static routing workload as described in 3.2.1, "DSW static routing" on page 22 by using CICS TS for z/OS V5.1. Table 5-18 lists the CPU cost per transaction and number of real frames of storage that is used by the workload when running with transaction isolation disabled.

Table 5-18 CPU cost per transaction and real storage frames used with transaction isolation disabled

ETR	Real storage frames	CPU per transaction (ms)
2072.30	163,292	0.618
2842.24	163,292	0.609
4130.87	163,335	0.594
5047.97	163,335	0.594
5681.45	163,487	0.586

The average CPU cost per transaction is calculated to be 0.600 ms.

Table 5-19 lists the CPU cost per transaction and number of real frames of storage that are used by the same workload when running with transaction isolation enabled.

Table 5-19 CPU cost per transaction and real storage frames used with transaction isolation enabled

ETR	Real storage frames	CPU per transaction (ms)
2073.25	188,103	0.677
2842.38	188,103	0.670
4129.20	188,138	0.659
5044.09	185,032	0.658
5676.44	185,111	0.655

The average CPU cost per transaction is calculated to be 0.664 ms.

The data in Table 5-18 on page 75 and Table 5-19 shows that the use of transaction isolation does have a cost. In this workload, the increase in CPU was about 10% where all of the transactions ran in unique subspaces. There was also an increase in real storage usage.

5.12.5 Page and extent sizes

The transaction isolation facility increases the allocation of some 31-bit virtual storage for CICS regions that are running with transaction isolation active.

If you are running with transaction isolation active, CICS allocates storage for task-lifetime storage in multiples of 1 MB for user-key tasks that require 31-bit storage. The minimum unit of storage allocation in the EUDSA when transaction isolation is active is 1 MB.

Table 5-20 lists the page and extent sizes for the UDSA and EUDSA storage areas, where transaction isolation is disabled and transaction isolation is enabled.

Table 5-20 Page and extent sizes with transaction isolation disabled and enabled

Transaction isolation	Page size		Extent size	
	No	Yes	No	Yes
UDSA	4 KB	4 KB	256 KB	1 MB
EUDSA	64 KB	1 MB	1 MB	1 MB

As described in 5.6.4, "Mirror transactions" on page 61, CICS TS V5.1 changes the default value for the TASKDATALOC attribute on a TRANSACTION resource from BELOW to ANY. This change also affects the mirror transaction CSMI.

If running with transaction isolation enabled, the change to TASKDATALOC for the mirror transaction might cause higher peak usage in EUDSA because any GETMAIN request for user storage by this transaction is allocated in 31-bit storage, not 24-bit storage. As shown in Table 5-20, the minimum storage that is allocated is 1 MB, not 4 KB.



CICS TS for z/OS V5.2

This chapter describes many of the performance-related improvements that are found in the CICS TS for z/OS V5.2 release. The following subject areas are included in the CICS TS V5.2 performance report:

- ▶ Key performance benchmarks that are presented as a comparison against the CICS TS V5.1 release.
- ▶ An outline of improvements that were made regarding the threadsafe characteristics of the CICS run time.
- ▶ Details of the changes that were made to performance-critical CICS initialization parameters and the affect of these updates.
- ▶ Description of all the new and updated monitoring fields, including examples where necessary.
- ▶ High-level views of new functions that were introduced in the CICS TS V5.2 release, including performance benchmark results where appropriate.

This chapter includes the following topics:

- ▶ 6.1, “Introduction” on page 78
- ▶ 6.2, “Release-to-release comparisons” on page 78
- ▶ 6.3, “Improvements in threadsafe API and SPI commands” on page 85
- ▶ 6.4, “Changes to system initialization parameters” on page 86
- ▶ 6.5, “Enhanced instrumentation” on page 87
- ▶ 6.6, “Kerberos” on page 91
- ▶ 6.7, “JSON support” on page 95
- ▶ 6.8, “Java applications and trace” on page 102
- ▶ 6.9, “Web services over HTTP improvements” on page 104
- ▶ 6.10, “Java 7.0 and Java 7.1” on page 106

6.1 Introduction

When the results for this chapter were compiled, the workloads were run on an IBM zEnterprise EC12 model HA1 (machine type 2827). A maximum of 32 dedicated central processors (CPs) were available on the measured logical partition (LPAR), with a maximum of four dedicated CPs available to the LPAR used to simulate users. These LPARs are configured as part of a parallel sysplex. An internal coupling facility was co-located on the same central processor complex (CPC) as the measurement and driving LPARs, which were connected by using internal coupling peer (ICP) links. An IBM System Storage DS8870 (machine type 2424) was used to provide external storage.

This chapter presents the results of several performance benchmarks when run in a CICS TS for z/OS V5.2 environment. Unless otherwise stated in the results, the CICS V5.2 environment was the code available at the general availability (GA) date of 13 June 2014. Several of the performance benchmarks are presented in the context of a comparison against CICS TS V5.1. The CICS TS V5.1 environment contained all PTFs issued before 18 February 2014. All LPARs used z/OS V2.1.

For more information about performance terms that are used in this chapter, see Chapter 1, “Performance terminology” on page 3. A description of the test methodology that was used can be found in Chapter 2, “Test methodology” on page 11. For more information about the workloads that were used, see Chapter 3, “Workload descriptions” on page 21.

Where reference is made to an *LSPR processor equivalent*, the indicated machine type and model can be found in the large systems performance reference (LSPR) document. For more information about obtaining and the use of LSPR data, see 1.3, “Large Systems Performance Reference” on page 6.

6.2 Release-to-release comparisons

This section describes some of the results from a selection of regression workloads that are used to benchmark development releases of CICS TS. For more information about the use of regression workloads, see Chapter 3, “Workload descriptions” on page 21.

6.2.1 Data Systems Workload static routing

The static routing variant of the Data Systems Workload (DSW) is described in 3.2.1, “DSW static routing”. This section presents the performance figures that were obtained by running this workload. Table 6-1 lists the results of the DSW static routing workload that used the CICS TS V5.1 release.

Table 6-1 CICS TS V5.1 results for DSW static routing workload

ETR	CICS CPU	CPU per transaction (ms)
2563.06	57.03%	0.223
3011.97	66.75%	0.222
3613.27	79.61%	0.220
4515.94	98.11%	0.217
6029.03	128.57%	0.213

Table 6-2 lists the same figures for the CICS TS V5.2 release.

Table 6-2 CICS TS V5.2 results for DSW static routing workload

ETR	CICS CPU	CPU per transaction (ms)
2562.81	57.00%	0.222
3011.61	66.74%	0.222
3613.01	79.61%	0.220
4515.30	98.47%	0.218
6028.32	129.29%	0.214

The average CPU per transaction value for CICS TS V5.1 is calculated to be 0.219 ms. The same value for CICS TS V5.2 is also calculated to be 0.219 ms. The performance of this workload is considered to be equivalent across the two releases.

These figures are shown in Figure 6-1.

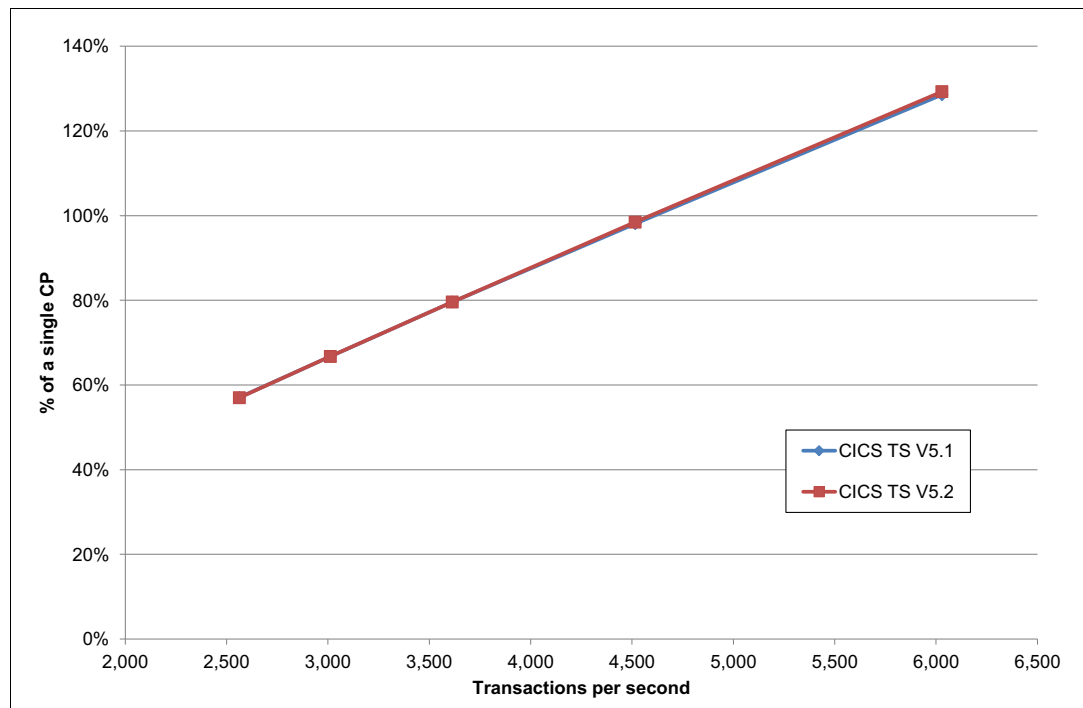


Figure 6-1 Plot of CICS TS V5.1 and V5.2 performance figures for DSW static routing workload

The measured CPU cost for each transaction rate is similar for CICS TS V5.1 and V5.2, which is demonstrated by the two plot lines being almost identical. A final observation is that the CPU cost scales linearly in accordance with the transaction rate.

6.2.2 DSW dynamic routing

The dynamic routing variant of the DSW workload is described in 3.2.2, “DSW dynamic routing”. This section presents the performance figures that were obtained by running this workload.

Table 6-3 lists the results of the DSW dynamic routing workload that used the CICS TS V5.1 release.

Table 6-3 CICS TS V5.1 results for DSW dynamic routing workload

ETR	CICS CPU	CPU per transaction (ms)
3006.60	158.00%	0.526
6118.61	308.48%	0.504
8830.54	440.00%	0.498
11962.02	599.67%	0.501
16238.38	815.93%	0.502

Table 6-4 lists the same figures for the CICS TS V5.2 release.

Table 6-4 CICS TS V5.2 results for DSW dynamic routing workload

ETR	CICS CPU	CPU per transaction (ms)
3005.68	159.81%	0.532
6111.82	311.00%	0.509
8827.54	441.50%	0.500
11963.57	596.41%	0.499
16252.29	817.04%	0.503

The results from Table 6-3 and Table 6-4 are shown in Figure 6-2.

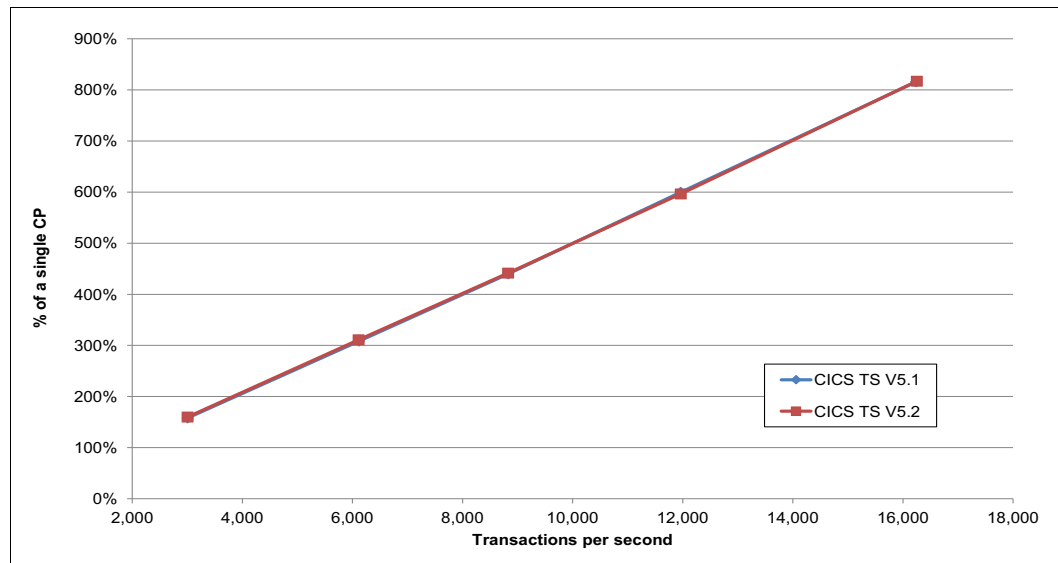


Figure 6-2 Plot of CICS TS V5.1 and V5.2 performance figures for DSW dynamic routing workload

As with the DSW static routing workload, the V5.1 and V5.2 lines are almost identical, which indicates near-identical CPU cost per transaction. The plot lines are also straight, which indicates linear scaling as transaction throughput increases.

6.2.3 Relational Transactional Workload non-threadsafe

The Relational Transactional Workload (RTW) is described in 3.3, “Relational Transactional Workload” on page 25. This section presents the performance figures for the non-threadsafe variant when CICS TS V5.1 is compared with CICS TS V5.2.

Table 6-5 lists the performance results for the RTW non-threadsafe workload that used the CICS TS V5.1 release.

Table 6-5 CICS TS V5.1 results for the RTW non-threadsafe workload

ETR	CICS CPU	CPU per transaction (ms)
250.08	60.71%	2.428
332.92	80.40%	2.415
453.24	113.15%	2.496
585.73	147.30%	2.515
709.60	180.50%	2.544

Table 6-6 lists the same figures for the CICS TS V5.2 release.

Table 6-6 CICS TS V5.2 results for the RTW non-threadsafe workload

ETR	CICS CPU	CPU per transaction (ms)
250.20	60.37%	2.413
332.92	79.88%	2.399
453.54	110.97%	2.447
586.15	145.72%	2.486
710.25	178.82%	2.518

For the V5.1 release, the average CPU cost per transaction is calculated to be 2.480 ms. The V5.2 release produces an average of 2.453 ms. This difference represents a reduction in CPU per transaction of approximately 1%.

The results from Table 6-5 on page 81 and Table 6-6 on page 81 are shown in Figure 6-3.

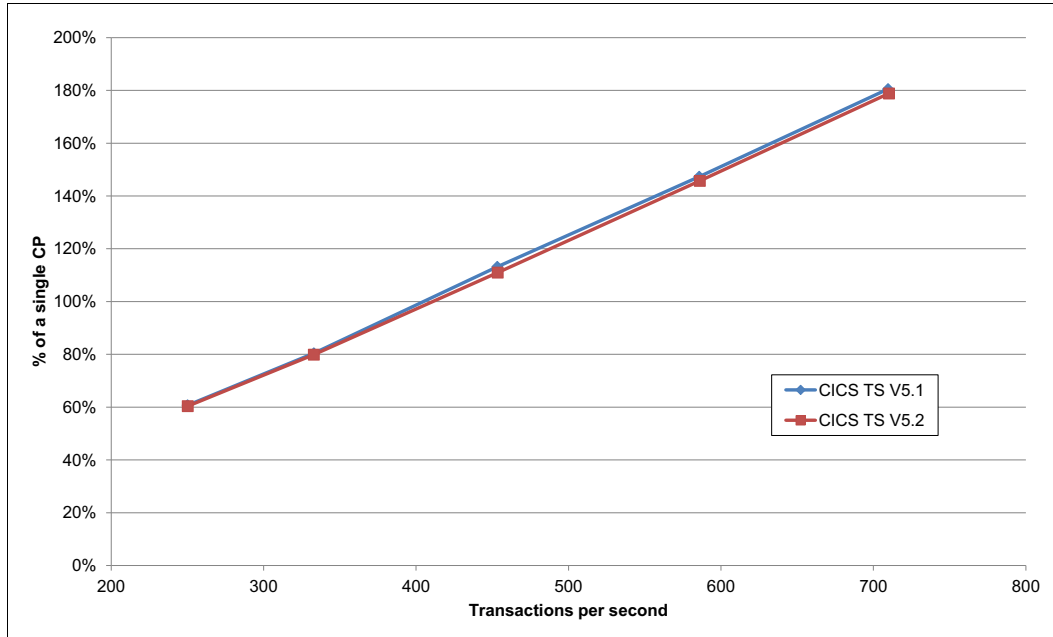


Figure 6-3 Plot of CICS TS V5.1 and V5.2 performance figures for RTW non-threadsafe workload

The straight line represents linear scaling as throughput increases. The CICS TS V5.2 line shows a slight improvement when compared with CICS TS V5.1.

6.2.4 RTW threadsafe

This section presents the performance figures for the threadsafe variant of the RTW, as described in 3.3, “Relational Transactional Workload” on page 25.

Table 6-7 lists the results of the RTW threadsafe workload that used the CICS TS V5.1 release.

Table 6-7 CICS TS V5.1 results for the RTW threadsafe workload

ETR	CICS CPU	CPU per transaction (ms)
333.14	53.86%	1.617
498.78	80.12%	1.606
711.30	114.03%	1.603
990.59	157.05%	1.585
1227.39	195.89%	1.596

Table 6-8 lists the same figures for the CICS TS V5.2 release.

Table 6-8 CICS TS V5.2 results for the RTW threadsafe workload

ETR	CICS CPU	CPU per transaction (ms)
333.79	53.63%	1.607
499.18	79.72%	1.597
711.84	114.13%	1.603
991.11	157.43%	1.588
1228.72	196.47%	1.599

These performance results are shown in Figure 6-4.

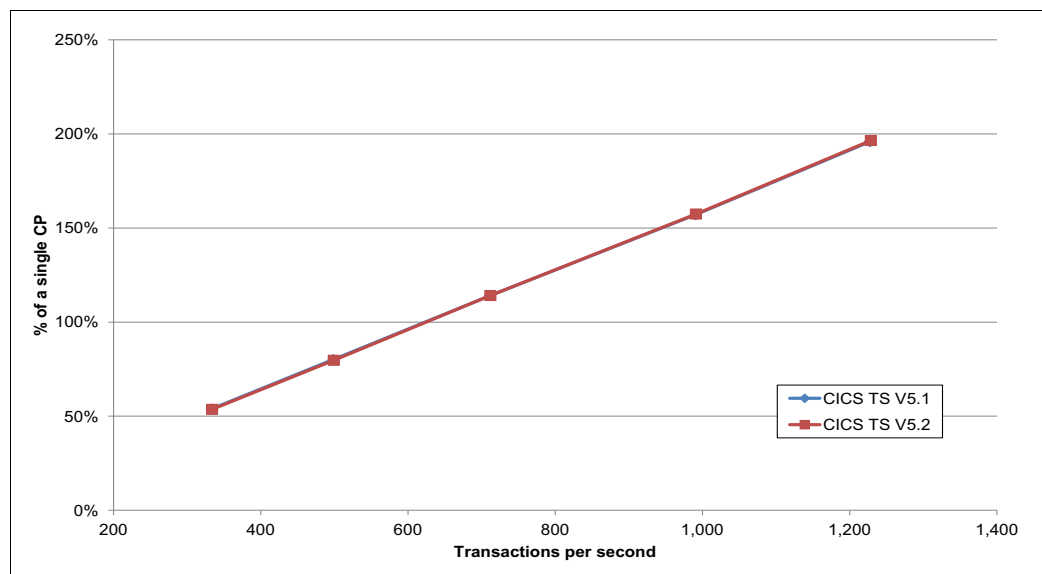


Figure 6-4 Plot of CICS TS V5.1 and V5.2 performance figures for RTW threadsafe workload

The straight line indicates linear scaling as throughput increases, while the almost identical lines demonstrate equivalent performance between the two CICS releases.

6.2.5 Java servlet that uses JDBC and VSAM

The JDBC and VSAM Java servlet benchmark was run in the environment as described in 6.1, “Introduction” on page 78. The LPAR that was used for performance measurements was configured with three dedicated CPs and one dedicated IBM System z® Integrated Information Processor (zIIP). The LPAR that was used to drive workload into the measured system was configured with three dedicated CPs. Both of these LPARs represent an LSPR equivalent processor of 2827-703. All configurations used IBM DB2 for z/OS V10, with IBM Workload Simulator for z/OS V1.1.0.1 driving the workload as 200 simulated HTTP clients.

All tests used a CICS region that contained a single JVMSERVER resource with the THREADLIMIT attribute set to 100. The following garbage collection and memory management options were specified in the relevant JVM profile configuration file:

- ▶ -Xgcpolicy:gencon
- ▶ -Xcompressedrefs

- ▶ -XXnosuballc32bitmem
- ▶ -Xmx200M
- ▶ -Xms200M
- ▶ -Xmnx60M
- ▶ -Xmns60M
- ▶ -Xmox140M
- ▶ -Xmos140M

For more information about these JVM tuning parameters, see the “Command-line options” topic in IBM Knowledge Center at the following website:

<https://ibm.biz/Bd4YSB>

To minimize variance in the performance results that might be introduced by the just-in-time (JIT) compiler, the workload was run at a constant transaction rate for 20 minutes to provide a warm-up period. The request rate was increased every 10 minutes, with the mean CPU usage per request calculated by using the final five minutes of data from the 10-minute interval.

As described for other workloads in 2.6, “Collecting performance data” on page 18, IBM Resource Measurement Facility™ (RMF) was used to record CPU consumption at an address space level.

CICS TS V5.1 and CICS TS V5.2 used Java 7.0 SR7 (64-bit). In addition to the software environment that is described in 6.1, “Introduction” on page 78, the following software versions were used:

- ▶ CICS TS V5.1 with IBM WebSphere Application Server Liberty V8.5.0.0
- ▶ CICS TS V5.2 with IBM WebSphere Application Server Liberty V8.5.0.1

Table 6-9 lists the performance measurements when running the workload in a CICS TS V5.1 environment.

Table 6-9 CICS TS V5.1 results for the Java servlet JDBC and VSAM workload

ETR	CICS CPU			CPU per request (ms)
	Not zIIP-eligible	zIIP-eligible	Total	
200	20.87%	30.03%	50.90%	2.545
400	42.02%	60.81%	102.83%	2.571
800	86.61%	120.95%	207.55%	2.594

Table 6-10 lists the performance measurements when running the workload in a CICS TS V5.2 environment.

Table 6-10 CICS TS V5.2 results for the Java servlet JDBC and VSAM workload

ETR	CICS CPU			CPU per request (ms)
	Not zIIP-eligible	zIIP-eligible	Total	
200	20.82%	29.42%	50.24%	2.512
400	42.34%	60.12%	102.46%	2.562
800	86.46%	120.38%	206.85%	2.586

A subset of the performance measurements in Table 6-9 on page 84 and Table 6-10 on page 84 is shown Figure 6-5.

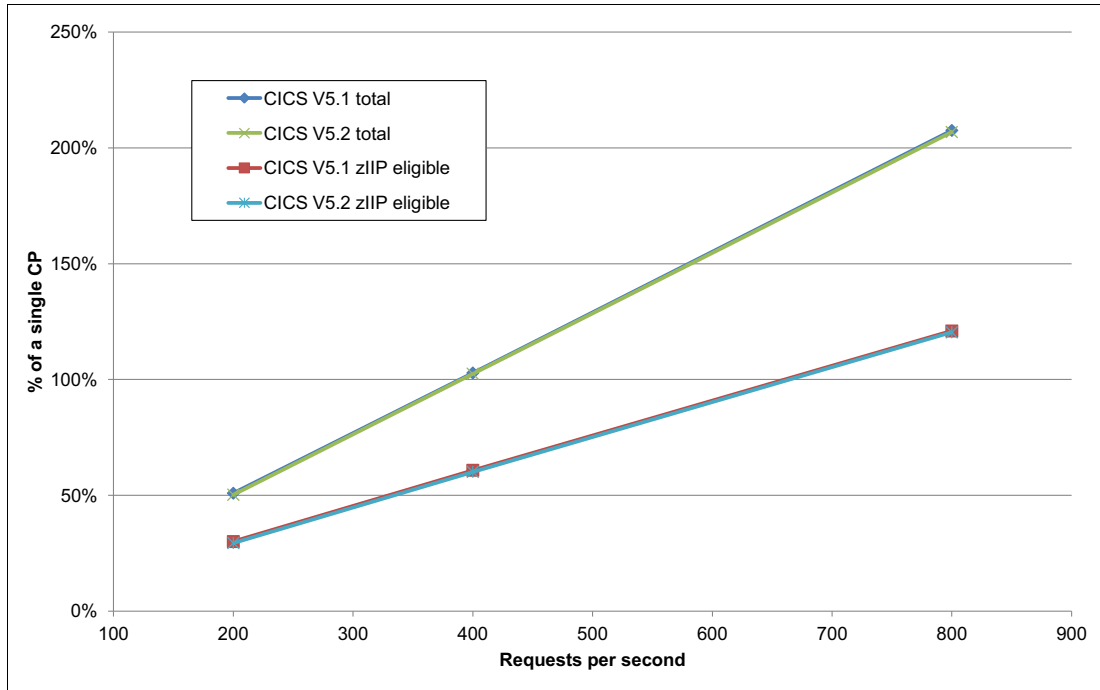


Figure 6-5 CICS TS V5.1 and CICS TS V5.2 performance results for JDBC and VSAM workload

The total CPU lines for CICS TS V5.1 and CICS TS V5.2 are almost identical and demonstrate similar performance when comparing the two CICS TS releases. The straight lines represent linear scaling as the transaction rate increases.

Also, the fraction of total workload that is eligible for offload features similar profiles between the CICS TS V5.1 and CICS TS V5.2 releases. Again, the straight line demonstrates the linear scaling nature of the environment.

For the sake of clarity, the non-zIIP-eligible performance results were omitted from Figure 6-5; however, it can be concluded that non-zIIP-eligible time must scale in a similar manner as shown in the following equation:

$$\text{Total CPU time} = \text{non zIIP-eligible time} + \text{zIIP-eligible time}$$

6.3 Improvements in threadsafe API and SPI commands

CICS V5.2 brings further improvements in threadsafe API and SPI commands. The following new CICS API commands are threadsafe:

- ▶ INVOKE APPLICATION
- ▶ VERIFY TOKEN

For more information about CICS API commands, see the “CICS command summary” topic in IBM Knowledge Center at this website:

<https://ibm.biz/Bd4YSx>

The following CICS SPI commands were made threadsafe:

- ▶ INQUIRE MVSTCB
- ▶ DISPATCHER commands:
 - INQUIRE DISPATCHER
 - SET DISPATCHER
- ▶ MONITOR commands:
 - INQUIRE MONITOR
 - SET MONITOR
- ▶ PROGRAM commands:
 - INQUIRE PROGRAM
 - SET PROGRAM
 - DISCARD PROGRAM
- ▶ STATISTICS commands:
 - EXTRACT STATISTICS
 - INQUIRE STATISTICS
 - SET STATISTICS
- ▶ SYSTEM commands:
 - INQUIRE SYSTEM
 - SET SYSTEM
- ▶ TRANSACTION commands:
 - INQUIRE TRANSACTION
 - SET TRANSACTION
 - DISCARD TRANSACTION

For more information about CICS SPI commands, see the “System commands” topic in IBM Knowledge Center at this website:

<https://ibm.biz/Bd4YSD>

6.4 Changes to system initialization parameters

Several performance-related CICS system initialization parameters were changed in the CICS TS V5.2 release.

6.4.1 Maximum tasks (MXT)

The default value for the maximum tasks (MXT) system initialization parameter was changed to 250.

Note: The default MXT value is provided as a starting point when tuning a CICS system and does not represent an optimal configuration for all workloads. The MXT parameter must be configured according to the expected transaction throughput for the system.

You must ensure that enough storage is available to support the maximum number of tasks value. For more information about setting the MXT value, see the “Setting the maximum task specification (MXT)” topic in IBM Knowledge Center at this website:

<https://ibm.biz/Bd4YSi>

CICS allocates several MVS workload manager (WLM) performance blocks (PBs) that are based on the current setting of the MXT parameter. Specifying an excessively large value for MXT can result in wasted use of local system queue area (LSQA) storage. Where many unused PBs exist, CPU cycles can be consumed unnecessarily by WLM because all blocks must be scanned to determine the status of the performance goal metrics.

6.4.2 Maximum open TCBs (MAXOPENTCBS)

The maximum open TCBs (MAXOPENTCBS) system initialization parameter was removed in CICS TS V5.1, but for tuning purposes is reinstated in the CICS TS V5.2 release. If the MAXOPENTCBS parameter is not specified, CICS sets the parameter automatically based on the current value of the MXT system initialization parameter. The value of MAXOPENTCBS is calculated by using the same algorithm implemented in CICS TS V5.1, as shown in the following example:

$$\text{MAXOPENTCBS} = (2 \times \text{MXT}) + 32$$

For more information about the MAXOPENTCBS parameter, see the “MAXOPENTCBS” topic in IBM Knowledge Center at this website:

<https://ibm.biz/Bd4YvM>

Note: The minimum value for the MAXOPENTCBS parameter was increased to 32 in CICS TS V5.4.

6.4.3 Maximum XP TCBs (MAXXPTCBS)

The maximum XP TCBs (MAXXPTCBS) system initialization parameter was removed in CICS TS V5.1. However, it is reinstated in the CICS TS V5.2 release for tuning purposes. If the MAXXPTCBS parameter is not specified, CICS sets it automatically based on the current value of the MXT system initialization parameter. The value of MAXXPTCBS is calculated by using the same algorithm implemented in CICS TS V5.1, as shown in the following example:

$$\text{MAXXPTCBS} = \text{MXT}$$

For more information about the MAXXPTCBS parameter, see the “MAXXPTCBS” topic in IBM Knowledge Center at this website:

<https://ibm.biz/Bd4YvS>

6.5 Enhanced instrumentation

The CICS TS V5.2 release continues the expansion of information that is reported by the CICS monitoring and statistics component. This section describes the extra fields that are now available in the CICS statistics SMF records.

6.5.1 Dispatcher global statistics

The following new fields were added to the collected dispatcher global statistics:

- ▶ Last excess TCB scan (field DSGLXSCN)

The date and time of the last CICS dispatcher excess MVS TCB scan.

- ▶ Last excess TCB scan - no TCB detached (field DSGLXSND)

The date and time of the last CICS dispatcher excess MVS TCB scan that did not detach any TCBs.

A sample DFHSTUP report that contains the new fields is shown in Example 6-1.

Example 6-1 Sample dispatcher global statistics report produced by CICS TS V5.2 DFHSTUP

Dispatcher Start Date and Time.	05/16/2014 04:04:34.9633
Address Space CPU Time.	00:00:29.882586
Address Space SRB Time.	00:00:16.516442
Current number of dispatcher tasks.	30
Peak number of dispatcher tasks	75
Current ICV time (msec)	1000
Current ICVR time (msec).	5000
Current ICVTS time (msec).	100
Current PRTYAGE time (msec)	1000
Current MRO (QR) Batching (MROBTCH) value	1
Last Excess TCB Scan.	05/16/2014 05:28:10.1478
Number of Excess TCB Scans.	1
Last Excess TCB Scan - No TCB Detached.	05/16/2014 05:28:10.1478
Excess TCB Scans - No TCB Detached.	1
Number of Excess TCBs Detached.	0
Average Excess TCBs Detached per Scan	0
Number of CICS TCB MODEs.	18
Number of CICS TCB POOLs.	4

For more information about dispatcher global statistics, see the “Dispatcher domain: Global statistics” topic in IBM Knowledge Center at this website:

<https://ibm.biz/Bd4Yvm>

6.5.2 Dispatcher TCB mode statistics

The following new fields were added to the collected dispatcher TCB mode statistics:

- ▶ Dispatchable Queue – Current (field DSGTMCDQ)
The current number of dispatchable tasks that are queued for the TCB.
- ▶ Dispatchable Queue – Peak (field DSGTMPDQ)
The peak number of dispatchable tasks that were queued for the TCB.
- ▶ Dispatchable Queue – Average (field DSGTMADQ)
The average number of dispatchable tasks that were queued for the TCB.

A sample DFHSTUP report that contains the new fields is shown in Example 6-2. The new fields are presented in the final three columns for TCB modes where only one TCB exists.

Example 6-2 Sample dispatcher TCB mode statistics report produced by CICS TS V5.2 DFHSTUP

TCB Mode	Open	TCB Pool	< TCBs Attached > Current	Peak	<- TCBs In Use -> Current	Peak	TCB Attaches	<- Dispatchable Queue -> Current	Peak	Average
QR	No	N/A	1	1	1	1	0	1	27	1.12
RO	No	N/A	1	1	1	1	0	1	1	1.00
CO	Unk	N/A	0	0	0	0	0	0	0	0.00
SZ	Unk	N/A	0	0	0	0	0	0	0	0.00

RP	Unk	N/A	0	0	0	0	0	0	0	0.00
F0	No	N/A	1	1	1	1	0	0	0	0.00
SL	No	N/A	1	1	1	1	0	0	0	0.00
S0	No	N/A	1	1	1	1	0	0	0	0.00
SP	No	N/A	1	1	1	1	0	0	0	0.00
EP	No	N/A	2	2	2	2	0			
TP	Unk	N/A	0	0	0	0	0			
D2	Unk	N/A	0	0	0	0	0			
S8	Unk	N/A	0	0	0	0	0			
L8	Yes	Open	1	1	0	1	0			
L9	Unk	N/A	0	0	0	0	0			
X8	Unk	N/A	0	0	0	0	0			
X9	Unk	N/A	0	0	0	0	0			
T8	Unk	N/A	0	0	0	0	0			
Totals			9		8		0			

For more information about dispatcher TCB mode statistics, see the “Dispatcher domain: TCB Mode statistics” topic in IBM Knowledge Center at this website:

<https://ibm.biz/Bd4YvK>

6.5.3 Dispatcher TCB pool statistics

The Time Max TCB Pool Limit last reached (field DSGLTCBL) field was added to the collected dispatcher TCB pool statistics. This field shows the time at which the pool reached the maximum TCB limit.

A sample fragment of DFHSTUP report that contains the new field is shown in Example 6-3.

Example 6-3 Sample dispatcher TCB pool statistics report produced by CICS TS V5.2 DFHSTUP

```

TCB Pool. . . . . : OPEN
Current TCBS attached in this TCB Pool. . . . . : 170 ...
Peak TCBS attached in this TCB Pool . . . . . : 170 ...
Max TCB Pool limit (MAXOPENTCBS). . . . . : 170 ...
Time Max TCB Pool Limit last reached. . . . . : 15:47:39.2782 ...
Total Requests delayed by Max TCB Pool Limit. . . : 819 ...
Total Max TCB Pool Limit delay time . . . . . : 00:01:57.2105 ...
Current Requests delayed by Max TCB Pool Limit. : 0 ...
Current Max TCB Pool Limit delay time . . . . . : 00:00:00.0000 ...
Peak Requests delayed by Max TCB Pool Limit . . . : 67 ...

```

For more information about dispatcher TCB pool statistics, see the “Dispatcher domain: TCB Pool statistics” topic in IBM Knowledge Center at this website:

<https://ibm.biz/Bd4YvG>

6.5.4 Monitoring domain global statistics

The following fields were added to the collected monitoring domain statistics:

- User transactions ended (field MNGUTNUM)

The number of user transactions that ended.

- ▶ System transactions ended (field MNGSTNUM)
The number of system transactions that ended.
- ▶ Time last user transaction attached (field MNGLOTAT)
The date and time of the last transaction attach processed by the monitoring domain.
- ▶ Time last user transaction ended (field MNGLOTCL)
The date and time at which the last transaction ended.
- ▶ MXT at last user transaction attach (field MNGMXUTA)
The current MXT value at the time of the last transaction attached.
- ▶ Current tasks at last attach (field MNGCAUTA)
The current number of user transactions that are attached in the region at the time of the last transaction attached.
- ▶ Average user transaction resp time (field MNGAUTRT)
The rolling average user transaction response time.
- ▶ Peak user transaction resp time (field MNGPUTRT)
The maximum user transaction response time.
- ▶ Peak user transaction resp time at (field MNGLOTURT)
The time stamp of the maximum user transaction response time.

A sample DFHSTUP report that shows the new statistics fields is shown in Example 6-4.

Example 6-4 Sample monitoring domain global statistics report produced by CICS TS V5.2 DFHSTUP

User transactions ended	:	905698	
System transactions ended	:	11	
Time last user transaction attached	:	05/16/2014 05:28:43.5198	...
Time last user transaction ended	:	05/16/2014 05:28:43.5215	...
Average user transaction resp time	:	00:00:00.001168	
Peak user transaction resp time	:	00:00:00.104882	
Peak user transaction resp time at	:	05/16/2014 05:26:55.8512	
		... MXT at last user transaction attach	650
		... Current tasks at last attach	8

For more information about monitoring domain statistics, see the topic “Monitoring domain: global statistics” in IBM Knowledge Center at this website:

<https://ibm.biz/Bd4Yve>

The rolling average transaction response time is computed using the following equation:

$$\text{avg resp time} = \frac{(\text{current avg resp time} \times \text{num of completions}) + \text{this resp time}}{\text{num of completions} + 1}$$

6.5.5 Transaction manager global statistics

The following new fields were added to the collected transaction manager statistics:

- ▶ Time MAXTASKS last changed (field XMGLSMXT)
The date and time when MXT was last set or changed dynamically.
- ▶ Time last transaction attached (field XMGLTAT)
The date and time when the last user transaction was attached.
- ▶ Time the MAXTASKS limit last reached (field XMGLAMXT)
The date and time when the number of active user transactions last equaled MXT.
- ▶ Currently at MAXTASKS limit (field XMGATMXT)
Indicates whether the CICS region is at MXT.

A sample DFHSTUP report that contains the new fields is shown in Example 6-5.

Example 6-5 Sample transaction manager global statistics report fragment produced by CICS TS V5.2 DFHSTUP

Total number of transactions (user + system)	:	19,274
Current MAXTASKS limit	:	650
Time MAXTASKS last changed	:	05/15/2014 12:20:16.9640
Current number of active user transactions	:	1
Time last transaction attached	:	05/15/2014 12:40:24.6738
Current number of MAXTASK queued user transactions	:	0
Times the MAXTASKS limit reached	:	7
Time the MAXTASKS limit last reached	:	05/15/2014 12:34:21.7237
Currently at MAXTASKS limit	:	No
Peak number of MAXTASK queued user transactions	:	164
Peak number of active user transactions	:	650
Total number of active user transactions	:	19232
Total number of MAXTASK delayed user transactions	:	456
Total MAXTASK queuing time	:	000-00:00:13
Total MAXTASK queuing time of currently queued user transactions	:	00:00:00

For more information about transaction manager statistics, see the topic “Transaction manager: Global statistics” in IBM Knowledge Center at this website:

<https://ibm.biz/Bd4YvA>

6.6 Kerberos

CICS TS V5.2 introduces support for Kerberos token validation. CICS TS supports Kerberos by using the external security manager (ESM). The level of support depends on the support that is provided by the ESM. If your ESM is RACF, support is based on Kerberos Version 5 and Generic Security Services (GSS).

CICS can verify a Kerberos token by configuring a service provider pipeline or by using the API command **VERIFY TOKEN**. An extract of a web service provider pipeline file that is configured to use Kerberos support is shown in Example 6-6.

Example 6-6 Extract of web services pipeline file configured for Kerberos support

```
<service_handler_list>
  <wsse_handler>
    <dfhwsse_configuration version="1">
      <authentication trust="basic" mode="basic-kerberos"/>
    </dfhwsse_configuration>
  </wsse_handler>
</service_handler_list>
```

A sample invocation of the **VERIFY TOKEN** command that uses COBOL is shown in Example 6-7.

Example 6-7 Sample COBOL invocation of the VERIFY TOKEN command

```
EXEC CICS VERIFY TOKEN(WS-KERBEROS-TOKEN)
          TOKENLEN(WS-LENGTH)
          KERBEROS
          BASE64
          ESMREASON(WS-ESMREAS)
          ESMRESP(WS-ESMRESP)
          RESP(WS-RESP)
```

For more information about Kerberos support that is provided by CICS TS, see the topic “Kerberos support” in IBM Knowledge Center at this website:

<https://ibm.biz/Bd4YvQ>

6.6.1 Kerberos test configuration

A high-level topology diagram of the test application is shown in Figure 6-6.

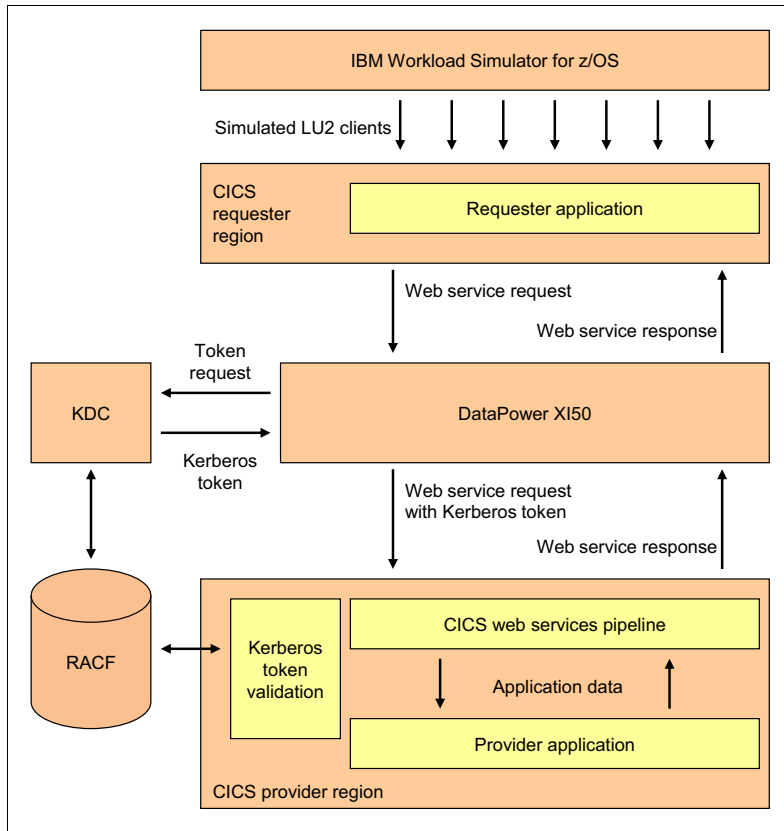


Figure 6-6 Topology of Kerberos performance test application

A CICS *requester* region is used to create web services request messages. This requester region runs transactions that are started over LU2, where IBM Workload Simulator for z/OS (Workload Simulator) provides several simulated clients that can be configured to provide requests at a consistent, but adjustable transaction rate.

The transactions run in the requester region issue web services requests that are directed to an IBM WebSphere DataPower® Integration Appliance XI50 for zEnterprise.

The DataPower appliance requests a Kerberos token from a Kerberos key distribution center (KDC), which is hosted in z/OS. The Kerberos token is obtained by using information that is retrieved from RACF.

After a Kerberos token is obtained, it is inserted into the received web service request and the modified request is forwarded to the CICS web services *provider* region under test.

The CICS web services pipeline handler logic is configured to validate the Kerberos token by using the configuration that is shown in Example 6-7 on page 92. This pipeline processing involves the CICS provider region passing the received Kerberos token to RACF for validation.

After the Kerberos token is validated, control is passed to the user application in the CICS provider region with the received data supplied by using the CICS channels interface.

The user application performs a trivial processing operation and then returns the application response as binary data to the CICS web services pipeline in the DFHWS-DATA container. The web services pipeline converts the binary data response to a well-formed XML response that is returned to the DataPower appliance.

Finally, the DataPower appliance returns the web services response back to the CICS requester region.

6.6.2 Kerberos performance results

To verify the Kerberos token within the CICS provider pipeline, RACF calls the KDC to verify the token. Therefore, in addition to the CPU cycles that are consumed by the CICS provider address space, CPU cycles are consumed by the KDC address space. This publication focuses on the CICS overhead and does not account for this extra CPU overhead in the performance results.

Figure 6-7 plots the measured CPU consumption for a web service request rate, with and without a Kerberos token included in the message body.

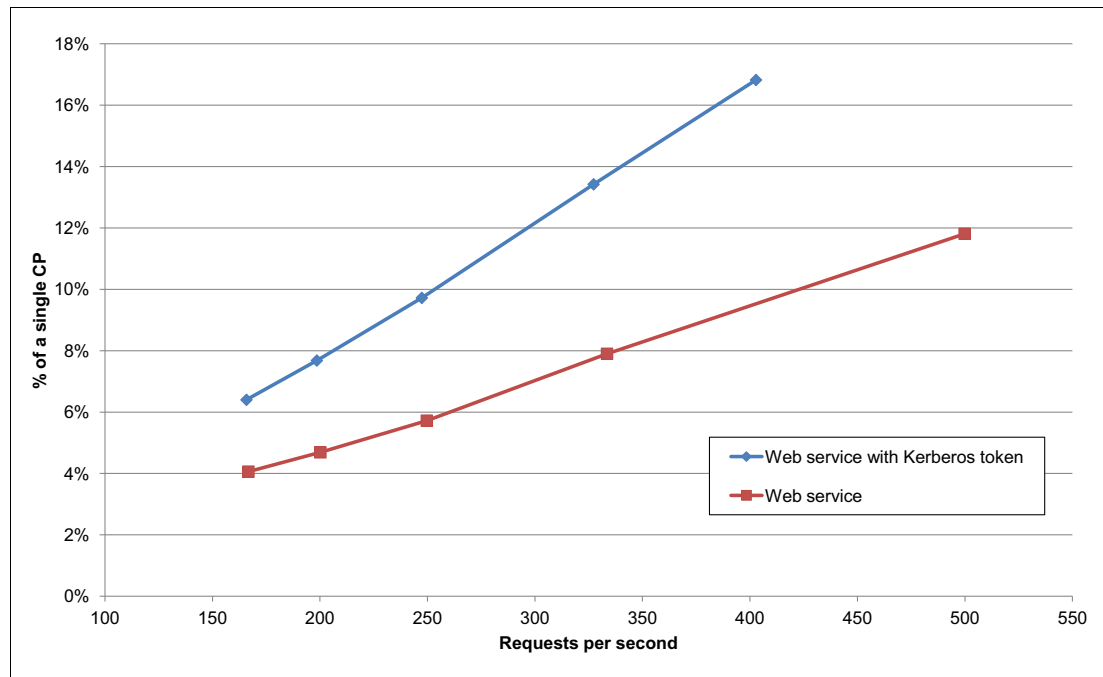


Figure 6-7 Plot of CPU usage for web service workload with and without Kerberos token

6.6.3 Kerberos support conclusion

By using the data that is shown in Figure 6-7, you can see that processing a Kerberos token for each web service request added a burden of approximately 0.165 ms of CPU. This extra usage remained constant as the workload increased.

In this scenario, a maximum throughput of 410 web service requests per second was achieved with a Kerberos token, which is compared to over 500 requests per second without a token. This reduction in throughput was primarily due to usage involved with these processes:

- ▶ DataPower obtaining the Kerberos token
- ▶ CICS using RACF to verify the Kerberos token

6.7 JSON support

CICS TS V5.2 includes support for JSON data mapping, originally released as the CICS Transaction Server Feature Pack for Mobile Extensions V1.0. This support can be used by configuring a web service to use a JSON data stream, or by invoking the JSON transformer linkable interface using the correct CICS container data structures. The JSON data mapping support is implemented in Java and uses a CICS JVM server that is configured with Axis2 services to provide the required runtime environment.

For more information about the use of the JSON support in a web services environment, see the topic “Getting started with JSON web services” in IBM Knowledge Center at this website:

<https://ibm.biz/Bd4Ym2>

For more information about the use of the JSON support as an API, see the topic “JSON transformer linkable interface containers” in IBM Knowledge Center at this website:

<https://ibm.biz/Bd4Ymf>

6.7.1 JSON support test configuration

The performance test application uses the pipeline interface to determine the cost per request of the JSON data mapping support.

A JSON request is accepted into the CICS provider region on a TCP/IP port by using the HTTP transport and is processed by the CICS pipeline handler mechanism. The pipeline handler transforms the received JSON payload into a binary data structure as defined by the COBOL copybook that is used when the bind files are created. The *.wsbind files are created by using the *bottom-up* approach of taking a language copybook and generating a JSON schema and wsbind file.

The pipeline transforms the JSON request payload into a binary data structure by using CICS supplied code that runs in a Java virtual machine (JVM). The converted binary data structure is passed to the user application as containers within a channel. When the user application is completed, the output binary data structure is converted back to a JSON response. The conversion process to a JSON data stream is again implemented as Java code that runs in a CICS JVM server. The use of this Java processing code is specified in the pipeline configuration file.

Example 6-8 shows sample input to the **DFHCSDUP** utility that is used to create the JVMSERVER CSD definition that is required for enabling JSON mapping services support.

Example 6-8 Sample JVMSERVER definition for JSON support

```
DEFINE JVMSERVER(JSONJVM)
  GROUP(GJSON)
  JVMPROFILE(DFHJVMAX)
  THREADLIMIT(50)
```

The JVMSERVER name is referenced by the pipeline configuration XML file (see the sample XML configuration file in Example 6-13 on page 97). The definition references the JVM profile file named DFHJVMAX.jvmprofile. CICS determines the hierarchical file system (HFS) location of this file by using the JVMPROFILEDIR SIT parameter.

For more information about the attributes that are available on a JVMSERVER resource definition, see the topic “JVMSERVER attributes” in IBM Knowledge Center at this website:

<https://ibm.biz/Bd4YmM>

The CICS supplied sample JVM profile file was amended to specify system-specific values for the JAVA_HOME and WORK_DIR parameters. The following parameters were also specified to fix the Java heap sizes to minimize variation in performance results:

- ▶ -Xmx400M
- ▶ -Xms400M
- ▶ -Xmnx120M
- ▶ -Xmns120M
- ▶ -Xmox280M
- ▶ -Xmos280M

A TCP/IP port was opened in CICS that was dedicated to the JSON endpoint. Example 6-9 shows sample input to the DFHCSDUP utility that was used to create the TCPIPSERVICE CSD definition that is required for opening a port in CICS.

Example 6-9 Sample TCPIPSERVICE definition for JSON support

```
DEFINE TCPIPSERVICE(JSONTCP1)
      GROUP(GJSON)
      PORTNUMBER(6000)
      TRANSACTION(CWXN)
      PROTOCOL(HTTP)
      URM(DFHWBAAX)
      IP(1.2.3.4)
      BACKLOG(250)
```

The CWXN value for the TRANSACTION attribute specifies the transaction to run when an HTTP request is received on the TCP/IP port that matches the URIMAP pattern as specified in the construction of the *.wsbind files. Example 6-14 on page 98 shows an example of creating a *.wsbind file. The URM value of DFHWBAAX specifies the default HTTP analyzer program.

For more information about attributes that are available on a TCPIPSERVICE resource definition, see the topic “TCPIPSERVICE attributes” in IBM Knowledge Center at this website:

<https://ibm.biz/Bd4YmS>

By default, work is accepted by the CSOL (socket listener) transaction, which starts a CWXN (web attach) transaction. The CWXN transaction then creates a CPIH (pipeline handler) transaction to process the request. For improved separation of work, you can use an alternative transaction ID rather than CPIH.

Example 6-10 shows an alternative transaction ID: JPIH. This alternative transaction ID provides the same functions as the CICS supplied CPIH transaction, but enables CPU cost from this JSON workload to be separated by filtering on transaction ID.

Example 6-10 Sample TRANSACTION definition for JSON support

```
DEFINE TRANSACTION(JPIH)
      GROUP(GJSON)
      PROGRAM(DFHPIDSH)
      TRANCLASS(JSONTCLH)
      SPURGE(YES)
      TASKDATALOC(ANY)
      DESCRIPTION(JSON HTTP inbound router)
```

A transaction class value of JSONTCLH is specified in Example 6-10 that is used to restrict the number of JPIH transactions that are attached at any one time within the CICS region.

Example 6-11 shows a sample definition for the JSONTCLH transaction class.

Example 6-11 Sample TRANCLASS definition for JSON support

```
DEFINE  TRANCLASS(JSONTCLH)
        GROUP(GJSON)
        MAXACTIVE(100)
        PURGETHRESH(NO)
```

A pipeline is configured in CICS by combining two definitions: The PIPELINE resource and an XML file in HFS.

Example 6-12 shows sample input to the **DFHCSDUP** utility that is used to create the PIPELINE CSD definition that is required to enable JSON mapping services support.

Example 6-12 Sample PIPELINE definition for JSON support

```
DEFINE  PIPELINE(JSONPIP1)
        GROUP(GJSON)
        CONFIGFILE(/path/to/config/file/jsonjavaprovider.xml)
        SHELF(/var/cicsts/myapplid/)
        WSDIR(/wsdir_prefix/wsbind)
```

The CONFIGFILE attribute specifies a fully qualified HFS file name for the XML configuration file. The SHELF attribute indicates the HFS directory that contains the in-use wsbind files. The WSDIR attribute indicates the HFS directory that contains the *.wsbind files that are used to determine the mapping between JSON and the native data structures.

For more information about the attributes that are available on a PIPELINE resource definition, see the topic “PIPELINE attributes” in IBM Knowledge Center at this website:

<https://ibm.biz/Bd4Ymm>

Example 6-13 shows the pipeline XML configuration file that was used in the test application. By using the resource definition in Example 6-12, the following fully qualified XML configuration file name is available:

/path/to/config/file/jsonjavaprovider.xml

Example 6-13 Sample pipeline configuration file to use JSON data mapping support

```
<?xml version="1.0" encoding="EBCDIC-CP-US"?>
<provider_pipeline xmlns="http://www.ibm.com/software/htp/cics/pipeline">
  <service>
    <terminal_handler>
      <cics_json_handler_java>
        <jvmserver>JSONJVM</jvmserver>
      </cics_json_handler_java>
    </terminal_handler>
  </service>
  <apphandler_class>com.ibm.cicsts.axis2.CICSAxis2ApplicationHandler</apphandler_class>
</provider_pipeline>
```

The JSON handler is configured to use the CICS JVMSERVER resource named JSONJVM as defined in Example 6-8 on page 95. A sample XML configuration file for the JSON pipeline handler is supplied in the following location relative to the CICS installation root directory:

```
./samples/pipelines/jsonjavaprovider.xml
```

As described in 6.7.1, “JSON support test configuration” on page 95, the wsbind files were generated by using a *bottom-up* approach. A copybook was used to generate the wsbind and JSON schema files.

Example 6-14 shows a snippet of the JCL that runs the **DFHLS2JS** utility to generate one of the wsbind files.

Example 6-14 Extract of JCL used to generate wsbind files

```
//CABASIC EXEC DFHLS2JS,
//  JAVADIR='java6_31/J6.0',
//  USSDIR='cics690',
//  PATHPREF='',
//  TMPDIR='/tmp',
//  TMPFILE='LS2JS'
//INPUT.SYSUT1 DD *
JSON-SCHEMA-REQUEST=/schema_path/CABASIC-req.schema
JSON-SCHEMA-RESPONSE=/schema_path/CABASIC-resp.schema
LANG=COBOL
LOGFILE=/log_path/LS2JS_CABASIC.log
MAPPING-LEVEL=3.0
PDSLIB=h1q.COPY           <- PDS containing COPY members
PGMINT=COMMAREA          <- Application interface
PGMNAME=CABASIC          <- Program name to invoke
REQMEM=BASICQ            <- COPY member for request structure
RESPMEM=BASICP           <- COPY member for response structure
TRANSACTION=JPIH         <- Pipeline transaction
URI=JSON/CABASIC         <- PATH attribute of generated URIMAP
WSBIND=/wsbind_path/CABASIC.wsbind <- Output wsbind file
/*
```

For more information about the use of the DFHLS2JS utility and the input parameters, see the topic “DFHLS2JS: High-level language to JSON schema conversion for request-response services” in IBM Knowledge Center at this website:

<https://ibm.biz/Bd4Ymn>

The receiving CICS region used the channel interface to communicate with a COBOL back-end application. The application contained no business logic and generated only a trivial response. The use of a minimal back-end program enables you to understand the base cost of processing a JSON request in a CICS region.

The inbound request contains 32 bytes of application data, which corresponds to 180 bytes of JSON.

Example 6-15 shows a sample JSON request. Formatting is added for clarity; the request as received by CICS does not contain white space.

Example 6-15 Sample JSON input for mobile workload

```
{
  "CABASICOperation" : {
    "count_in" : 32,
    "count_out": 1
  }
}
```

The request JSON is produced to populate the input fields that match with the COBOL copybook that is shown in Example 6-16.

Example 6-16 Sample COBOL copybook for mobile workload request

```
05 COUNT-IN    PIC 9(8) COMP-4.
05 COUNT-OUT   PIC 9(8) COMP-4.
05 FILLER      PIC X(24).
```

The application request can result in one of the following four possible responses:

- ▶ 32 bytes of user data (103 bytes of JSON)
- ▶ 1,024 bytes of user data (1,638 bytes of JSON)
- ▶ 4,096 bytes of user data (6,342 bytes of JSON)
- ▶ 16,384 bytes of user data (25,159 bytes of JSON)

Multiple response sizes were achieved by creating multiple endpoints. The endpoints were defined by creating four different `wsbind` files: One for each of the possible response sizes. The four `wsbind` files were created by using multiple invocations of the `DFHLS2JS` utility, each specifying a different copybook for the `RESPMEM` parameter.

Example 6-17 shows a fragment of a sample JSON response. Formatting is added for readability; to reduce data transmission burden, the response that is generated by CICS does not contain white spaces.

Example 6-17 Sample fragment of JSON response

```
{
  "CABASICOperationResponse":{
    "recv_size":32,
    "send_size":1024,
    "taskid":44,
    "trandid":JPIH,
    "user_data":[
      {"user_data":"0001-ABCDEFGHIJKLMNOPQRSTUVWXYZ-"},
      {"user_data":"0002-ABCDEFGHIJKLMNOPQRSTUVWXYZ-"},
      ...
      {"user_data":"0031-ABCDEFGHIJKLMNOPQRSTUVWXYZ-"}
    ]
  }
}
```

The JSON response in Example 6-17 represents a response size of 1 KB of user data. Several lines of response were omitted for clarity. The corresponding copybook is shown in Example 6-18.

Example 6-18 Sample COBOL copybook for response

```

05 RECV-SIZE  PIC 9(8) COMP-4.
   05 SEND-SIZE PIC 9(8) COMP-4.
   05 TASKID   PIC 9(8) COMP-4.
   05 TRANID   PIC X(4).
   05 FILLER   PIC X(16).
   05 USER-DATA PIC X(32) OCCURS 31 TIMES.

```

The varying response size was achieved by having multiple copybooks, where the OCCURS clause for the USER-DATA field was updated to reflect the required size of response.

6.7.2 Scalability as a function of response size

To determine the scalability of the JSON pipeline handler as the payload increases, the workload was run by using a range of response sizes. As described in 6.7.1, “JSON support test configuration” on page 95, the input request message consisted of 32 bytes of user data, which corresponds to 180 bytes of JSON data.

The workload ran at a rate of approximately 290 requests per second, with the response size that varied from 32 bytes to 16 KB of user data. Overall CPU consumption figures are taken from RMF data and are listed in Table 6-11. The “GCP cost” values represent work that is not eligible for offload, and “zIIP cost” values represent workload that is eligible for offload to a specialty engine.

Table 6-11 Average CPU cost per request as response size increases

Response size	GCP cost (ms)	zIIP cost (ms)	Total (ms)
32 bytes	0.529	0.084	0.613
1 KB	0.627	0.245	0.872
4 KB	0.619	0.907	1.525
16 KB	0.643	3.988	4.630

The data in Table 6-11 is plotted in Figure 6-8 on page 101.

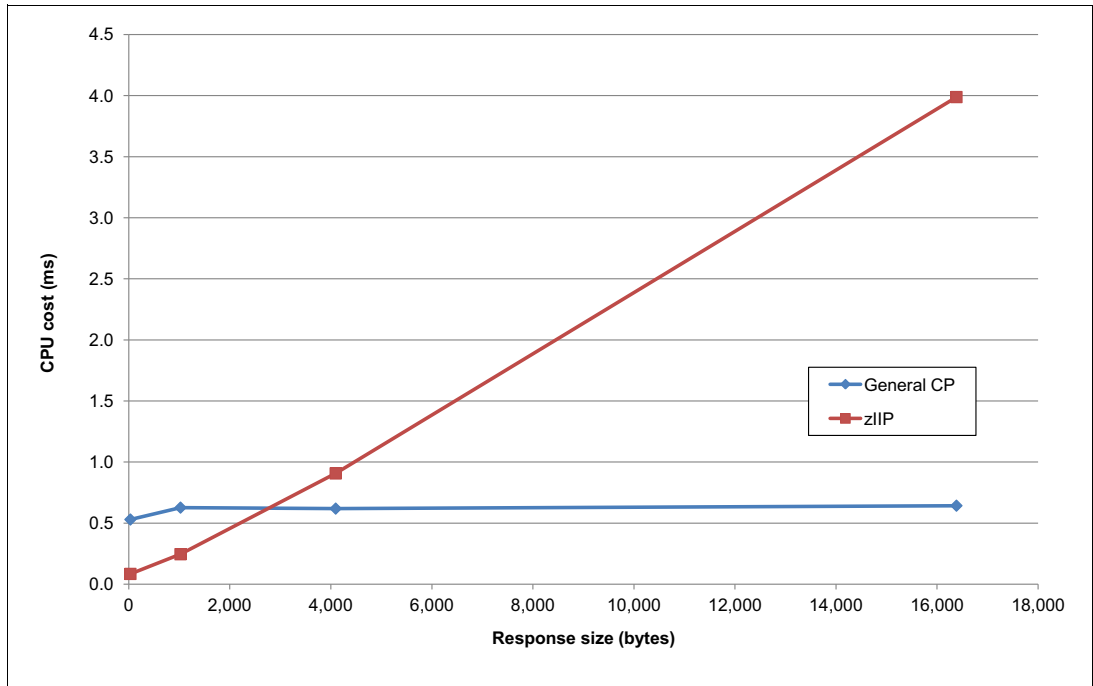


Figure 6-8 Plot of CP cost per request against response size

As the response size increases, the non-zIIP-eligible CP usage remains constant. It is also clear that as the response size increases, the zIIP-eligible CP usage increases linearly.

6.7.3 Scalability as a function of request rate

To determine the scalability of the JSON pipeline handler as throughput increases, the workload was run with a range of inbound request rates. As described in 6.7.1, “JSON support test configuration” on page 95, the input request consisted of 32 bytes of user data, which corresponds to 180 bytes of JSON data.

The workload used the 1 KB of user data response size configuration (a response of 1,638 bytes of JSON), and the request rate varied from approximately 300 - 3,300 requests per second. Overall CPU consumption figures are taken from RMF data and are listed in Table 6-12. The “% of single general CP” values represent work that is not eligible for offload, and “% of single zIIP” values represent workload that is eligible for offload to a specialty engine.

Table 6-12 CP utilization as a function of request rate

Requests per second	% of single general CP	% of single zIIP
334.36	19.69%	6.55%
499.66	29.52%	9.13%
999.77	59.48%	17.56%
1995.16	118.68%	34.62%
3315.31	196.36%	70.20%

The values in Table 6-12 are plotted Figure 6-9 on page 102.

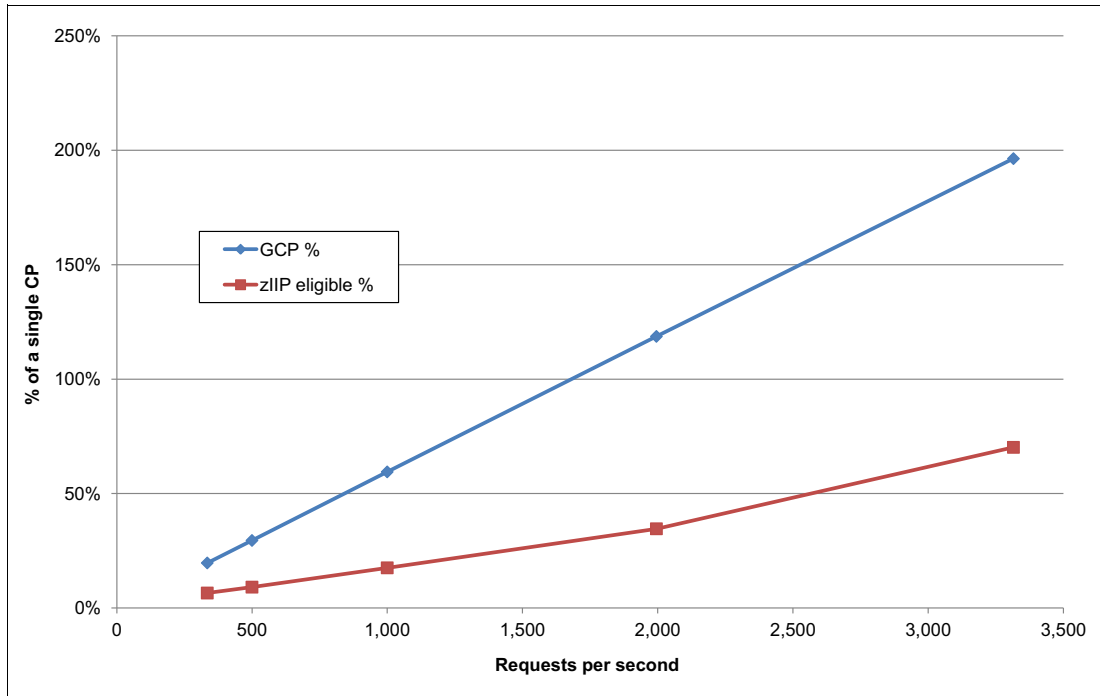


Figure 6-9 Plot of total GCP and zIIP usage against request rate

The straight lines for GCP and zIIP usage demonstrate linear scalability of CPU usage in the workload. The ratio of non-eligible to eligible CPU consumption remains constant as the transaction rate increases.

6.7.4 JSON support conclusion

Section 6.7.2, “Scalability as a function of response size” demonstrated good scalability of the JSON support functions as payload size increases. Section 6.7.3, “Scalability as a function of request rate” demonstrated good scalability as transaction rate increased.

The experience that was gained while performance testing the JSON support infrastructure showed that performance tuning and monitoring of the JSON support functions is a similar process to managing an HTTP XML web services workload within CICS. Many of the resources that are required serve the same purpose in SOAP over HTTP and JSON over HTTP configurations, with only minor differences in resource definitions observed.

One final observation is related to the situation where it is wanted to throttle work arriving in the JVM server. When measuring CPU consumption, adding the pipeline handler transaction (JPIH) to a transaction class was slightly more efficient than the use of the THREADLIMIT attribute of the JVMSERVER resource.

6.8 Java applications and trace

During the CICS V5.2 development cycle, a review of all the trace points in the direct to CICS domain resulted in many of these trace points moving from level 1 to level 2 trace. As a consequence of these changes, the performance of a Java application is improved when running CICS with default level 1 trace enabled.

The first workload that was used to demonstrate the performance improvements is the CICS *Hello World* Java sample. This simple application is started at a console and echoes back a simple string to confirm that execution completed successfully.

For more information about the use of the Hello World Java sample application, see the topic “Running the Hello World examples” in IBM Knowledge Center at this website:

<https://ibm.biz/Bd4YWr>

The second workload that was used to demonstrate these performance improvements is a Java application that issues 120 file read operations by using the JCICS interface.

Figure 6-10 plots the CPU cost per transaction of the simple Hello World Java workload in the following configurations:

- ▶ CICS TS V5.1 with trace disabled
- ▶ CICS TS V5.1 with default level 1 trace enabled
- ▶ CICS TS V5.2 with default level 1 trace enabled

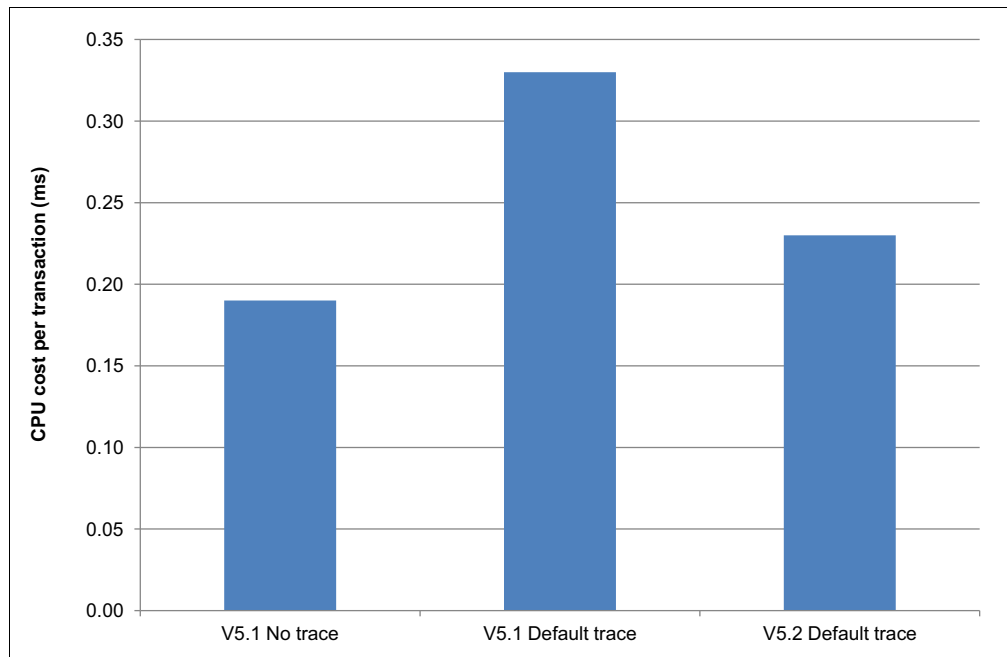


Figure 6-10 Plot of CPU per transaction for the Hello world Java workload

Figure 6-10 shows a clear reduction in CPU consumption in CICS TS V5.2 when compared to the same configuration that uses CICS TS V5.1. The same three configurations were studied for the file read Java workload, with the results shown in Figure 6-11 on page 104.

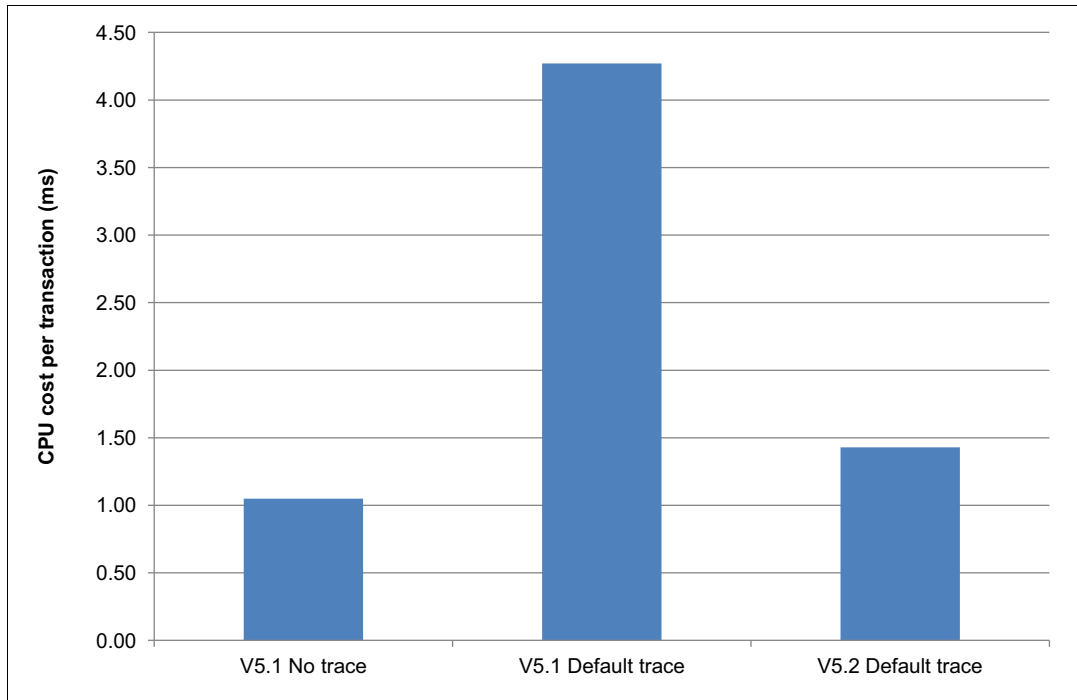


Figure 6-11 Plot of CPU per transaction for the file read Java workload

As with the Hello World Java workload, there is a clear reduction in CPU consumption per transaction when CICS TS V5.2 is used. The results of both of these workloads are listed in Table 6-13.

Table 6-13 Summary of CPU cost per transaction with varying trace and release configurations

Workload	V5.1 no trace	V5.1 default trace	V5.2 default trace
Hello world	0.19 ms	0.33 ms	0.23 ms
File read	1.05 ms	4.27 ms	1.43 ms

Most Java workloads that run with trace enabled in CICS TS V5.1 see a reduction in CPU when upgrading to CICS TS V5.2.

In CICS TS V5.2, enabling trace for a Java application results in a CPU overhead that is approximately equal to the CPU overhead that is incurred by enabling trace for an equivalent non Java application.

6.9 Web services over HTTP improvements

Performance improvements were delivered in CICS TS V5.2 for the scenario where CICS acts as a web services provider for HTTP requests. The improvements are a reduction in real storage usage and a reduction in CPU usage. This section quantifies the performance benefits by using sample workloads.

6.9.1 Web services real storage usage

For every inbound web service request, the amount of real storage that is used was reduced significantly. This reduction in real storage is manifested as a reduction in 31-bit virtual storage usage. Increased efficiency in the code page conversion processing reduces the amount of storage that is required, which enables CICS regions to process many concurrent requests.

Figure 6-12 shows the amount of 31-bit virtual storage that is used by a 1 MB web service request into CICS.

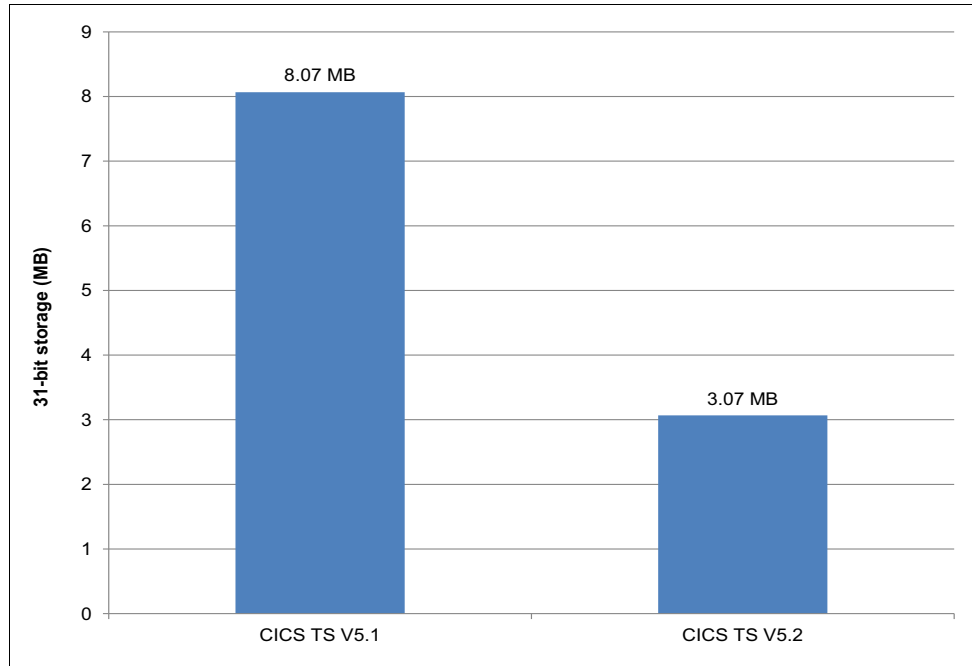


Figure 6-12 Storage usage for a 1 MB web service request with CICS as a provider

This reduction in storage usage can enable a CICS region to process a higher request rate for a specific configuration, or it might allow a consolidation exercise that reduces the number of CICS regions that are required to satisfy a workload.

6.9.2 Web services CPU reduction

For inbound requests into CICS as an HTTP web services provider, the number of TCB change mode operations that are required to fulfil the request was reduced. This reduction in TCB change mode operations provides a small performance gain in CPU for most workloads.

Three XML web services were configured to accept requests that consisted of a single XML element of 8 bytes, 32 KB, or 256 KB. The provider application returned a single 8-byte XML element as a response.

The total cost to process the web service request was calculated by using RMF data, which includes the CSOL, CWXN, and CPIH transactions. The results are shown in Figure 6-13.

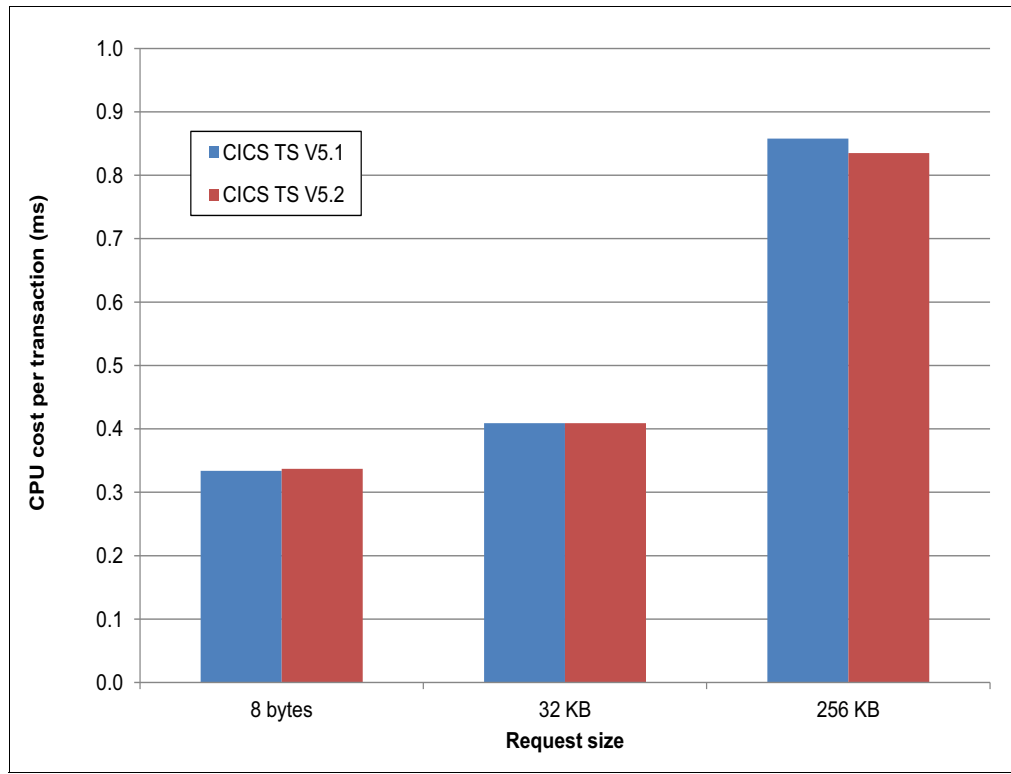


Figure 6-13 CPU cost per request for varying request sizes

For the largest message sizes, a small reduction in CPU cost per request is observed. The CPU costs for smaller message sizes remain equivalent when comparing CICS TS V5.1 with V5.2.

6.10 Java 7.0 and Java 7.1

By using the same workload and methodology that is described in 6.2.5, “Java servlet that uses JDBC and VSAM” on page 83, the performance of Java 7.0 and Java 7.1 was compared. For both scenarios, CICS TS V5.2 was used with Java 7.0 SR7 or Java 7.1 SR1.

Table 6-14 on page 107 lists the performance measurements when running the workload with Java 7.0 SR7.

Note: APAR PI52819 enables support for Java 8 in CICS TS V5.2.

Table 6-14 Java 7.0 SR7 results for the Java servlet JDBC and VSAM workload

ETR	CICS CPU			CPU per request (ms)
	Not zIIP-eligible	zIIP-eligible	Total	
200	20.82%	29.42%	50.24%	2.512
400	42.34%	60.12%	102.46%	2.562
800	86.46%	120.38%	206.85%	2.586

Table 6-15 lists the performance measurements when running the workload with Java SR1.

Table 6-15 Java 7.1 SR1 results for the Java servlet JDBC and VSAM workload

ETR	CICS CPU			CPU per request (ms)
	Not zIIP-eligible	zIIP-eligible	Total	
200	21.44%	29.70%	51.15%	2.558
400	43.55%	61.13%	104.68%	2.617
800	88.95%	124.12%	213.07%	2.663

The performance measurements in Table 6-14 and Table 6-15 are shown in Figure 6-14.

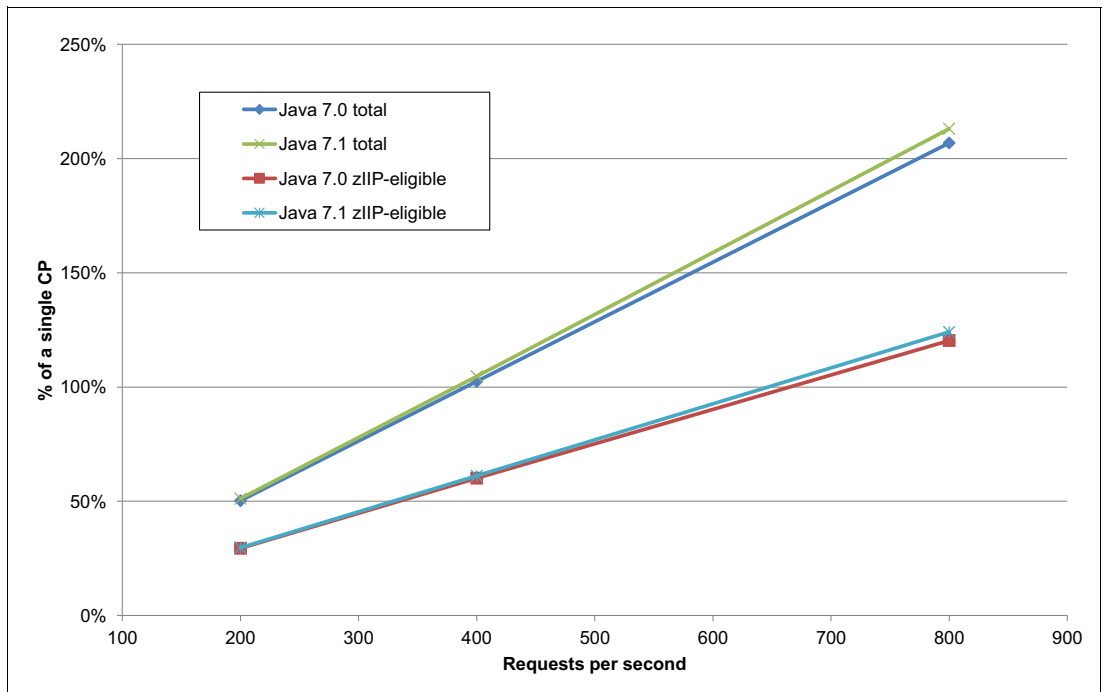


Figure 6-14 Plot of Java 7.0 and Java 7.1 performance results for JDBC and VSAM workload

For this workload, Java 7.0 delivers slightly better performance than Java 7.1, which provides a decrease of approximately 2% in terms of CPU usage per transaction.

It can be observed that Java 7.1 provides the same linear scaling characteristics as demonstrated by the Java 7.0 run time.



CICS TS for z/OS V5.3

The CICS Transaction Server for z/OS (CICS TS) V5.3 release introduces a significant number of performance improvements. Included in the CICS TS V5.3 performance report are the following subject areas:

- ▶ Key performance benchmarks that are presented as a comparison against the CICS TS V5.2 release.
- ▶ An outline of improvements made regarding the threadsafe characteristics of the CICS run time.
- ▶ Details of the changes that are made to performance-critical CICS initialization parameters, and the effect of these updates.
- ▶ Description of all the updated statistics and monitoring fields.
- ▶ Benchmarks that document improvements in XML and JavaScript Object Notation (JSON) web services.
- ▶ A description of how CICS can protect itself from unconstrained resource demand from inbound HTTP requests.
- ▶ High-level views of new functionality that was introduced in the CICS TS V5.3 release, including performance benchmark results where appropriate.

This chapter includes the following topics:

- ▶ 7.1, “Introduction” on page 110
- ▶ 7.2, “Release-to-release comparisons” on page 110
- ▶ 7.3, “Improvements in threadsafety” on page 112
- ▶ 7.4, “Changes to system initialization parameters” on page 114
- ▶ 7.5, “Enhanced instrumentation” on page 115
- ▶ 7.6, “Low-level CICS optimizations” on page 118
- ▶ 7.7, “Web support and web service optimization” on page 122
- ▶ 7.8, “Java workloads” on page 123
- ▶ 7.9, “Java 8 performance” on page 127
- ▶ 7.10, “Simultaneous multithreading with Java workloads” on page 130
- ▶ 7.11, “Reporting of CPU time to z/OS Workload Manager” on page 133
- ▶ 7.12, “z/OS Connect for CICS” on page 134
- ▶ 7.13, “HTTP flow control” on page 140

- ▶ 7.14, “High transaction rate performance study” on page 143
- ▶ 7.15, “WebSphere Liberty zIIP eligibility” on page 143
- ▶ 7.16, “Link to WebSphere Liberty” on page 144

7.1 Introduction

When the results were compiled for this chapter, the workloads were run on an IBM z13[®] model NE1 (machine type 2964). A maximum of 32 dedicated central processors (CPs) were available on the measured logical partition (LPAR), with a maximum of 4 dedicated CPs available to the LPAR that was used to simulate users. These LPARs are configured as part of a Parallel Sysplex. An internal coupling facility was co-located on the same central processor complex (CPC) as the measurement and driving LPARs. They were connected by using internal coupling peer (ICP) links. An IBM System Storage DS8870 (machine type 2424) was used to provide external storage.

This chapter presents the results of several performance benchmarks when run in a CICS TS for z/OS V5.3 environment. Unless otherwise stated in the results, the CICS V5.3 environment was the code that was available at general availability (GA) time. Several of the performance benchmarks are presented in the context of a comparison against CICS TS V5.2. The CICS TS V5.2 environment contained all PTFs that were issued before 10 March 2015. All LPARs used z/OS V2.1.

For more information about performance terms that are used in this chapter, see Chapter 1, “Performance terminology” on page 3. For more information about the test methodology that was used, see Chapter 2, “Test methodology” on page 11. For more information about the workloads that were used, see Chapter 3, “Workload descriptions” on page 21.

Where reference is made to an *LSPR processor equivalent*, the indicated machine type and model can be found in the large systems performance reference (LSPR) document. For more information about obtaining and the use of LSPR data, see 1.3, “Large Systems Performance Reference” on page 6.

7.2 Release-to-release comparisons

This section describes some of the results from a selection of regression workloads that are used to benchmark development releases of CICS TS. For more information about the use of regression workloads, see Chapter 3, “Workload descriptions” on page 21.

7.2.1 Data Systems Workload dynamic routing

The Data Systems Workload (DSW) dynamic routing workload is used in 7.6, “Low-level CICS optimizations” on page 118 to demonstrate several performance benefits that are combined to reduce the overall CPU cost per transaction. For more information about a comparison between CICS TS V5.2 and CICS TS V5.3 performance, see 7.6, “Low-level CICS optimizations” on page 118.

7.2.2 RTW threadsafe

This section presents the performance figures for the threadsafe variant of the IBM Rational[®] Transactional Workload (RTW), as described in 3.3, “Relational Transactional Workload” on page 25.

Table 7-1 lists the results of the RTW threadsafe workload that uses the CICS TS V5.2 release. Table 7-2 lists the same figures for the CICS TS V5.3 release.

Table 7-1 Performance results for CICS TS V5.2 with RTW threadsafe workload

ETR	CICS CPU	CPU per transaction (ms)
333.49	45.83%	1.374
499.64	68.29%	1.367
713.32	98.79%	1.385
996.24	138.84%	1.394
1241.42	173.42%	1.397

Table 7-2 Performance results for CICS TS V5.3 with RTW threadsafe workload

ETR	CICS CPU	CPU per transaction (ms)
334.12	46.29%	1.385
500.50	69.16%	1.382
714.30	98.77%	1.383
997.32	139.06%	1.394
1242.71	175.74%	1.414

The average CPU per transaction figure for CICS TS V5.2 is calculated to be 1.383 ms. The CICS TS V5.3 figure is calculated to be 1.392 ms. The difference between these two figures is 0.6%, which is within our measurement accuracy of $\pm 1\%$; therefore, the performance of the two releases is considered to be equivalent.

These figures are shown in Figure 7-1.

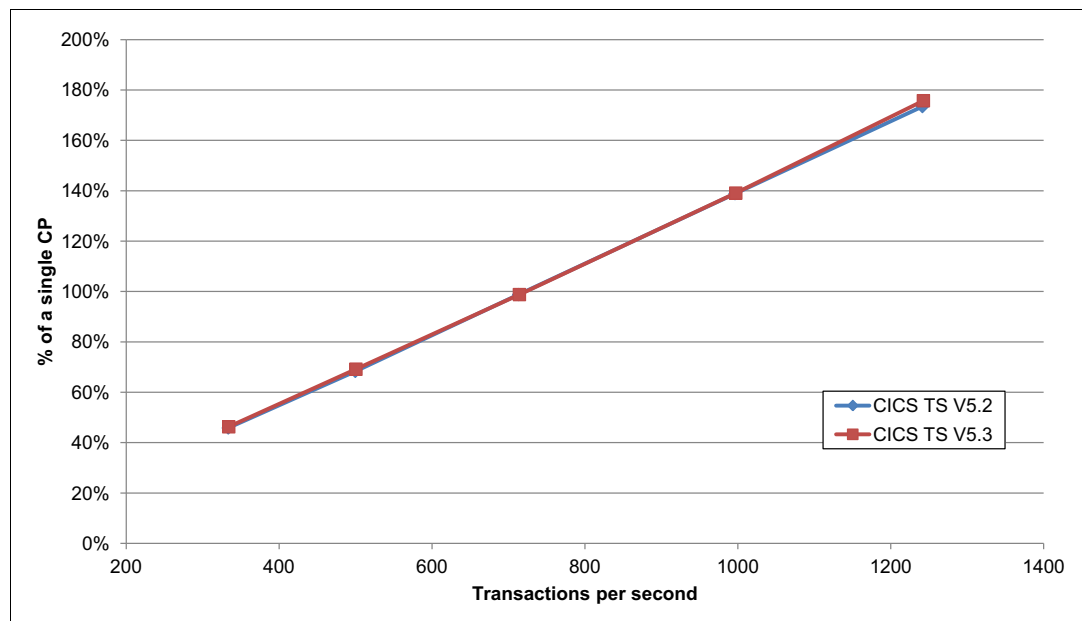


Figure 7-1 Plot of CICS TS V5.2 and V5.3 performance results for RTW threadsafe workload

As shown in Figure 7-1 on page 111, the lines are straight, which indicates linear scaling as transaction throughput increases. The lines also are overlaid, which indicates equivalent performance when the releases are compared.

7.3 Improvements in threadsafety

All new CICS API commands in CICS V5.3 are threadsafe. Also, some system programming interface (SPI) commands were made threadsafe in this release. There also were some specific functional areas that were improved to reduce task control block (TCB) switches.

7.3.1 Threadsafe API and SPI commands

The following new CICS API commands are threadsafe:

- ▶ REQUEST PASSTICKET
- ▶ CHANNEL commands:
 - DELETE CHANNEL
 - QUERY CHANNEL

The **WRITE OPERATOR** CICS API command was made threadsafe.

For more information about CICS API commands, see the “CICS command summary” topic in IBM Knowledge Center at this website:

<https://ibm.biz/Bd427B>

The following CICS SPI commands were made threadsafe:

- ▶ INQUIRE RRMS
- ▶ INQUIRE STORAGE
- ▶ INQUIRE STREAMNAME
- ▶ INQUIRE SUBPOOL
- ▶ INQUIRE TASK LIST
- ▶ INQUIRE TSPool
- ▶ INQUIRE UOWENQ
- ▶ PERFORM SECURITY REBUILD
- ▶ PERFORM SSL REBUILD
- ▶ ENQMODEL commands:
 - INQUIRE ENQMODEL
 - SET ENQMODEL
 - DISCARD ENQMODEL
- ▶ JOURNALMODEL commands:
 - INQUIRE JOURNALMODEL
 - DISCARD JOURNALMODEL
- ▶ JOURNALNAME commands:
 - INQUIRE JOURNALNAME
 - SET JOURNALNAME
 - DISCARD JOURNALNAME

- ▶ TCLASS commands:
 - INQUIRE TCLASS
 - SET TCLASS
- ▶ TCP/IP commands:
 - INQUIRE TCPIP
 - SET TCPIP
- ▶ TCPIPSERVICE commands:
 - INQUIRE TCPIPSERVICE
 - SET TCPIPSERVICE
 - DISCARD TCPIPSERVICE
- ▶ TDQUEUE commands:
 - INQUIRE TDQUEUE
 - SET TDQUEUE
 - DISCARD TDQUEUE
- ▶ TRANCLASS commands:
 - INQUIRE TRANCLASS
 - SET TRANCLASS
 - DISCARD TRANCLASS
- ▶ TSMODEL commands:
 - INQUIRE TSMODEL
 - DISCARD TSMODEL
- ▶ TSQUEUE / TSQNAME commands:
 - INQUIRE TSQUEUE / TSQNAME
 - SET TSQUEUE / TSQNAME
- ▶ UOW commands:
 - INQUIRE UOW
 - SET UOW
- ▶ WEB commands:
 - INQUIRE WEB
 - SET WEB

For more information about CICS SPI commands, see the “System commands” topic in IBM Knowledge Center at this website:

<https://ibm.biz/Bd427x>

7.3.2 Optimizations for SSL support

Several TCB switches were removed for inbound requests that use SSL. For more information about this and other improvements in CICS web support, see *IBM CICS Performance Series: Web Services Performance in CICS TS V5.3*, REDP-5322, which is available at this website:

<http://www.redbooks.ibm.com/abstracts/redp5322.html>

7.3.3 Offloading authentication requests to open TCBs

RACF APAR OA43999 introduced the Enhanced Password Algorithm, which applies to z/OS V1.12, V1.13, and V2.1. This RACF APAR implements the following support:

- ▶ Accept more special characters within passwords
- ▶ Allow stronger encryption of passwords
- ▶ Define users with a password phrase and no password
- ▶ Expire a password without changing it
- ▶ Clean up password history

For more information about the new function APAR, see the following IBM support website:

<http://www.ibm.com/support/docview.wss?uid=isg10A43999>

If the APARs are installed, CICS starts a new callable service IRRSPW00 for password authentication. This service is used for the following authentication operations:

- ▶ Basic authentication requests
- ▶ EXEC CICS VERIFY PASSWORD API command
- ▶ EXEC CICS VERIFY PHRASE API command
- ▶ EXEC CICS SIGNON API command

The IRRSPW00 service runs on open TCBs or switches to an L8 TCB, which reduces contention on the resource-owning (RO) TCB.

Note: The ability to perform authentication requests on an open TCB was also made available to CICS TS V4.2 in APAR PI21865, and CICS TS V5.1 and V5.2 in APAR PI21866.

7.4 Changes to system initialization parameters

Several performance-related CICS system initialization (SIT) parameters were changed in the CICS TS V5.3 release. This section describes changes to the SIT parameters that have the most affect on CICS performance. All comparisons to previous limits or default values refer to CICS TS V5.2.

7.4.1 Storage protection (STGPROT)

Storage protection (SIT parameter STGPROT) is now enabled by default. For more information about storage protection, see the “The storage protection global option” topic in IBM Knowledge Center at this website:

<https://ibm.biz/Bd427M>

7.4.2 Internal trace table size (TRTABSZ)

The default size for the internal trace table (SIT parameter TRTABSZ) increased to 12 MB. For more information about the internal trace facility, see the “Internal trace table” topic in IBM Knowledge Center at this website:

<https://ibm.biz/Bd427G>

Storage for the internal trace table is allocated outside of any CICS DSA. In CICS releases since CICS TS V4.2, the internal trace table is allocated in 64-bit virtual storage.

7.5 Enhanced instrumentation

The CICS TS V5.3 release continues the expansion of information that is reported by the CICS monitoring and statistics component. This section describes the extra fields that are now available in the CICS statistics SMF records.

For more information about changes in monitoring fields across a range of CICS releases, see the “Changes to CICS monitoring” topic in IBM Knowledge Center at this website:

<https://ibm.biz/Bd427b>

7.5.1 The DFHCICS performance group

The number of named counter server GET requests (field NCGETCT) field was added to the DFHCICS performance group. This field shows the total number of requests to a named counter server to satisfy **EXEC CICS GET COUNTER** and **EXEC CICS GET DCOUNTER** API commands that are issued by the user task.

For more information about counters that are available in the DFHCICS performance group, see the “Performance data in group DFHCICS” topic in IBM Knowledge Center at this website:

<https://ibm.biz/Bd427g>

7.5.2 The DFHTASK performance group

The dispatcher allocate pthread wait time (field DSAPHTWT) field was added to the DFHTASK performance group. This field shows the dispatcher allocated pthread wait time. This time is the time that the transaction waited for a WebSphere Liberty *pthread* to be allocated during links to WebSphere Liberty programs.

For more information about counters that are available in the DFHTASK performance group, see the “Performance data in group DFHTASK” topic in IBM Knowledge Center at this website:

<https://ibm.biz/Bd4ZpL>

7.5.3 The DFHTEMP performance group

The following fields were added to the DFHTEMP performance group:

- ▶ Number of shared temporary storage GET operations (field TSGETSCT)
Number of temporary storage GET requests from shared temporary storage that are issued by the user task.
- ▶ Number of shared temporary storage GET operations (field TSPUTSCT)
Number of temporary storage PUT requests to shared temporary storage that are issued by the user task.

The total temporary storage operations (field TSTOTCT) field in the DFHTEMP performance group was updated. This field is the sum of the temporary storage read queue (TSGETCT), read queue shared (TSGETSCT), write queue auxiliary (TSPUTACT), write queue main (TSPUTMCT), write queue shared (TSPUTSCT), and delete queue requests that are issued by the user task.

For more information about counters that are available in the DFHTEMP performance group, see the “Performance data in group DFHTEMP” topic in IBM Knowledge Center at this website:

<https://ibm.biz/Bd427L>

7.5.4 The DFHWEBB performance group

The following fields were added to the DFHWEBB performance group:

- ▶ JSON request body length (field WBJSNRQL)
For JSON web service applications, the JSON message request length.
- ▶ JSON response body length (field WBJSNRPL)
For JSON web service applications, the JSON message response length.

For more information about counters that are available in the DFHWEBB performance group, see the “Performance data in group DFHWEBB” topic in IBM Knowledge Center at this website:

<https://ibm.biz/Bd427T>

7.5.5 Monitoring domain global statistics

The following fields were added to the collected monitoring domain statistics:

- ▶ Total transaction CPU time (field MNGCPU)
The total transaction CPU time that is accumulated for the CICS dispatcher managed TCB modes that are used by the transactions that completed during the interval.
- ▶ Total transaction CPU time on CP (field MNGTONCP)
The total transaction CPU time on a standard processor that is accumulated by the CICS dispatcher managed TCB modes that are used by the transactions that completed during the interval.
- ▶ Total transaction CPU offload on CP (field MNGOFLCP)
The total transaction CPU time on a standard processor but was eligible for offload to a specialty processor (zIIP or zAAP) that was accumulated for the CICS dispatcher that was managed TCB modes used by the transactions that completed during the interval.

A sample DFHSTUP report that contains the new fields is shown in Example 7-1.

Example 7-1 Sample CICS TS V5.3 DFHSTUP monitoring domain global statistics report fragment

Average user transaction resp time. . . :	00:00:00.001256
Peak user transaction resp time . . . :	00:00:00.061583
Peak user transaction resp time at. . . : 11/24/2015	22:25:58.7568
Total transaction CPU time. :	00:00:14.192698
Total transaction CPU time on CP. . . :	00:00:14.192698
Total transaction CPU offload on CP . . :	00:00:00.000000

For more information about monitoring domain statistics, see the “Monitoring domain: global statistics” topic in IBM Knowledge Center at this website:

<https://ibm.biz/Bd42WH>

7.5.6 TCP/IP global statistics

The following fields were added to TCP/IP global statistics:

- ▶ Performance tuning for HTTP connections (field SOG_SOTUNING)
Indicates whether performance tuning for HTTP connections occurs.
- ▶ Socket listener has paused listening for HTTP connections (field SOG_PAUSING_HTTP_LISTENING)
Indicates whether the listener paused listening for HTTP connection requests because the number of tasks in the region reached the limit for accepting new HTTP connection requests.
- ▶ Number of times socket listener notified at task accept limit (field SOG_TIMES_AT_ACCEPT_LIMIT)
Is the number of times the listener was notified that the number of tasks in the region reached the limit for accepting new HTTP connection requests.
- ▶ Last time socket listener paused listening for HTTP connections (field SOG_TIME_LAST_PAUSED_HTTP_LISTENING)
The last time the socket listener paused listening for HTTP connection requests because the number of tasks in the region reached the limit for accepting new HTTP connection requests.
- ▶ Region stopping HTTP connection persistence (field SOG_STOPPING_PERSISTENCE)
Indicates whether the region is stopping HTTP connection persistence because the number of tasks in the region exceeded the limit.
- ▶ Number of times region stopped HTTP connection persistence (field SOG_TIMES_STOPPED_PERSISTENT)
The number of times the region took action to stop HTTP connection persistence because the number of tasks in the region exceeded the limit.
- ▶ Last time stopped HTTP connection persistence (field SOG_TIME_LAST_STOPPED_PERSISTENT)
The last time the region took action to stop HTTP connection persistence because the number of tasks in the region exceeded the limit.
- ▶ Number of persistent connections made non-persistent (field SOG_TIMES_MADE_NON_PERSISTENT)
The number of times a persistent HTTP connection was made non-persistent because the number of tasks in the region exceeded the limit.
- ▶ Number of times disconnected an HTTP connection at max uses (field SOG_TIMES_CONN_DISC_AT_MAX)
The number of times a persistent HTTP connection was disconnected because the number of uses exceeded the limit.

For more information about performance tuning for HTTP connections and a sample DFHSTUP report, see 7.13, “HTTP flow control” on page 140. For more information about TCP/IP global statistics, see the “TCP/IP: Global statistics” topic in IBM Knowledge Center at this website:

<https://ibm.biz/Bd42WY>

7.5.7 URIMAP global statistics

The direct attach count (field WBG_URIMAP_DIRECT_ATTACH) field was added to URIMAP global statistics. This field shows the number of requests that are processed by a directly attached user task.

The direct attach count statistics field was added in support of the web optimizations, as described in 7.7, “Web support and web service optimization” on page 122. For more information about URIMAP global statistics, see the “URIMAP definitions: Global statistics” topic in IBM Knowledge Center at this website:

<https://ibm.biz/Bd42WP>

7.6 Low-level CICS optimizations

The CICS TS V5.3 release includes the following low-level optimizations that can provide a performance benefit to many workloads:

- ▶ Use of the store clock fast (STCKF) hardware instruction that was introduced by the IBM System z9 processor.
- ▶ Storage alignment of some key CICS control blocks to improve the interaction between the CICS TS run time and the hardware cache subsystem.
- ▶ Use of hardware instructions to pre-fetch data into the processor cache, which reduces the number of CPU cycles that are wasted while waiting for data.
- ▶ A reduction in lock contention through tuning the CICS Monitoring Facility algorithms.
- ▶ More efficient algorithms that are used for multiregion operation (MRO) session management.
- ▶ More tuning of other internal procedures.

These improvements in efficiency have particular benefit for CICS trace, CICS monitoring, and for MRO connections that have high session counts.

The remainder of this section describes the results of performance benchmarks that use the DSW workload. For this performance benchmark, two TOR regions were configured to dynamically route transactions to four AOR regions that use CICSplex System Manager. Each AOR function shipped file control requests to an FOR, where VSAM data is accessed in Local Shared Resources (LSR) mode. A more information about the workload, see 3.2, “Data Systems Workload” on page 22.

The following configurations were tested to show the relative benefits of the improvements in each of the monitoring, trace, and MRO session management components:

- ▶ Monitoring and trace enabled
- ▶ Monitoring disabled, trace enabled
- ▶ Monitoring enabled, trace disabled
- ▶ Monitoring and trace disabled
- ▶ Monitoring and trace disabled with low numbers of MRO sessions

Comparisons are made between CICS TS V5.2 and CICS TS V5.3.

7.6.1 Monitoring and trace enabled

For this scenario, performance class monitoring was enabled by using MN=ON and MNPER=ON. Internal trace was enabled with INTTR=ON. All other trace-related SIT parameters used their default values. Figure 7-2 shows the benchmark results for this configuration that uses CICS TS V5.2 and V5.3.

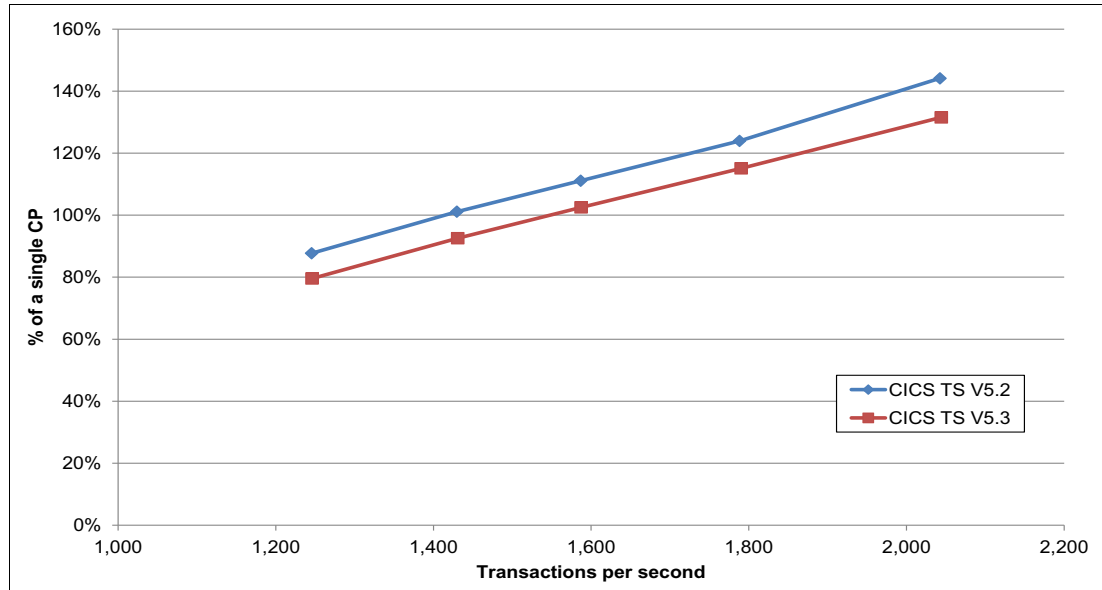


Figure 7-2 DSW performance results with monitoring and trace enabled

The average CPU per transaction for CICS TS V5.2 was 0.702 ms, and the equivalent value for V5.3 was 0.643 ms. For this workload, a reduction of 0.059 ms per transaction represents a decrease of 8%.

The straight lines in the plot indicate that both configurations scale linearly as the transaction rate increases.

7.6.2 Monitoring disabled, trace enabled

This scenario extends the scenario that is described in 7.6.1, “Monitoring and trace enabled” on page 119 by disabling performance class monitoring. Performance class monitoring was disabled by using the SIT parameter MN=OFF. Internal trace was enabled by using INTTR=ON, and all other trace-related SIT parameters used their default values. Figure 7-3 on page 120 shows the results of the benchmark for CICS TS V5.2 and V5.3.

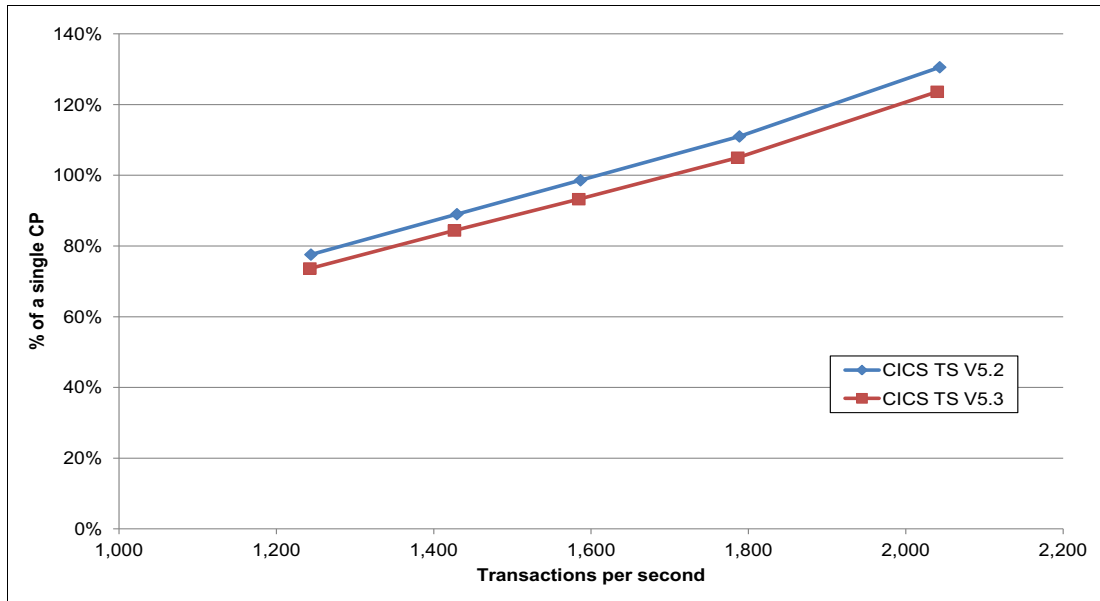


Figure 7-3 DSW performance results with monitoring disabled and trace enabled

Average CPU per transaction for CICS TS V5.2 was 0.625 ms, and the equivalent value for V5.3 was 0.593 ms. A reduction of 0.032 ms per transaction represents a decrease of 5% for this workload.

7.6.3 Monitoring enabled, trace disabled

In this scenario, the configuration is a mirror of the scenario that is described in 7.6.2, “Monitoring disabled, trace enabled” on page 119. In this scenario, performance class monitoring was enabled by using MN=0N and MNPER=0N. Internal trace was disabled with INTTR=OFF and all other trace-related SIT parameters used their default values. Figure 7-4 shows the results of the benchmark results for CICS TS V5.2 and V5.3.

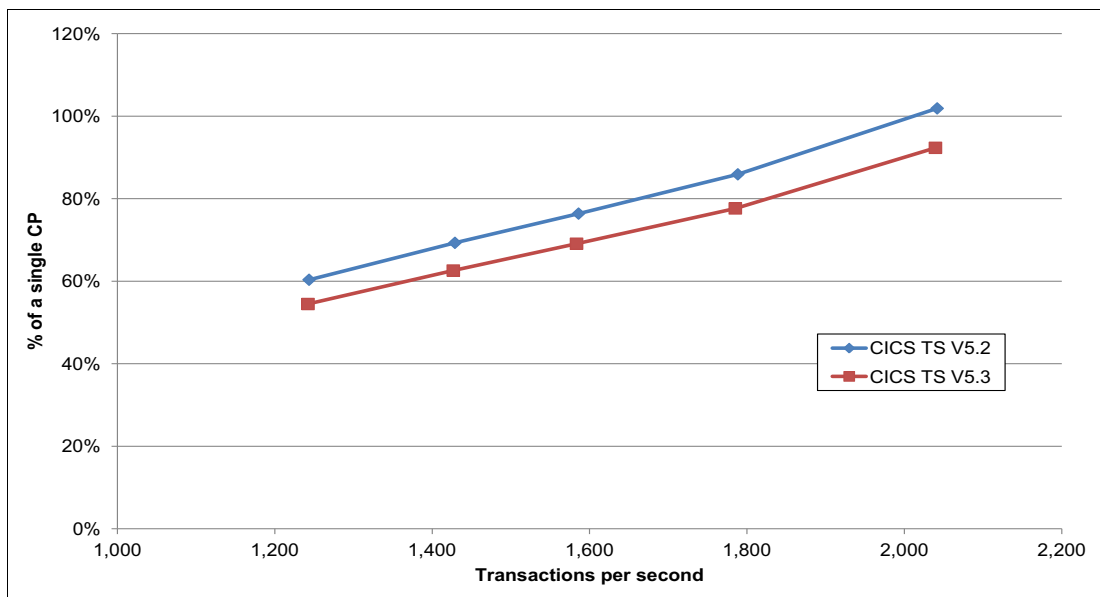


Figure 7-4 DSW performance results with monitoring enabled and trace disabled

Average CPU per transaction for CICS TS V5.2 was 0.486 ms, and the equivalent value for V5.3 was 0.440 ms. A reduction of 0.046 ms per transaction represents a decrease of 9% for this workload.

7.6.4 Monitoring and trace disabled

In this scenario, performance class monitoring and trace were disabled. Performance class monitoring was disabled by using MN=OFF. Internal trace was disabled by setting INTTR=OFF and all other trace-related SIT parameters used their default values. Figure 7-5 shows the results of the benchmark results for CICS TS V5.2 and V5.3.

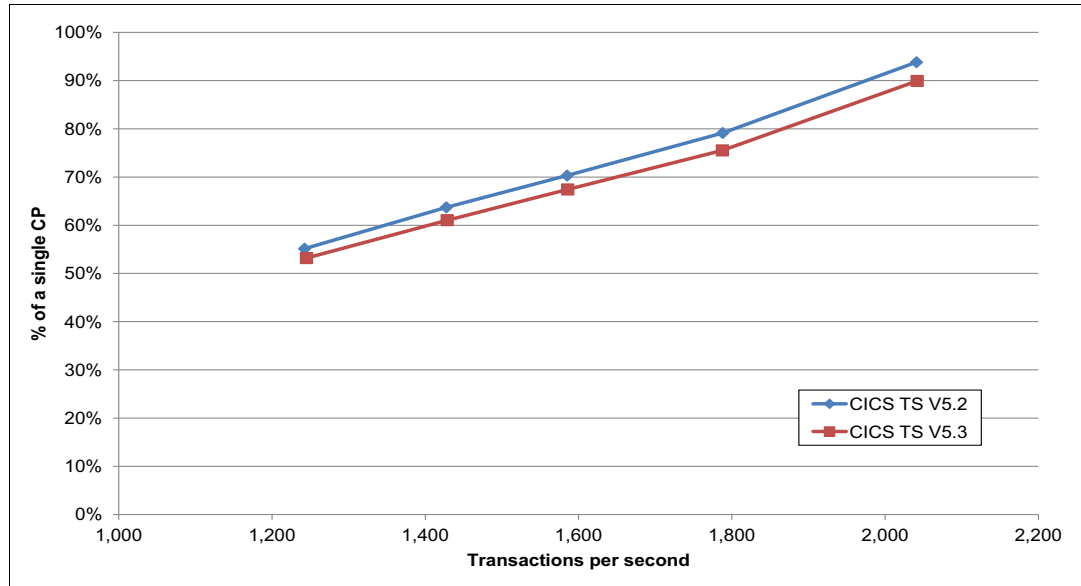


Figure 7-5 DSW performance results with monitoring and trace disabled

Average CPU per transaction for CICS TS V5.2 was 0.447 ms, and the equivalent value for V5.3 was 0.428 ms. A reduction of 0.019 ms per transaction represents a decrease of 4% for this workload.

7.6.5 Monitoring and trace disabled with low numbers of MRO sessions

The final scenario isolates the performance improvements in CICS that are not directly related to monitoring, trace, or MRO session management. Performance class monitoring was disabled by using MN=OFF. Internal trace was disabled with INTTR=OFF and all other trace-related SIT parameters used their default values. All MRO connections were configured to have a minimal number of sessions defined. Figure 7-6 on page 122 shows the benchmark results for CICS TS V5.2 and V5.3.

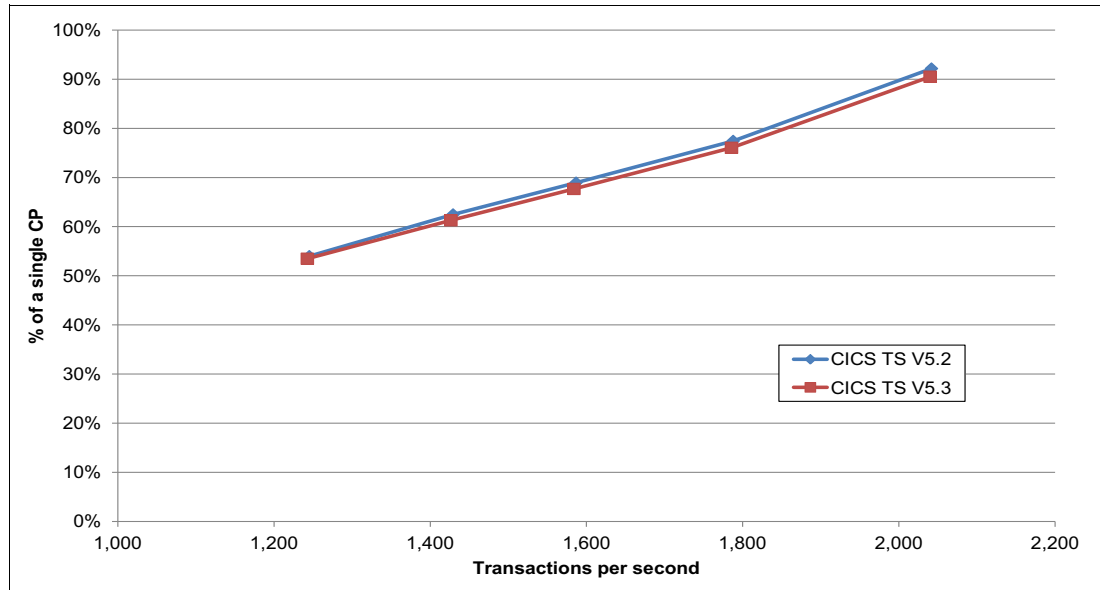


Figure 7-6 DSX performance results with monitoring and trace disabled and low session count

Average CPU per transaction for CICS TS V5.2 was 0.438 ms, and the equivalent value for V5.3 was 0.431 ms. A reduction of 0.007 ms per transaction represents a decrease of 2% for this workload.

7.6.6 Low-level CICS optimizations conclusions

Each scenario demonstrated a reduction in CPU usage per transaction for this workload. Where a workload uses any combination of performance class monitoring, trace, or many MRO sessions, the benefits that are realized in CICS V5.3 can be significant.

Even workloads that do not use these facilities can achieve a reduction in CPU use, as described in 7.6.5, “Monitoring and trace disabled with low numbers of MRO sessions” on page 121.

7.7 Web support and web service optimization

In CICS TS V5.3, the pipeline processing of HTTP requests is streamlined so that an intermediate web attach task (CWYN transaction) is no longer required in most situations. Removing the intermediate web attach task reduces CPU and memory overheads for most types of SOAP and JSON-based HTTP CICS web services.

The socket listener task (CS0L transaction) is optimized to attach user transactions directly for fast-arriving HTTP requests. The web attach task is bypassed, which reduces the CPU time that is required to process each request.

There also is a benefit for inbound HTTPS requests, where SSL support is provided by the Application Transparent Transport Layer Security (AT-TLS) feature of IBM z/OS Communications Server. In CICS, TCPIP SERVICE resources define the association between ports and CICS services, including CICS web support. These resources can be configured as AT-TLS aware and obtain security information from AT-TLS.

Performance is also improved for HTTPS requests where SSL support is provided by CICS. Although these requests still require the CWXN transaction, the number of TCB change mode operations was reduced.

For more information about the CPU savings that were achieved for an HTTP web services workload in several configuration scenarios, see *IBM CICS Performance Series: Web Services Performance in CICS TS V5.3*, REDP-5322, which is available at this website:

<http://www.redbooks.ibm.com/abstracts/redp5322.html>

7.8 Java workloads

Optimizations to the thread and TCB management mechanisms in CICS TS V5.3 provide a benefit to Java applications that are hosted in OSGi JVM servers and WebSphere Liberty JVM servers.

This section presents a comparison between CICS TS V5.2 and V5.3 when Java workloads are run.

7.8.1 Java workload configuration

The hardware and software that was used for the benchmarks is described in 7.1, “Introduction” on page 110. The measurement LPAR was configured with three GCPs and one zIIP, which resulted in an LSPR equivalent processor of 2964-704. The driving LPAR was configured with three GCPs, which resulted in an LSPR equivalent processor of 2964-703.

To minimize variance in the performance results that might be introduced by the Just-In-Time compiler (JIT), the workload was run at a constant transaction rate for 20 minutes to provide a warm-up period. The request rate was increased every 5 minutes, with the mean CPU usage per request calculated by using the final minute of data from the 5-minute interval. CPU usage data was collected by using IBM z/OS Resource Measurement Facility (RMF).

All configurations used a single CICS region with one installed JVMSERVER resource with a configured maximum of 25 threads. CICS TS V5.2 and CICS TS V5.3 used Java 7.1 SR3 (64-bit) and IBM WebSphere Application Server Liberty V8.5.5.7.

Note: IBM WebSphere Application Server Liberty V8.5.5.7 support for CICS V5.1 and V5.2 is provided by CICS APAR PI50345.

For database access, all workload configurations accessed DB2 V10 by using the JDBC type 2 driver.

7.8.2 Java servlet workload

The Java servlet application is hosted in a CICS JVM server that uses the embedded WebSphere Liberty server. The workload is driven through HTTP requests by using IBM Workload Simulator for z/OS, as described in section 2.4, “Driving the workload” on page 16. The servlet application accesses VSAM data by using the JCICS API and accesses DB2 by using the JDBC API. For more information about the workload, see 3.4, “WebSphere Liberty servlet with JDBC and JCICS access” on page 26.

Both configurations used the following JVM options:

- ▶ -Xgcpolicy:gencon
- ▶ -Xcompressedheap
- ▶ -XXnosuballoc32bitmem
- ▶ -Xmx200M
- ▶ -Xms200M
- ▶ -Xmnx60M
- ▶ -Xmns60M
- ▶ -Xmox140M
- ▶ -Xmos140M

The results of the benchmark are shown in Figure 7-7.

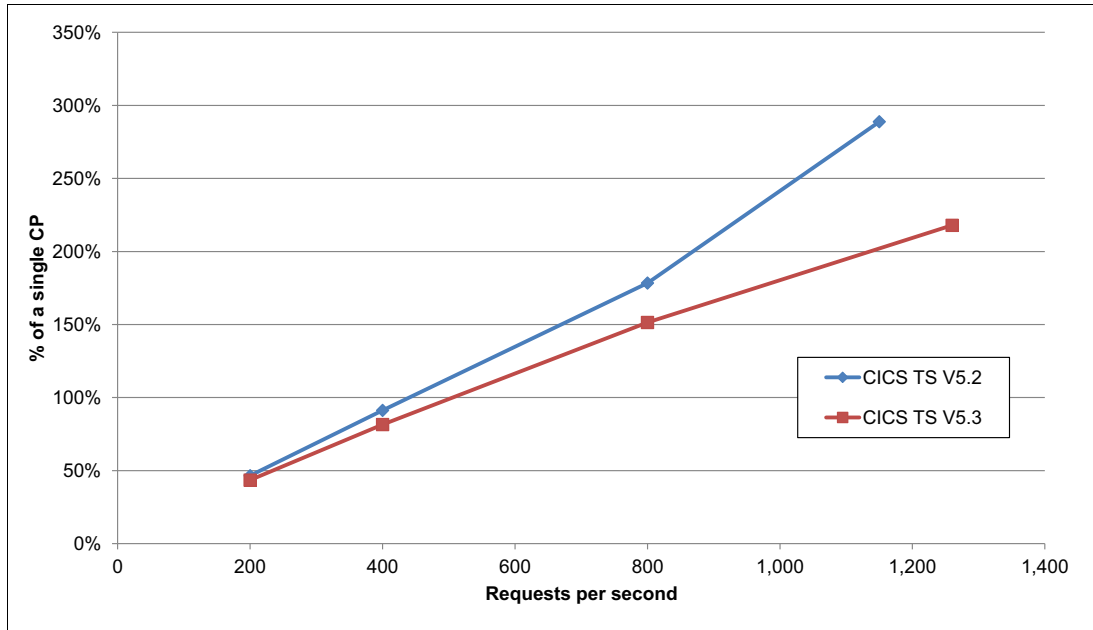


Figure 7-7 Comparing overall CPU utilization for Java servlet workload with CICS TS V5.2 and V5.3

As shown in Figure 7-7, the new thread management mechanism in CICS WebSphere Liberty provides reduced CPU costs and improved scalability characteristics, with V5.3 maintaining cost per request to higher request rates than V5.2.

The chart in Figure 7-8 presents the same data as Figure 7-7 on page 124, but broken into usage that is non-eligible for offload and usage that is eligible for offload to a zIIP engine.

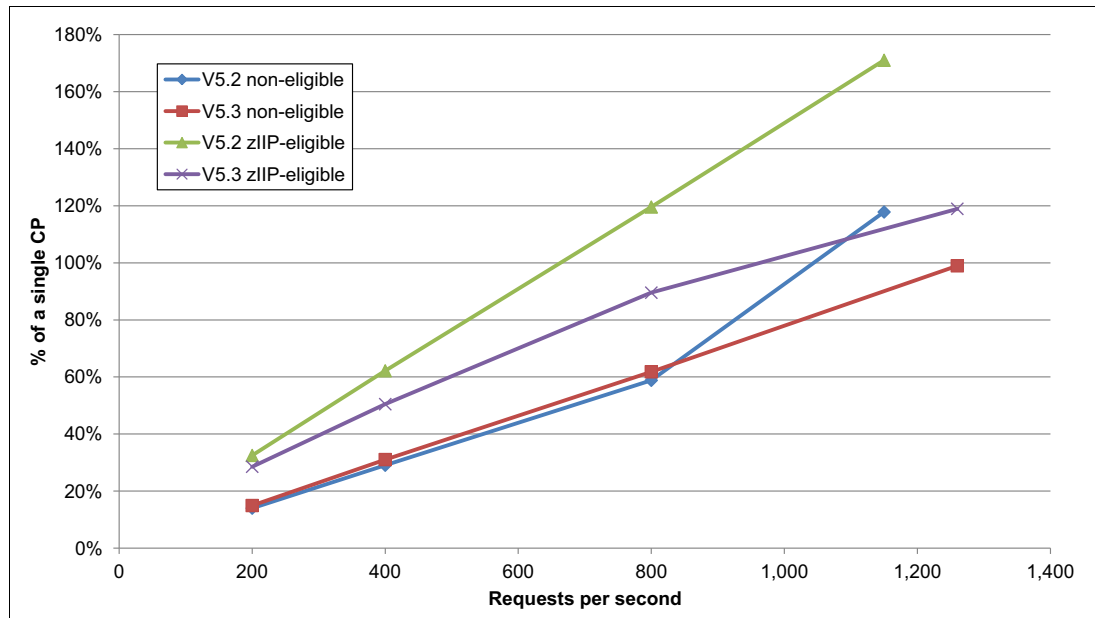


Figure 7-8 Comparing offload-eligible CPU utilization for Java workload with CICS TS V5.2 and V5.3

The chart in Figure 7-8 shows better scalability for the non-eligible component of the CPU usage. The chart also shows that the overall reduction in CPU usage that is shown in Figure 7-7 on page 124 is achieved by reducing the amount of zIIP-eligible CPU.

7.8.3 Java OSGi workload

The Java OSGi workload is composed of several applications and is described in 3.5, “Java OSGi workload” on page 27. The CICS TS V5.2 and CICS TS V5.3 configurations both used the following JVM options:

- ▶ -Xgcpolicy:gencon
- ▶ -Xcompressedheap
- ▶ -XXnosuballoc32bitmem
- ▶ -Xmx100M
- ▶ -Xms100M
- ▶ -Xmnx70M
- ▶ -Xmns70M
- ▶ -Xmox30M
- ▶ -Xmos30M

The benchmark results are shown in Figure 7-9.

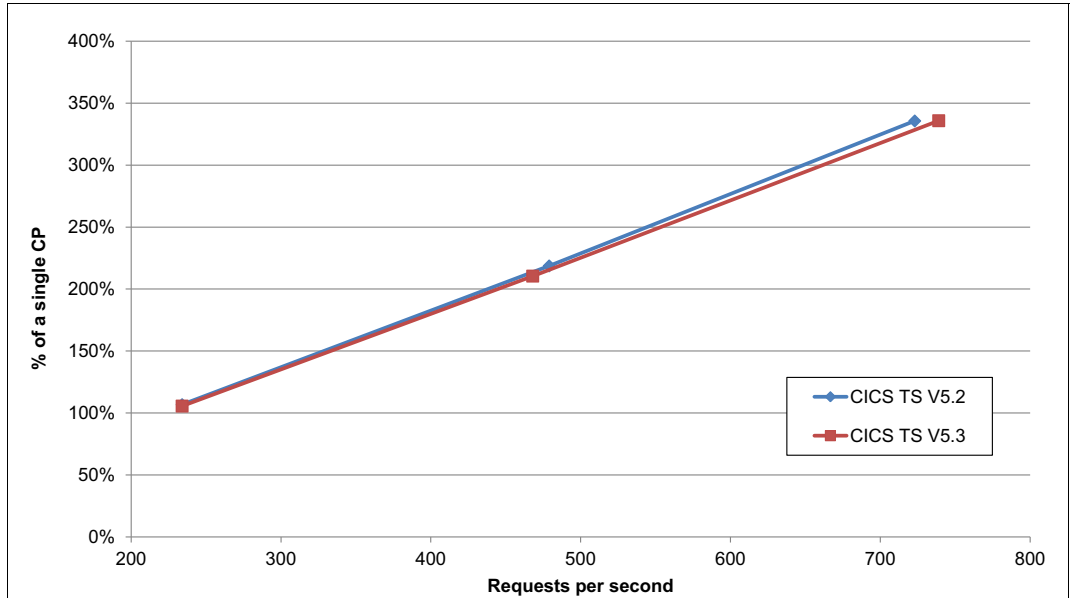


Figure 7-9 Comparing overall CPU utilization for Java OSGi workload with CICS TS V5.2 and V5.3

The chart in Figure 7-9 shows a slight reduction in overall CPU usage per transaction because of the improved TCB management.

The chart in Figure 7-10 shows the same data as Figure 7-9, but broken into usage that is non-eligible for offload and usage that is eligible for offload to a zIIP engine.

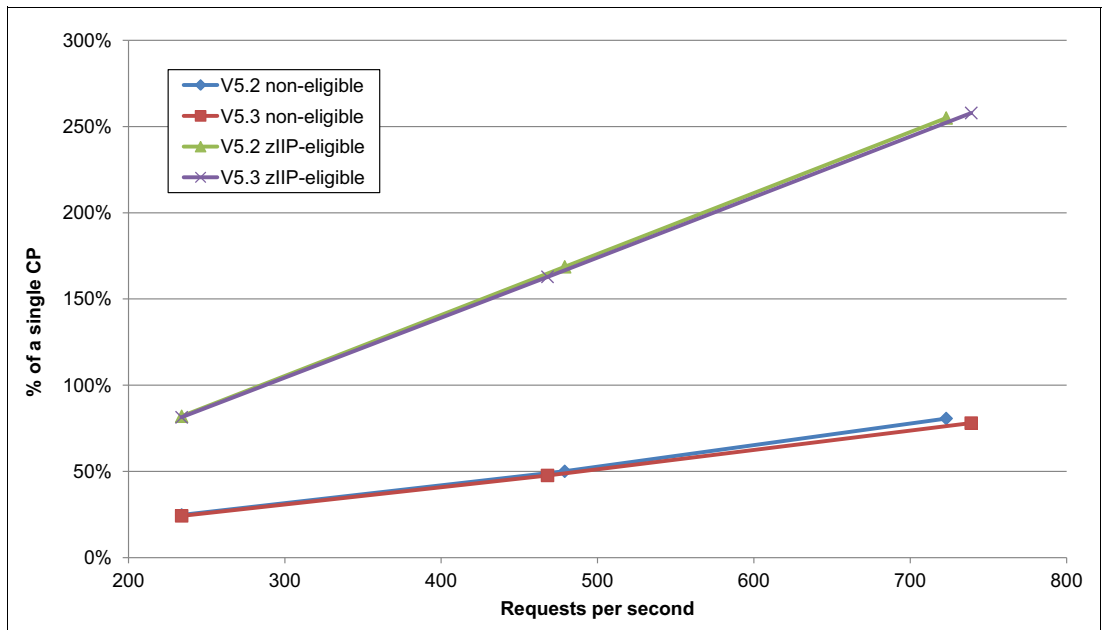


Figure 7-10 Comparing offload-eligible CPU utilization for OSGi workload with CICS TS V5.2 and V5.3

Both configurations scale well, with the ratio of eligible to non-eligible work remaining consistent between the V5.2 and V5.3 releases.

7.9 Java 8 performance

Every new release of Java provides more scope for performance improvements, the magnitude of which depends on the application. This section describes the effects of varying the Java release within a CICS environment for various workloads.

A JVM server in CICS TS for z/OS V5.3 can use Java 7.0, Java 7.1, or Java 8 as the runtime environment. A single CICS region can host multiple JVM server instances, with a different Java runtime version used in each instance.

7.9.1 Improvements in Java 7.0, Java 7.1, and Java 8

Java 7.0 uses hardware instructions that were introduced in the IBM zEnterprise 196 (z196) and the IBM zEnterprise EC12 (zEC12) machines. When running on a zEC12, the JVM also uses the new transactional memory capabilities of the hardware.

Java 7.1 extends the zEC12 exploitation by using technologies, such as IBM z Systems Data Compression (zEDC) for zip acceleration. Java 7.1 SR3 introduces improved zIIP-offload characteristics, which can reduce cost for Java applications in CICS.

Java 8 introduces the use of hardware instructions that were introduced in the IBM z13 machine. Also used are technologies, such as single instruction multiple data (SIMD) instructions and improved cryptographic performance that uses Crypto Express5S and CP Assist for Cryptographic Function (CPACF).

The IBM Java Crypto Engine (JCE) in Java 8 SR1 automatically detects and uses an on-core hardware cryptographic accelerator that is available through the CPACF. It also uses the SIMD vector engine that is available in the IBM z13 to provide industry-leading security performance. CPACF instructions are used to accelerate the following cryptographic functions:

- ▶ Symmetric key algorithms (AES, 3DES and DES with CBC, CFB and OBF modes)
- ▶ Hashing (SHA1 and SHA2)

Optimized routines accelerate the popular P256 NIST Elliptic Curve (ECC) Public Key Agreement. SIMD instructions are used in these routines to further enhance performance.

Java 8 SR2 also introduces the same improved zIIP-offload characteristics as seen in Java 7.1 SR3.

7.9.2 Java performance benchmarks in CICS

The following workloads were used to examine the behavior of Java applications in a CICS environment:

- ▶ A OSGi JVM server with a mixture of applications that use JDBC and JCICS calls to access DB2, VSAM data, and CICS temporary storage
- ▶ A WebSphere Liberty servlet application that uses JDBC and JCICS calls to access DB2 and VSAM data
- ▶ A WebSphere Liberty JSON-based web service that uses z/OS Connect

For performance testing, the following Java runtime environment levels were used:

- ▶ Java 7.0 SR9
- ▶ Java 7.1 SR3
- ▶ Java 8 SR2

7.9.3 Java 8 and OSGi applications

This workload uses the configuration as described in 7.8.3, “Java OSGi workload” on page 125. Several applications provide a mixture of operations, including JDBC access, VSAM access, string manipulation, and mathematical operations.

Figure 7-11 shows the average cost per transaction for each of the Java versions under test when the mixed OSGi application workload is run.

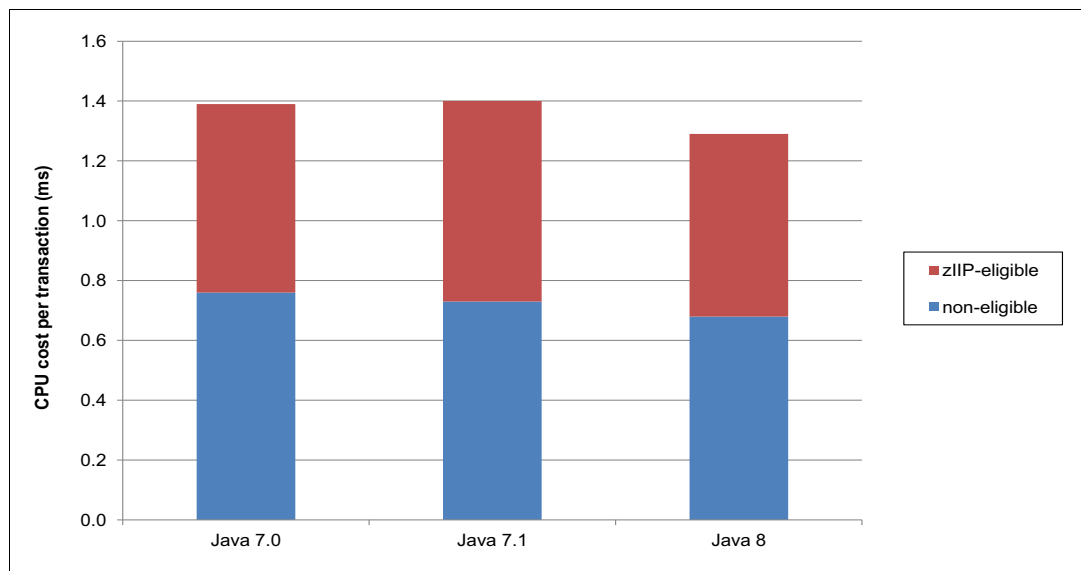


Figure 7-11 Comparing Java versions for OSGi JVM server workload

The chart shows a slight improvement in zIIP eligibility in Java 7.1 when compared to Java 7.0, but with no reduction in overall CPU per transaction.

Java 8 improves the Java 7.1 benchmark result by reducing the overall cost per transaction (from 1.40 ms to 1.29 ms) and reducing the amount of non-eligible CPU (from 0.73 ms to 0.68 ms). The improvements in the Java 8 environment are achieved by improvements to the JIT compiler and Java class library changes.

7.9.4 Java 8 and WebSphere Liberty servlet applications

This workload uses the configuration as described in 3.4, “WebSphere Liberty servlet with JDBC and JCICS access” on page 26. In all, 200 simulated web clients accessed the Java application at a rate of approximately 2,500 requests per second.

Figure 7-12 on page 129 shows the cost per request for each of the Java 7.0, Java 7.1, and Java 8 run times when the CICS WebSphere Liberty servlet application is run.

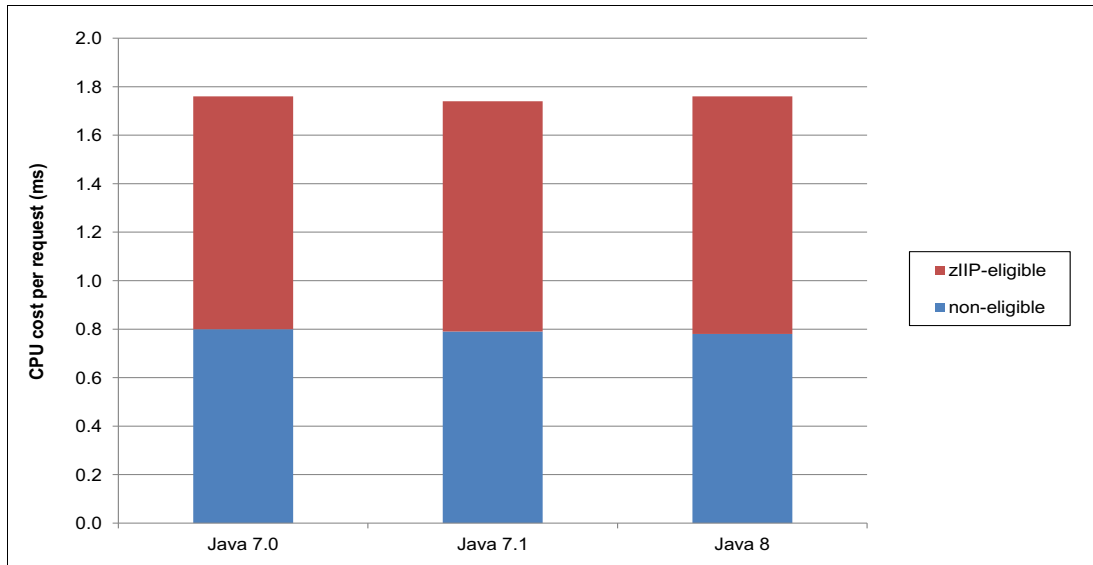


Figure 7-12 Comparing Java versions for JDBC and JCIICS servlet workload

No significant differences in total CPU per request are observed for this workload when comparing Java 7.0, Java 7.1, and Java 8. zIIP eligibility is slightly improved when Java 8 is used.

7.9.5 Java 8 and z/OS Connect applications

The z/OS Connect application that is described in 7.12, “z/OS Connect for CICS” on page 134 was used to compare the effects of the supported Java versions. A small JSON request and response was used, which contained 32 bytes of user data for each HTTP flow. The data was transmitted by using SSL with persistent connections.

The results of the benchmark comparing the three Java versions are shown in Figure 7-13.

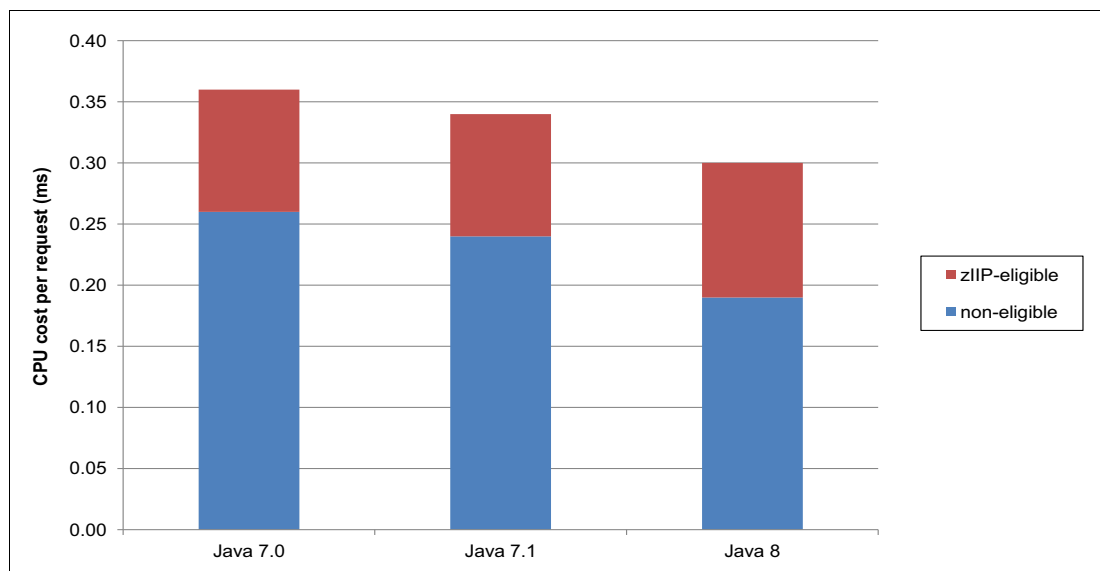


Figure 7-13 Comparison of Java versions for a z/OS Connect workload

Java 7.1 provides a reduction in overall CPU per request by reducing the amount of non-eligible CPU that is used.

Java 8 further improves on the Java 7.1 result through a reduction in non-eligible and overall CPU cost for each request. The use of persistent SSL connections means that most of the performance improvements are achieved because of the increased AES performance.

As the transmitted document size increases, the SSL payload size increases. Increasing the size of the SSL payload allows an application to achieve greater performance benefits when compared to Java 7.0 or Java 7.1.

7.10 Simultaneous multithreading with Java workloads

The zIIP processors in a z13 system can run to two threads simultaneously in a single core while sharing certain processor resources, such as execution units and caches. This capability is known as simultaneous multithreading (SMT). The use of SMT for two threads concurrently is known as *SMT mode 2*.

This section describes SMT, the methods that are used to measure the effectiveness of the technology, and the results of a Java benchmark in CICS to demonstrate the increased capacity that is available when SMT is enabled.

7.10.1 Introduction to SMT

SMT technology allows instructions from more than one thread to run in any pipeline stage at a time. Each thread has its own unique state information, such as program status word (PSW) and registers. The simultaneous threads cannot necessarily run instructions instantly and at times must compete to use certain core resources that are shared between the threads. In some cases, threads can use shared resources that are not experiencing competition.

Generally, SMT mode 2 can run more threads over the same period on a single core. This increased core usage leads to greater core capacity and a higher throughput of work. Figure 7-14 shows how SMT increases the capacity of a single core by enabling the simultaneous running of two threads.

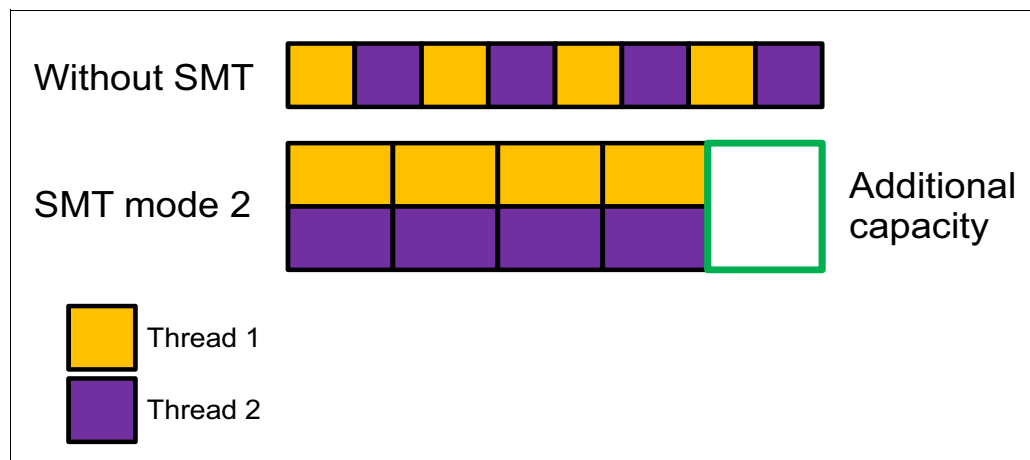


Figure 7-14 Demonstrating increased capacity by enabling SMT mode 2

Although each of the threads that are shown in Figure 7-14 on page 130 can take longer to run, the capability of SMT to run both simultaneously means that more threads can complete during a specific period, which increases the overall thread execution rate of a single core. Running more threads in a specific time increases the system throughput.

For more information about SMT, see *IBM z13 Technical Guide*, SG24-8251, which is available at this website:

<http://www.redbooks.ibm.com/abstracts/sg248251.html>

7.10.2 Measuring SMT performance

IBM z/OS RMF fully supports the extra performance information that is available when operating in SMT mode 2.

The IIP service times that are found in an RMF *Workload Activity* report are normalized by an SMT *capacity factor* (CF) when zIIP processors are in SMT mode 2. The CF is the ratio of work performed with SMT mode 2 enabled, when compared to SMT disabled. The normalization process reflects the increased ability of a zIIP in SMT mode 2 to perform more work.

RMF provides key metrics in the *Multi-threading Analysis* section of a *CPU Activity* report when zIIP processors are in SMT mode 2. The following terms are used when describing the workload performance:

- ▶ MAX CF reports the maximum CF: The ratio of the maximum amount of work the zIIPs performed with multithreading enabled compared to disabled.
The MAX CF value can be 0.0 - 2.0, with typical values 1.1 - 1.4.
- ▶ CF reports the average capacity factor: The ratio of the average amount of work the zIIPs performed with multithreading enabled compared to disabled.
The CF value can be 0.0 - 2.0, with typical values 1.0 - 1.4.
- ▶ AVG TD reports the average thread density: The average number of running threads while the core is busy.
The AVG TD value can be 1.0 - 2.0.

Figure 7-15 shows an extract of an RMF CPU Activity report. The average CF for the zIIP processors is highlighted for use in a later calculation.

----- MULTI-THREADING ANALYSIS -----					
CPU TYPE	MODE	MAX CF	CF	AVG TD	
CP	1	1.000	1.000	1.000	
IIP	2	1.303	1.303	1.996	

Figure 7-15 Extract of RMF CPU Activity report

In the IBM z13 hardware, SMT mode 2 is available for zIIP processors only; therefore, the MODE and CF values for general CPs is always 1.

Figure 7-16 shows an extract of an RMF Workload Activity report. The IIP service time and IIP APPL% figures are highlighted for use in a later calculation.

-TRANSACTIONS-	TRANS-TIME	HHH.MM.SS.TTT	--DASD I/O--	---SERVICE---	SERVICE TIME	---APPL% ---
AVG	1.00	ACTUAL	31	SSCHRT 0.0	IOC 24160	CPU 268.334 CP 213.51
MPL	1.00	EXECUTION	0	RESP 0.0	CPU 21360K	SRB 0.008 AAPCP 0.00
ENDED	26121	QUEUED	0	CONN 0.0	MSO 0	RCT 0.000 IIPCP 180.65
END/S	435.36	R/S AFFIN	0	DISC 0.0	SRB 607	IIT 0.000
#SWAPS	0	INELIGIBLE	0	Q+PEND 0.0	TOT 21385K	HST 0.000 AAP N/A
EXCTD	0	CONVERSION	0	IOSQ 0.0	/SEC 356418	IIP 179.42
AVG ENC	0.00	STD DEV	6			IIP 140.240
REM ENC	0.00				ABSRPTN 356K	
MS ENC	0.00				TRX SERV 356K	

Figure 7-16 Extract of RMF Workload Activity report

The APPL% IIP value is the amount of actual zIIP resource used. The APPL% IIP value is not normalized and shows how busy the processors are. The LPAR that was used for this benchmark was configured with three dedicated CPs and two dedicated zIIPs. Therefore, the maximum value for APPL% CP is 300%, and the maximum value for APPL% IIP is 200%.

The SERVICE TIME IIP value is the normalized zIIP time, factored by the CF. Note the relationship between SERVICE TIME IIP and APPL% IIP in the following equation:

$$\text{APPL \% IIP} = \frac{\text{SERVICE TIME IIP}}{\text{Interval} \times \text{CF}} \times 100$$

The reports that are shown in Figure 7-15 on page 131 and Figure 7-16 are extracted from an RMF report with an interval of 60 seconds; therefore, the highlighted values that are shown in Figure 7-15 on page 131 and Figure 7-16 can be used in the previous equation, as shown in the following equation:

$$\text{APPL \% IIP} = \frac{140.240}{60 \times 1.303} \times 100 = 179.38\%$$

The slight discrepancy between the calculated and reported APPL% IIP values is because other values in the report are rounded.

7.10.3 CICS throughput improvement

A z/OS Connect workload was used to demonstrate the change in CPU utilization when running Java in CICS with SMT mode 2 disabled and enabled. Figure 7-17 on page 133 shows a comparison of a Java-based workload when running with the two SMT configurations.

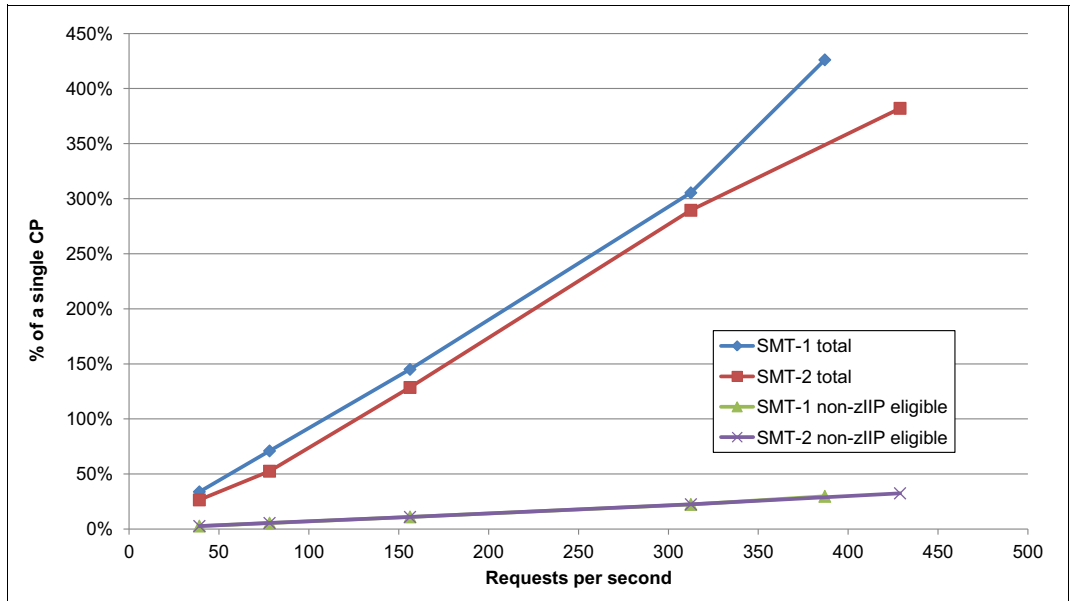


Figure 7-17 Comparing a z/OS Connect workload with SMT mode disabled and SMT mode 2

The chart that is shown in Figure 7-17 plots the sum of the APPL% CP and APPL% IIP values from the RMF Workload Activity report.

Comparing the SMT-1 total and SMT-2 total lines, it can be seen that the total CPU cost is lower with SMT mode 2 enabled and the maximum throughput is increased.

The plot lines that show the amount of work that was not eligible to be offloaded to a System z Integrated Information Processor (zIIP) remains constant between the comparisons. The performance benefits are achieved through increased zIIP capacity.

7.11 Reporting of CPU time to z/OS Workload Manager

Mobile Workload Pricing is an IBM Software Pricing Option that was announced in May 2014. It offers a discount on MSUs consumed by transactions that originated on a mobile device. To use this discount, customers need a process that is agreed upon by IBM to identify (tag and track) their mobile-sourced transactions and their use.

Before CICS TS V5.3, the identification and accumulation of CPU time for certain transaction types required CICS Performance class monitoring to be active. The collection of high-volume SMF data in a production environment can introduce significant overhead.

z/OS Workload Manager (WLM) APAR OA47042 introduces enhancements to simplify the identification and reporting of mobile-sourced transactions and their processor consumption. For more information about updates to WLM, see the following APAR website:

<http://www.ibm.com/support/docview.wss?uid=isg10A47042>

The associated APAR OA48466 is available for IBM z/OS RMF, which provides support for the new WLM function that is provided by APAR OA47042. For more information about the updates to RMF, see the following APAR website:

<http://www.ibm.com/support/docview.wss?uid=isg10A48466>

The CICS TS V5.3 release introduces support for the new functions that were introduced by WLM APAR OA47042. CPU time is reported to WLM on a per-transaction basis, which enables a granular approach to transaction CPU tracking without the requirement for CMF data.

No configuration changes are required in CICS to use the WLM updates. CPU information is reported to WLM if CICS detects Mobile Workload Pricing support was installed, with no other CPU overhead in the CICS region.

7.12 z/OS Connect for CICS

IBM z/OS Connect is software that enables systems that run on z/OS to better participate in today's mobile computing environment. z/OS Connect for CICS enables CICS programs to be called with a JSON interface.

z/OS Connect is distributed with CICS to enable connectivity, such as between mobile devices and CICS programs. The CICS embedded version of z/OS Connect is a set of capabilities that are used to enable CICS programs as JSON web services. z/OS Connect is an alternative to the JSON capabilities of the Java-based pipeline. The two technologies are broadly equivalent. Most JSON web services can be redeployed from one environment to the other without application or WSBIND file changes. However, the URI and security configuration can be different in each environment.

7.12.1 CICS TS V5.3 performance enhancement

A significant performance enhancement in the CICS TS V5.3 release is the introduction of a JSON parser that is implemented in native (non-Java) code.

The parser implementation that is used by CICS is controlled by the `java_parser` attribute of the `provider_pipeline_json` XML element in the pipeline configuration file. For more information about the `provider_pipeline_json` element, see the “The `<provider_pipeline_json>` element” topic in IBM Knowledge Center at this website:

<https://ibm.biz/Bd4zBa>

A sample XML configuration file for the z/OS Connect pipeline handler is supplied in the following location relative to the CICS installation root directory:

```
./samples/pipelines/jsonzosconnectprovider.xml
```

Example 7-2 shows a pipeline file that uses the CICS JVMSERVER resource that is named DFHWLP and specifies the use of the native parser.

Example 7-2 Sample pipeline configuration file that specifies the native parser implementation

```
<provider_pipeline_json java_parser="no">  
  <jvmserver>DFHWLP</jvmserver>  
</provider_pipeline_json>
```

This section provides a performance comparison when various JSON request and response sizes for the Java and native parser implementations are used. In all configurations, SSL was used.

The methodology and applications that were used to produce the performance test results for z/OS Connect in CICS were similar to the methodology and applications that were used when testing the JSON support in CICS TS V5.2. For more information, see 6.7, “JSON support” on page 95. To expand the workload, an extra request and response size of 64 KB was added.

7.12.2 Varying payload sizes by using Java parser

By using z/OS Connect for CICS with the default Java parser, CPU usage was measured for a range of payload sizes. The CPU cost per request is shown in Figure 7-18 for a range of request and response size combinations. Total CPU cost per request is broken into non-zIIP-eligible and zIIP-eligible components.

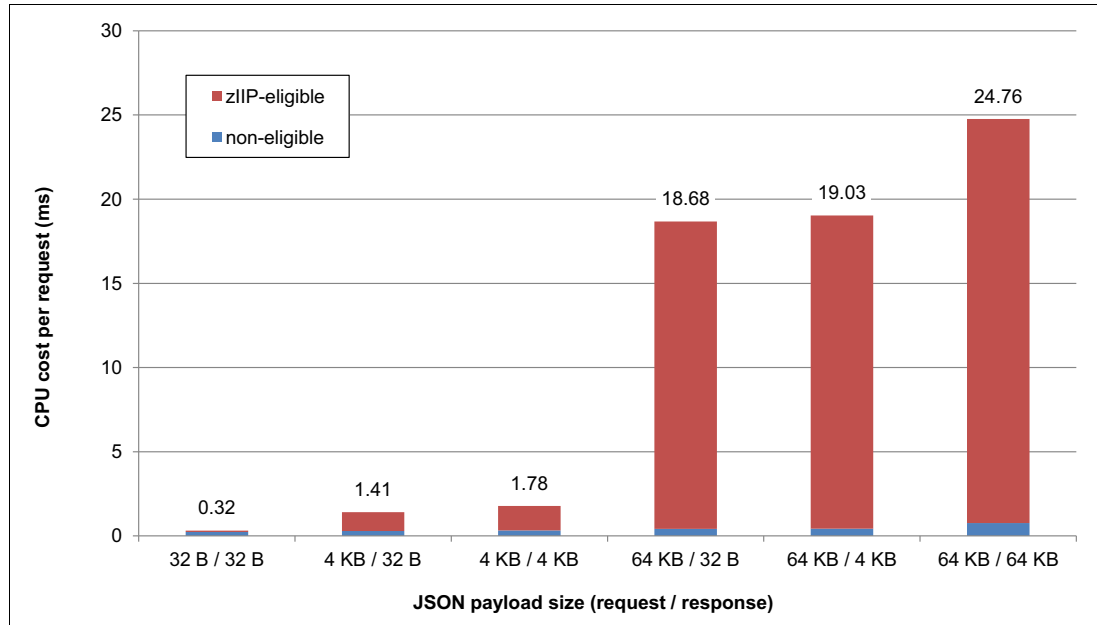


Figure 7-18 CPU comparison for various request and response payloads by using the Java parser

It is clear that the CPU cost per request depends on the size of the JSON documents that were received or transmitted. A significant fraction of the CPU cost incurred for larger JSON documents is zIIP-eligible.

7.12.3 Comparing Java and native parsers

By using a medium-sized JSON request and response, the CPU usage was compared for the Java and native parsers. The scenario used a 4 KB request and 4 KB response. The result of this comparison is shown in Figure 7-19. As per Figure 7-18 on page 135, the CPU usage is broken into non-zIIP-eligible and zIIP-eligible components.

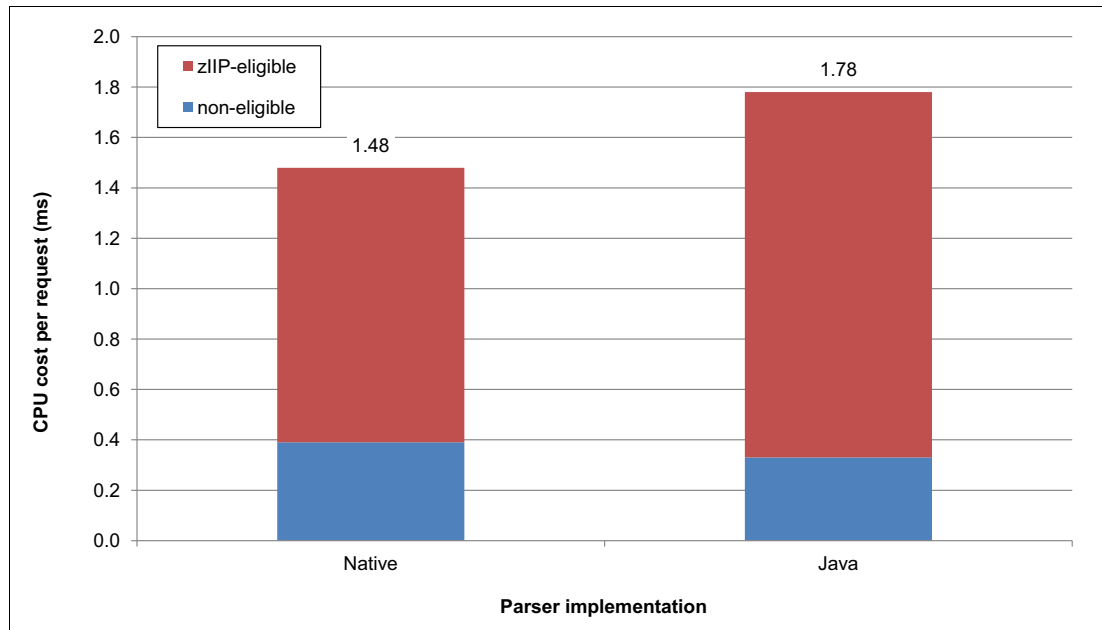


Figure 7-19 Comparing Java and native parsers for a medium-sized request and response

The chart in Figure 7-19 shows that for a medium-size request and response, the overall CPU cost per request is reduced with the native parser. Use of the native parser slightly increases the amount of non-zIIP-eligible CPU time from 0.33 ms to 0.39 ms per request.

7.12.4 Comparing Java and native parsers with varying request sizes

Extending the test scenario that is described in 7.12.3, “Comparing Java and native parsers” on page 136, various request sizes were used. Each request returns a response of 32 bytes. The following request sizes were tested:

- ▶ 32 bytes (as shown in Figure 7-20 on page 137)
- ▶ 4 KB (as shown in Figure 7-21 on page 137)
- ▶ 64 KB (as shown in Figure 7-22 on page 138)

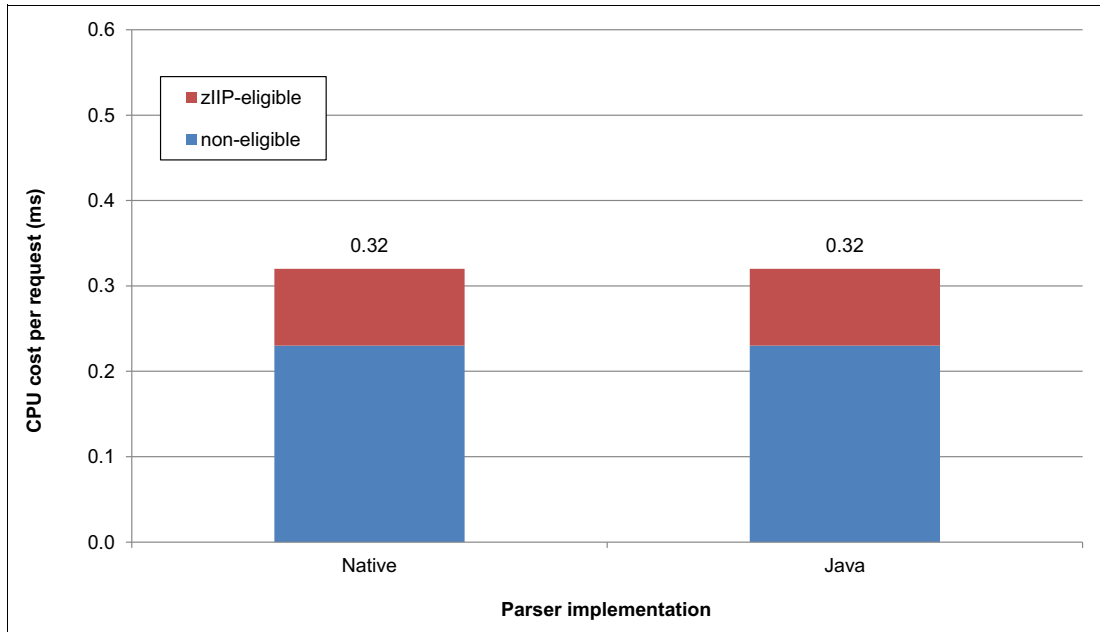


Figure 7-20 Comparing Java and native parsers for 32-byte request with 32-byte response

For the 32-bytes request with 32-bytes response scenario, there is no significant difference in CPU usage or zIIP-eligibility between the Java and native parsers.

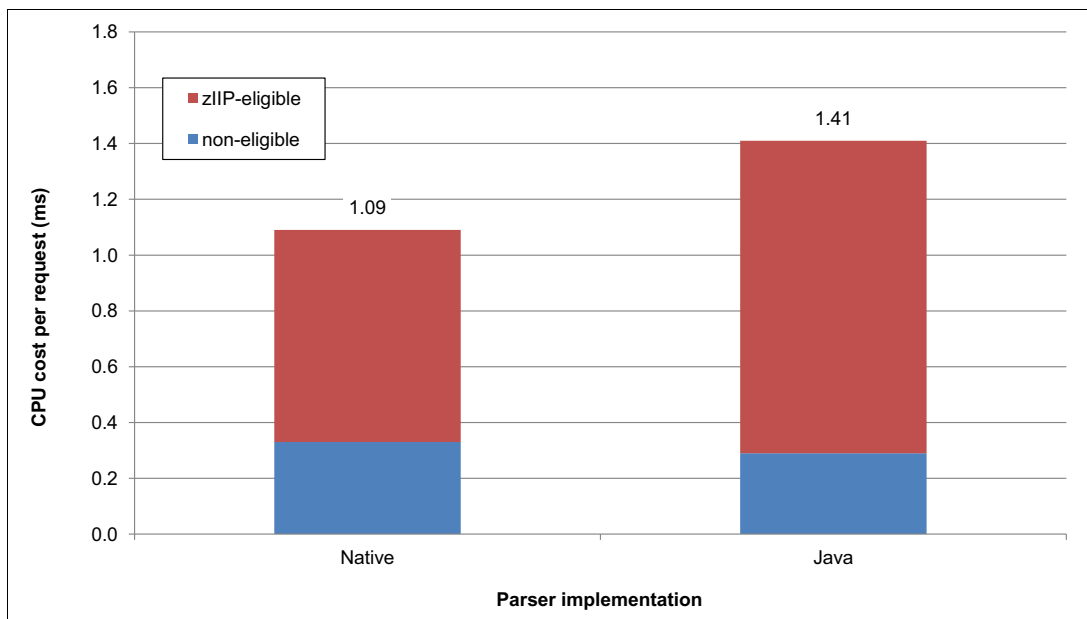


Figure 7-21 Comparing Java and native parsers for 4 KB request with 32-byte response

For the 4 KB request with 32-bytes response scenario, the use of the native parser results in a reduction in total CPU per request, from 1.41 ms to 1.09 ms. However, the native parser uses more slightly non-zIIP-eligible CPU time, which increases from 0.29 ms to 0.33 ms.

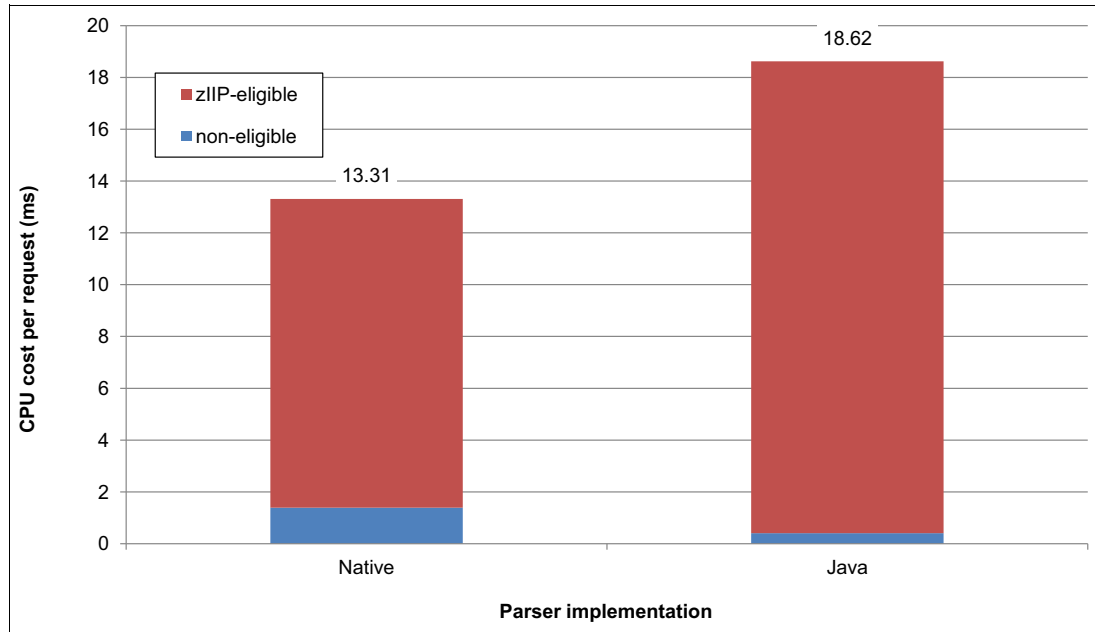


Figure 7-22 Comparing Java and native parsers for 64 KB request with 32-byte response

The 64 KB request with 32-byte response scenario shows a significant reduction in total CPU usage per request. Overall CPU usage per request reduces from 18.62 ms to 13.31 ms, but non-zIIP-eligible usage increases from 0.41 ms to 1.39 ms per request.

The charts that are shown in Figure 7-20 on page 137, Figure 7-21 on page 137, and Figure 7-22 show that the native parser reduces overall CPU usage for each JSON request. As expected, the largest performance gains are realized with large request sizes. The use of the native parser also has the expected effect of the use of more non-zIIP-eligible CPU than the Java parser.

7.12.5 Comparing Java and native parsers with varying response sizes

The benchmark was further modified such that various response sizes were used. Each request was 64 KB and the performance of the Java and native parsers were compared. The following response sizes were tested:

- ▶ 32 bytes (as shown in Figure 7-22 on page 138)
- ▶ 4 KB (as shown in Figure 7-23 on page 139)
- ▶ 64 KB (as shown in Figure 7-24 on page 139)

Unlike as described in 7.12.4, “Comparing Java and native parsers with varying request sizes” on page 136, this section describes scenarios in which the response sizes were modified. With an invariant request size, the benefit of the native parser is expected to remain constant across all scenarios because the parser operates on the incoming request only.

Performance results for a 64 KB request with 32-byte response are described in 7.12.4, “Comparing Java and native parsers with varying request sizes” on page 136. Figure 7-23 on page 139 shows the native parser reducing the overall CPU by 5.31 ms, but increasing the non-zIIP-eligible CPU by 0.98 ms per request.

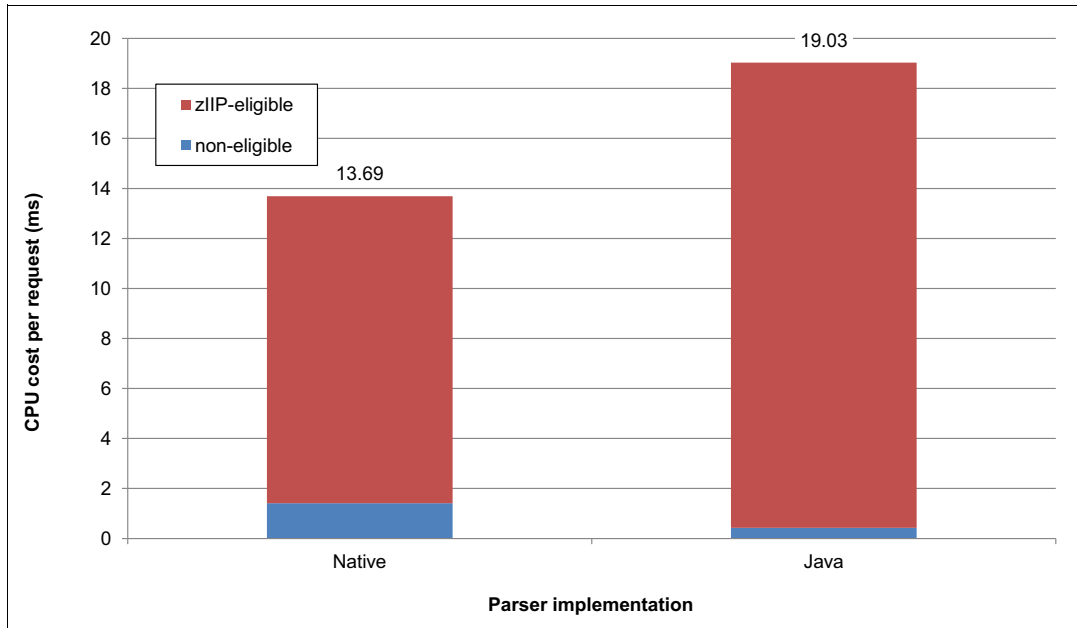


Figure 7-23 Comparing Java and native parsers for 64 KB request with a 4 KB response

With a 64 KB request and a 4 KB response, the native parser reduces overall CPU usage by 5.34 ms (as shown in Figure 7-24). The native parser uses 0.98 ms more non-zIIP-eligible CPU.

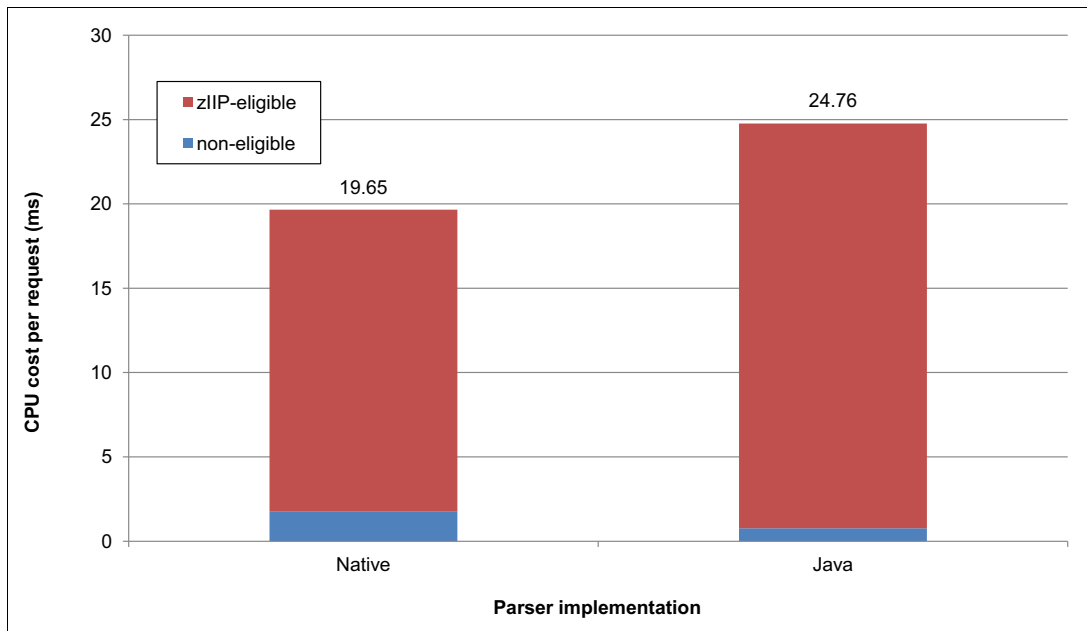


Figure 7-24 Comparing Java and native parsers for 64 KB request with a 64 KB response

For the 64 KB request and 64 KB response scenario, the native parser again reduces overall CPU usage by 5.11 ms. The native parser again increases non-zIIP-eligible CPU usage by 0.98 ms per request.

7.12.6 Native parser conclusion

The native parser can provide a significant reduction in overall CPU usage per request. The potential reduction in CPU usage is determined by the size of the inbound request. The native parser is not implemented by using Java; therefore, CPU usage by the native parser cannot be offloaded to a specialty engine.

The performance improvements in CICS TS V5.3 apply to the JSON parser component only; therefore, it has no effect on the CPU costs that are involved in producing a JSON response.

7.13 HTTP flow control

CICS TS V5.3 introduces the ability to enable performance tuning for HTTP to protect CICS from unconstrained resource demand. TCP/IP flow control in CICS TS V5.3 is for HTTP connections only. If enabled, it addresses the following situations:

- ▶ A pacing mechanism to prevent HTTP requests from continuing to be accepted by a CICS region, when the region reached its throughput capacity.
- ▶ Gives an opportunity to rebalance persistent connections on a periodic basis.

If HTTP flow control is enabled and the region becomes overloaded, CICS temporarily stops listening for new HTTP connection requests. If overloading continues, CICS closes HTTP persistent connections and marks all new HTTP connections as non-persistent. These actions prevent oversupply of new HTTP work from being received and queued within CICS, which allows feedback to TCP/IP port sharing and Sysplex Distributor. This ability promotes a balanced sharing of workload with other regions that are sharing the IP endpoint and allowing the CICS region to recover more quickly.

7.13.1 Server accept efficiency fraction

CICS HTTP flow control is implemented by queuing new HTTP connection requests in the TCP/IP socket backlog. Queuing requests in the TCP/IP socket backlog affects the server accept efficiency fraction (SEF).

Note: When ports that are managed by CICS are used, it is the CICS address space that is accepting connections; therefore, CICS is the *server application* (in IBM z/OS Communications Server terminology).

SEF is a measure (calculated at intervals of approximately 1 minute) of the efficiency of the server application in accepting new connection setup requests and managing its backlog queue. The SEF value is reported as a percentage. A value of 100% indicates that the server application is successfully accepting all its new connection setup requests. A value of 0% indicates that the server application is not responding to new connection set up requests. The SEF field is only available for a connection that is in listen state.

The **netstat** command can display the SEF value for an IP socket. This command is available in the TSO and z/OS UNIX System Services environments. The following sample commands produce the same output when inquiring about the state of port 4025:

- ▶ TSO environment
NETSTAT ALL (PORT 4025)
- ▶ z/OS UNIX System Services environment
netstat -A -P 4025

Example 7-3 shows a fragment of the output that is produced by the **netstat** command.

Example 7-3 Sample fragment of the output of the netstat command

ReceiveBufferSize:	0000065536	SendBufferSize:	0000065536
ConnectionsIn:	0000008574	ConnectionsDropped:	0000000000
MaximumBacklog:	0000001024	ConnectionFlood:	No
CurrentBacklog:	0000000000		
ServerBacklog:	0000000000	FRCABacklog:	0000000000
CurrentConnections:	0000001464	SEF:	100

When the SHAREPORTWLM option in a port definition is used, the SEF value is used to modify the IBM Workload Manager for z/OS server-specific weights, which influences how new connection setup requests are distributed to the servers sharing this port.

When the SHAREPORT option in a port definition is used, the SEF value is used to weight the distribution of new connection setup requests among the SHAREPORT servers.

Whether SHAREPORT or SHAREPORTWLM is specified, the SEF value is reported back to the sysplex distributor to be used as part of the target server responsiveness fraction calculation, which influences how new connection setup requests are distributed to the target servers.

For more information about the configuration of ports in IBM z/OS Communications Server, see the “PORT statement” topic in IBM Knowledge Center at this website:

<https://ibm.biz/Bd4zdY>

7.13.2 Flow control configuration

The behavior of HTTP flow control is specified by using the new system initialization parameter SOTUNING, which can be set to one of the following values:

- ▶ YES
Performance tuning for HTTP connections occurs to protect CICS from unconstrained resource demand. YES is the default value.
- ▶ 520
No performance tuning occurs.

Note: If sharing IP endpoints, ensure that all regions have the same SOTUNING value or uneven loading might occur.

For more information about the SOTUNING SIT parameter, see the “SOTUNING” topic in IBM Knowledge Center at this website:

<https://ibm.biz/Bd4zd2>

7.13.3 Flow control operation

When a CICS region reaches the maximum task limit (MXT), it stops accepting new HTTP connections and incoming requests are queued in the backlog for the TCP/IP socket. When the MXT condition is relieved, CICS starts accepting new connections again.

In the case of persistent connections (that is, connections that were accepted and maintained their connection), work can continue to be received even after reaching MXT. In this situation, when the number of active transactions in CICS and the number of queued requests in the IP socket reaches 110% of the MXT value, client connections are actively disconnected to route work away from an overloaded CICS region.

When actively disconnecting clients, the current request is permitted to complete and then the connection is closed. New connection requests are made non-persistent until the region drops below 95% of the MXT value.

In addition to these mechanisms, CICS also disconnects a client connection every 1,000 requests. This disconnection rate gives an opportunity for rebalancing the connection when the client reconnects.

7.13.4 CICS statistics enhancements

CICS TCP/IP global statistics was enhanced to provide information about how incoming work is being processed and the effect flow control is having on HTTP requests. Example 7-4 shows an extract of a sample DFHSTUP report for an HTTP workload where flow control is enabled.

Example 7-4 Extract of sample TCP/IP global statistics report produced by CICS TS V5.3 DFHSTUP

Performance tuning for HTTP connections	:	Yes	
Socket listener has paused listening for HTTP connections	:	Yes	
Number of times socket listener notified at task accept limit	:	25672	
Last time socket listener paused listening for HTTP connections	:	10/15/2015	11:13:26.3862
Region stopping HTTP connection persistence	:	Yes	
Number of times region stopped HTTP connection persistence	:	0	
Last time stopped HTTP connection persistence	:	--/--/----	---:---:---:----
Number of persistent HTTP connections made non-persistent	:	52554	
Number of times disconnected an HTTP connection at max uses	:	0	

For more information about available CICS statistics fields, see 7.5.6, “TCP/IP global statistics” on page 117.

7.13.5 Comparison of SOTUNING options

Table 7-3 lists CICS statistics reports, comparing a workload that is running in CICS with SOTUNING=YES to the same workload that is running in a CICS system with SOTUNING=520 for the same period. The workload in this case consisted of a simple HTTP web application where each inbound request made a new TCP/IP connection.

Table 7-3 Extract of CICS statistics reported values with SOTUNING=520 and SOTUNING=YES

CICS statistic	SOTUNING=520	SOTUNING=YES
Number of completed transactions	101,538	105,117
Peak queued transactions	2,193	3

CICS statistic	SOTUNING=520	SOTUNING=YES
Peak active transactions	150	150
Times stopped accepting new sockets	n/a	26,674
Number of times at MXT limit	1 (continuously)	29,418
CPU used	62.11 s	60.86 s
CPU per transaction	0.611 ms	0.578 ms
EDSA used (MB)	121 MB	60 MB

Although this example is an extreme case, it does demonstrate that it is more effective to queue work outside of CICS by preventing new connections being accepted in terms of CPU and EDSA usage.

In the SOTUNING=520 case, MXT was reached and the CICS region did not drop below that value of concurrent tasks, which remained permanently at MXT for the measurement interval. In the SOTUNING=YES case, the CICS system kept dropping in and out of MXT as it stopped new work arriving and then started accepting work as it dropped below MXT.

7.14 High transaction rate performance study

To demonstrate many of the performance improvements that were introduced in the CICS V5.3 release, a performance study was undertaken to drive a high rate of transactions through a CICS configuration. The study consisted of the following workloads:

- ▶ The first workload runs on a single z13 LPAR with 18 CPs up to a rate of 174,000 CICS transactions per second.
- ▶ The second workload runs on a single z13 LPAR with 26 CPs up to a rate of 227,000 CICS transactions per second.

For more information about the full results of this study, see *IBM CICS Performance Series: CICS TS V5.3 Benchmark on IBM z13*, REDP-5320, which is available at this website:

<http://www.redbooks.ibm.com/abstracts/redp5320.html>

7.15 WebSphere Liberty zIIP eligibility

As described in 7.8.2, “Java servlet workload” on page 123, a new thread management mechanism was introduced in CICS TS V5.3 for WebSphere Liberty workloads. As part of the CICS continuous delivery strategy this thread management support is further enhanced by APAR PI54263, increasing the zIIP eligibility of WebSphere Liberty workloads in CICS. This enhancement to CICS TS V5.3 can reduce the total general purpose (GP) CPU consumed for a given workload.

Prior to APAR PI54263, WebSphere Liberty assigned an HTTP request to a T8 TCB using a thread from its thread pool. Execution on the T8 TCB would then be suspended while the CICS transaction context was built for the task by executing on the QR TCB. On completion of the transaction initialization, the T8 TCB would resume, but it would resume in a state that was not eligible for execution on a zIIP, and even though regaining zIIP eligibility quickly the T8 would remain on a GP CPU until redispached by z/OS.

APAR PI54263 improves the zIIP eligibility of WebSphere Liberty workloads by allowing the transaction context to be built on a T8 TCB. This improves performance by removing the overhead of two TCB switches, and also never suspending or resuming the T8 whilst not zIIP eligible.

For more information about the changes in APAR PI54263, see the following article in the CICS Developer Center:

<https://developer.ibm.com/cics/2016/05/17/improving-ziiip-eligibility/>

To measure the performance improvements introduced by this change, a WebSphere Liberty workload was executed. You can find details about the performance results in the CICS Developer Center article referenced previously, and Table 7-4 presents a summary.

Table 7-4 Summary of performance improvements introduced by APAR PI54263

	GP per request (ms)	zIIP per request (ms)	CPU per request (ms)
CICS TS V5.2	0.039	0.421	0.460
CICS TS V5.3	0.018	0.322	0.340

Table 7-4 shows the total CPU consumed per request is reduced by 26% from 0.460 ms to 0.340 ms. The amount of GP CPU time is also reduced: from 0.039 ms to 0.018 ms per request, or a 53% reduction. Overall zIIP-eligibility of the workload is increased from 91.6% to 94.7%.

7.16 Link to WebSphere Liberty

A further continuous delivery enhancement to CICS TS V5.3 is provided by APAR PI63005, which enables any CICS program to link to a Java Platform, Enterprise Edition (Java EE) application, running in a WebSphere Liberty JVM server inside CICS.

To be invoked by a CICS program, the Java EE application is required to contain a plain old Java object (POJO), packaged as a web archive (WAR) or enterprise archive (EAR) file. A method in the Java EE application can be made a CICS program by use of the @CICSProgram annotation.

CICS creates the program resources defined by the @CICSProgram annotations when the application is started in a WebSphere Liberty JVM server. The WebSphere Liberty instance must be configured with the feature `cicsts:link-1.0`.

Data can be passed between non-Java and Java programs using the channels and containers interface; COMMAREA and INPUTMSG are not supported.

For more details about the Link to WebSphere Liberty functionality, see the topic “Invoking a Java EE application from a CICS program” in IBM Knowledge Center at this website:

<https://ibm.biz/BdjUp5>

7.16.1 Performance comparison

A benchmark was created to understand the relative CPU consumption of the following three scenarios, when linking from a COBOL application to a program that is written in one of the following languages:

- ▶ COBOL
- ▶ Java and hosted in a WebSphere Liberty JVM server
- ▶ Java and hosted in an OSGi JVM server

The benchmark consists of two programs, with the logic flow indicated:

1. PROGRAMA

- Create a CICS container (CONTAINER1) to pass data to.
- Use the **EXEC CICS LINK** command to PROGRAMB.
- Extract and validate contents of the CICS container, RESPONSE.
- Write a success message.
- End the transaction.

2. PROGRAMB

- Read the container, CONTAINER1.
- Build the container, RESPONSE.
- Return to the caller.

PROGRAMA was implemented using only COBOL. PROGRAMB was implemented using COBOL and Java, with two separate versions that are suitable for deployment in an OSGi JVM server and WebSphere Liberty JVM server environment. The COBOL version of PROGRAMB was defined as CONCURRENCY (REQUIRED) so that the program executed on an Open TCB.

The workload was executed on an IBM z13 with an LPAR configuration equivalent to a 2964-703, with three dedicated general-purpose CPs, and one dedicated zIIP in SMT=1 mode. The workload used z/OS V2.2, CICS TS V5.3 with APAR PI63005, and Java 8.0 SR3.

7.16.2 Link to WebSphere Liberty performance results

RMF was used to accurately measure the CPU cost per transaction. Response time information was obtained using CICS monitoring data. Table 7-5 lists the results of the workload when executed at a steady rate of approximately 5,000 transactions per second.

Table 7-5 CPU per transaction and response time comparison for COBOL and Java implementations

Implementation language	Total CPU (ms)	zIIP-eligible CPU (ms)	Response time (ms)
COBOL	0.0455	0.0000	1.3130
Java (WebSphere Liberty JVM server)	0.1014	0.0290	1.3810
Java (OSGi JVM server)	0.1491	0.0777	2.0310

The results from Table 7-5 are summarized in the chart in Figure 7-25 on page 146.

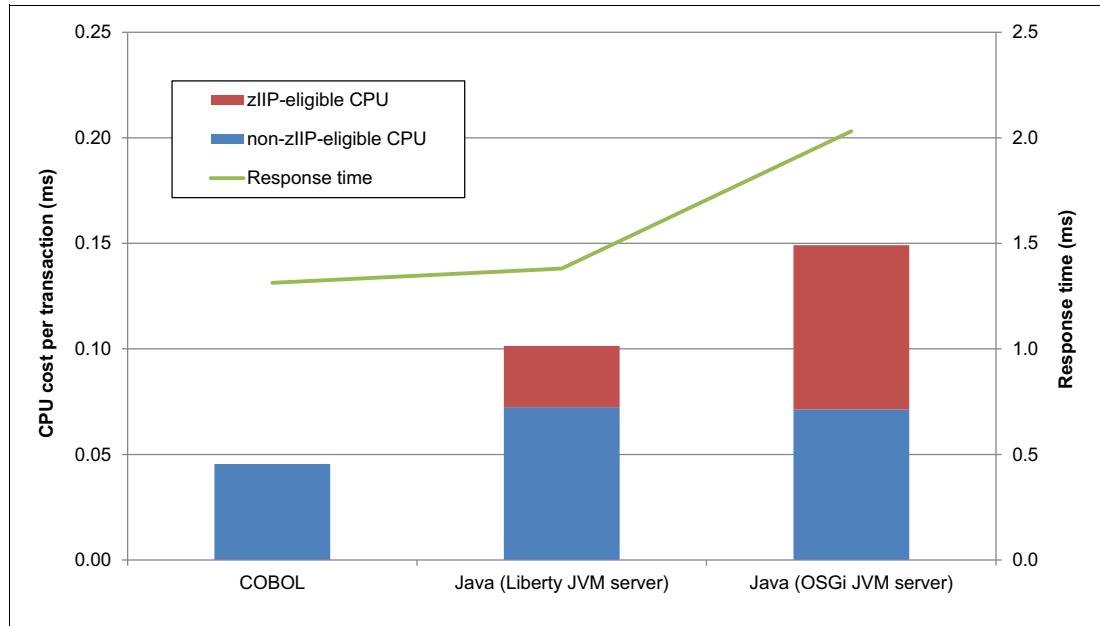


Figure 7-25 Plot of link to WebSphere Liberty benchmark results

7.16.3 Link to WebSphere Liberty performance conclusion

The programs used for this study are simple, with little application logic. They demonstrate the difference in CPU cost of the EXEC CICS LINK infrastructure within CICS that enables calls to COBOL or Java CICS programs.

Linking to COBOL is the lowest cost, both in terms of total and non-zIIP eligible CPU. Linking to the Java EE version of the application running in a WebSphere Liberty JVM server costs just over twice the total CPU cost of the COBOL version and about half of this extra time can be offloaded to run on a zIIP processor. Using either version of the Java program costs more total CPU than the COBOL version and also costs more general processor time too.

Much of the cost of running Java is zIIP-eligible but profiling analysis has shown that the management of the extra TCBs needed in the Java cases needs increased z/OS dispatcher time. The increased time spent in z/OS dispatching routines is not zIIP-eligible. Using JCICS also creates more calls to the CICS EXEC interface (with ASSIGN and CONTAINER management calls) which leads to a further increase in general processor time compared to the COBOL case.

Compared to the OSGi JVM server case, Java and UNIX threads are managed more efficiently in the WebSphere Liberty JVM server case by employing thread reuse techniques. This improved efficiency makes the WebSphere Liberty JVM server case significantly cheaper in terms of total CPU cost than using an OSGi JVM server.

The response times for the COBOL and WebSphere Liberty JVM server versions were similar and CICS suspend time was the main contributor to this time. At the transaction rate used, waiting for first CICS dispatch (recorded in the CICS monitor data field DSPDELAY) accounted for most of the time the transactions were suspended. A longer response time was observed for the OSGi JVM server version. This was due to a longer dispatch time caused by the T8 TCB being placed into an operating wait state for much longer times by the less efficient Java and UNIX thread management of the OSGi JVM server.

For more details about the performance testing described previously, see the following article in the CICS Developer Center:

<https://ibm.biz/BdjUpN>

7.16.4 Comparison of CICS monitoring and RMF data

The CPU data presented in Table 7-5 on page 145 was obtained using RMF because CICS monitoring facility (CMF) data does not account for all the CPU time consumed by a CICS region. Time spent on non-CICS TCBs, the SRB time for networking, or the overhead to initialize and terminate a transaction is not included in monitoring data.

Figure 7-26 plots the CPU per transaction for the Link to WebSphere Liberty workload, obtained using both RMF and CMF data.

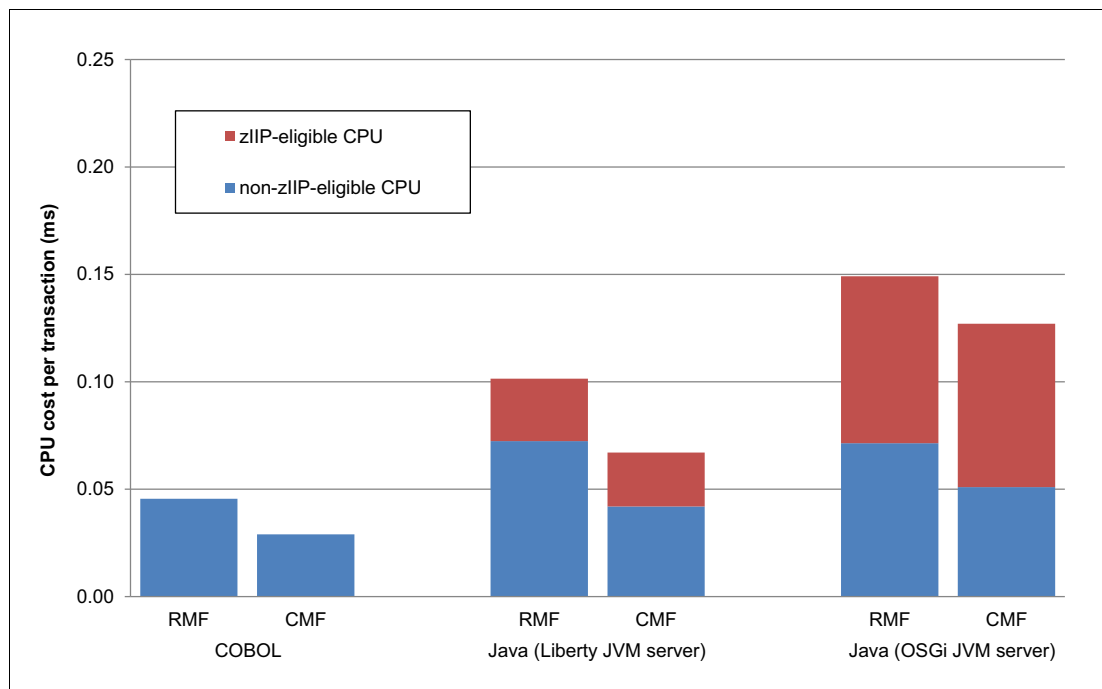


Figure 7-26 Comparison of RMF and CMF data for the Link to WebSphere Liberty workload

Comparing the total CPU cost obtained using the CMF and the RMF data, the COBOL example shows this difference is about 0.017ms.

For the OSGi JVM server example this difference is about 0.022ms. This difference includes the additional overhead of running in a JVM environment on non-CICS managed TCBs, such as JIT compilation and some garbage collection.

The WebSphere Liberty JVM server example shows a much greater difference of about 0.034ms. This greater difference is attributed to the additional overhead of running a WebSphere Liberty server, with time spent on extra non-CICS managed TCBs (UNIX pthreads) used by WebSphere Liberty support functions.

In all these cases the percentage difference between RMF and CICS monitoring data is maximized by the trivial nature of the application. For a more complex application the relative difference would be smaller. When measuring the total cost for a workload, it is important to note this discrepancy between CICS monitoring data and RMF data for any application type. Java applications will typically show a greater delta between RMF and CMF data due to the non-CICS TCBs used for normal JVM operation.



CICS TS for z/OS V5.4

IBM CICS Transaction Server for z/OS (CICS TS) V5.4 release introduces various technical and operational capabilities. Included in these updates are improvements that provide performance benefits over previous CICS releases. Included in the CICS V5.4 performance report are the following subject areas:

- ▶ Key performance benchmarks that are presented as a comparison with the CICS TS V5.3 release.
- ▶ An outline of improvements made regarding the threadsafe characteristics of the CICS TS run time.
- ▶ Details of the changes that are made to performance-critical CICS initialization parameters, and the effect of these updates.
- ▶ A description of all the updated statistics and monitoring fields.
- ▶ A description of the performance class monitoring field contents for various types of WebSphere Liberty requests.
- ▶ High-level views of new functionality that was introduced in the CICS TS V5.4 release, including performance benchmark results where appropriate.

This chapter includes the following topics:

- ▶ 8.1, “Introduction” on page 150
- ▶ 8.2, “Release-to-release comparisons” on page 150
- ▶ 8.3, “Improvements in threadsafety” on page 161
- ▶ 8.4, “Changes to system initialization parameters” on page 162
- ▶ 8.5, “Changes to resource definition attribute default values” on page 164
- ▶ 8.6, “Enhanced instrumentation” on page 165
- ▶ 8.7, “CICS tasks for WebSphere Liberty applications” on page 172
- ▶ 8.8, “z/OS WLM Health API” on page 173
- ▶ 8.9, “Asynchronous API” on page 173
- ▶ 8.10, “EXCI support for channels and containers” on page 182
- ▶ 8.11, “CICS support for IBM Health Checker for z/OS” on page 188
- ▶ 8.12, “Web services performance” on page 189

8.1 Introduction

When compiling the results for this chapter, the workloads were executed on an IBM z13 model NE1 (machine type 2964). A maximum of 32 dedicated CPs were available on the measured LPAR, with a maximum of six dedicated CPs available to the LPAR used to simulate users. These LPARs are configured as part of a parallel sysplex. An internal coupling facility was co-located on the same central processor complex (CPC) as the measurement and driving LPARs, connected using internal coupling peer (ICP) links. An IBM System Storage DS8870 (machine type 2424) was used to provide external storage.

This chapter presents the results of several performance benchmarks when executed in a CICS TS for z/OS V5.4 environment. Unless otherwise stated in the results, the CICS V5.4 environment was the code available at GA time. Several of the performance benchmarks are presented in the context of a comparison against CICS TS V5.3. The CICS TS V5.3 environment contained all PTFs issued before 24th February 2017. All LPARs used z/OS V2.2.

For a definition of performance terms used in this chapter, see Chapter 1, “Performance terminology” on page 3. A description of the test methodology used can be found in Chapter 2, “Test methodology” on page 11. For a full description of the workloads used, see Chapter 3, “Workload descriptions” on page 21.

Where reference is made to an *LSPR processor equivalent*, the indicated machine type and model can be found in the large systems performance reference (LSPR) document. For more information about obtaining and using LSPR data, see 1.3, “Large Systems Performance Reference” on page 6.

8.2 Release-to-release comparisons

This section describes some of the results from a selection of regression workloads that are used to benchmark development releases of CICS TS. For more information about the use of regression workloads, see Chapter 3, “Workload descriptions” on page 21.

8.2.1 Data Systems Workload static routing

The static routing variant of the Data Systems Workload (DSW) is described in 3.2.1, “DSW static routing” on page 22. This section presents the performance figures that were obtained by running this workload. Table 8-1 lists the results of the DSW static routing workload that used the CICS TS V5.3 release.

Table 8-1 CICS TS V5.3 results for DSW static routing workload

ETR	CICS CPU	CPU per transaction (ms)
1244.31	77.52%	0.623
1428.17	88.74%	0.621
1587.05	98.00%	0.617
1789.21	110.18%	0.616
2043.24	129.54%	0.634

Table 8-2 lists the same figures for the CICS TS V5.4 release.

Table 8-2 CICS TS V5.4 results for DSW static routing workload

ETR	CICS CPU	CPU per transaction (ms)
1240.69	76.90%	0.620
1425.05	87.84%	0.616
1582.76	96.98%	0.613
1784.61	108.81%	0.610
2038.77	128.18%	0.629

The average CPU per transaction value for CICS TS V5.3 is calculated to be 0.622 ms. The same value for CICS TS V5.4 is calculated to be 0.617 ms. The performance of this workload is within 1% and, therefore, considered to be equivalent for the two releases.

The figures from Table 8-1 on page 150 and Table 8-2 are plotted in the chart in Figure 8-1.

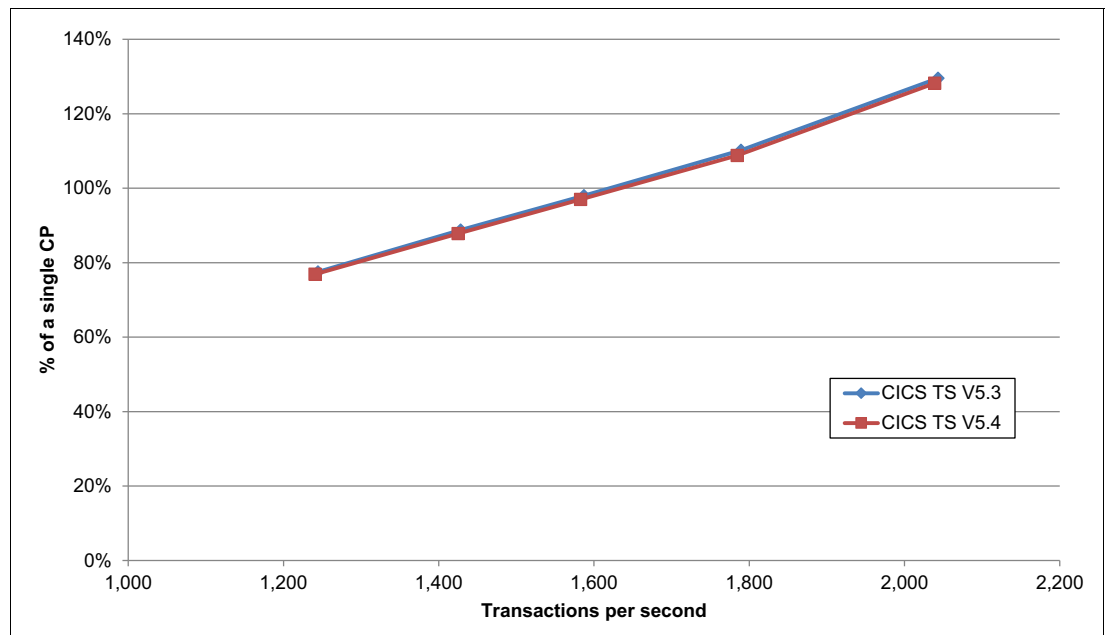


Figure 8-1 Plot of CICS TS V5.3 and V5.4 performance figures for DSW static routing workload

The measured CPU cost for each transaction rate is similar for CICS TS V5.3 and V5.4, which is demonstrated by the two plot lines being almost identical.

8.2.2 Data Systems Workload dynamic routing

The dynamic routing variant of the Data Systems Workload (DSW) is described in 3.2.2, “DSW dynamic routing” on page 23. This section presents the performance figures that were obtained by running this workload.

Table 8-3 lists the performance results of the DSW dynamic routing workload that used the CICS TS V5.3 release.

Table 8-3 CICS TS V5.3 results for DSW dynamic routing workload

ETR	CICS CPU	CPU per transaction (ms)
6139.73	253.15%	0.412
8974.81	365.76%	0.408
12099.66	483.50%	0.400
16444.72	661.87%	0.402
21386.62	871.39%	0.407

Table 8-4 lists the results of the DSW dynamic routing workload that used the CICS TS V5.4 release.

Table 8-4 CICS TS V5.4 results for DSW dynamic routing workload

ETR	CICS CPU	CPU per transaction (ms)
6140.75	252.93%	0.412
8968.03	366.44%	0.409
12119.32	491.94%	0.406
16433.46	667.78%	0.406
21282.97	871.85%	0.410

The average CPU per transaction value for CICS TS V5.3 is calculated to be 0.406 ms. The same value for CICS TS V5.4 is calculated to be 0.408 ms. The difference between the two values is less than 1% and within experimental error; therefore, the performance of this workload is considered to be equivalent for the two releases.

The results from Table 8-3 and Table 8-4 are shown in Figure 8-2 on page 153.

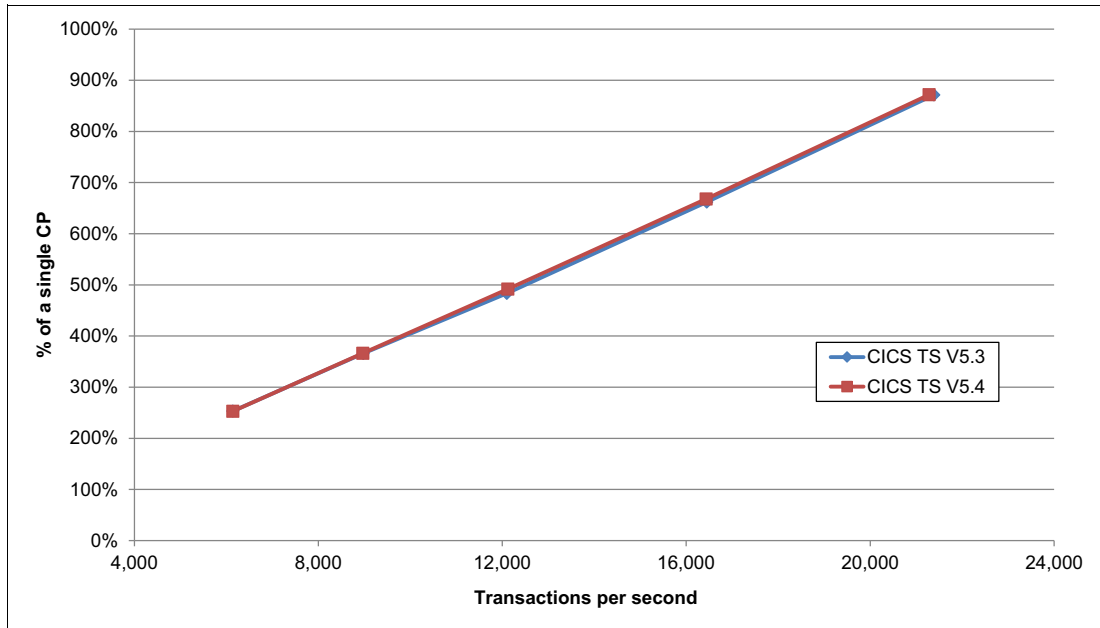


Figure 8-2 Plot of CICS TS V5.3 and V5.4 performance figures for DSW dynamic routing workload

The measured CPU cost for each transaction rate is similar for CICS TS V5.3 and V5.4, which is demonstrated graphically by the two plot lines being almost identical. A final observation is that the total CPU consumed by the CICS regions scales linearly in accordance with the transaction rate.

8.2.3 The Data Systems Workload HTTP interface

The HTTP interface variant of the Data Systems Workload (DSW) uses the same application logic and topology as the DSW static routing workload as described in 3.2.1, “DSW static routing” on page 22. The HTTP interface variant uses HTTP flows to drive the workload in a manner similar to the 3270 terminals used in the static routing workload. This section presents the performance figures that were obtained by running this HTTP workload.

Table 8-5 lists the results of the DSW HTTP interface workload that used the CICS TS V5.3 release.

Table 8-5 CICS TS V5.3 results for DSW HTTP interface workload

ETR	CICS CPU	CPU per transaction (ms)
1181.82	143.19%	1.212
1358.17	163.20%	1.202
1508.77	180.73%	1.198
1694.42	203.45%	1.201
1933.20	241.80%	1.251

Table 8-6 lists the same figures for the CICS TS V5.4 release.

Table 8-6 CICS TS V5.4 results for DSW HTTP interface workload

ETR	CICS CPU	CPU per transaction (ms)
1180.34	141.33%	1.197
1356.36	161.88%	1.193
1506.96	179.39%	1.190
1692.79	199.82%	1.180
1931.97	236.78%	1.226

The average CPU per transaction value for CICS TS V5.3 is calculated to be 1.213 ms. The same value for CICS TS V5.4 is calculated to be 1.197 ms. The difference in performance is within measurable limits and, therefore, is considered to be equivalent for the two releases.

The figures from Table 8-5 on page 153 and Table 8-6 are plotted in the chart in Figure 8-3.

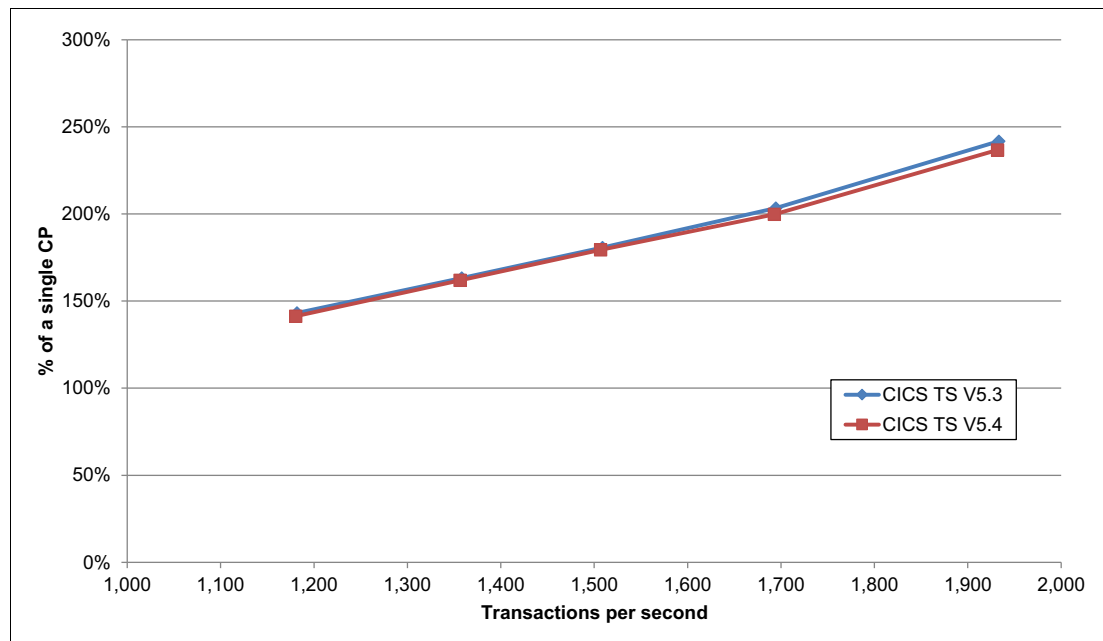


Figure 8-3 Plot of CICS TS V5.3 and V5.4 performance figures for DSW HTTP interface workload

The measured CPU cost for each transaction rate is similar for CICS TS V5.3 and V5.4, which is demonstrated by the two plot lines being almost identical.

8.2.4 The Relational Transactional Workload threadsafe

The Relational Transactional Workload (RTW) is described in 3.3, “Relational Transactional Workload” on page 25. This section presents the performance figures that were obtained by running this workload.

Table 8-7 lists the performance results for the RTW threadsafe workload that used the CICS TS V5.3 release.

Table 8-7 CICS TS V5.3 results for the RTW threadsafe workload

ETR	CICS CPU	CPU per transaction (ms)
333.44	44.11%	1.323
499.59	66.11%	1.323
713.15	94.10%	1.319
996.18	131.52%	1.320
1241.70	163.99%	1.321

Table 8-8 lists the performance results for the RTW threadsafe workload that used the CICS TS V5.4 release.

Table 8-8 CICS TS V5.4 results for the RTW threadsafe workload

ETR	CICS CPU	CPU per transaction (ms)
333.47	44.47%	1.334
499.70	66.52%	1.331
713.23	94.15%	1.320
996.34	131.97%	1.325
1241.71	165.19%	1.330

The average CPU per transaction figure for CICS TS V5.3 is calculated to be 1.321 ms. The CICS TS V5.4 figure is calculated to be 1.328 ms. The difference between these two figures is 0.5%, which is within our measurement accuracy of $\pm 1\%$; therefore, the performance of the two releases is considered to be equivalent.

These figures are shown in Figure 8-4 on page 156.

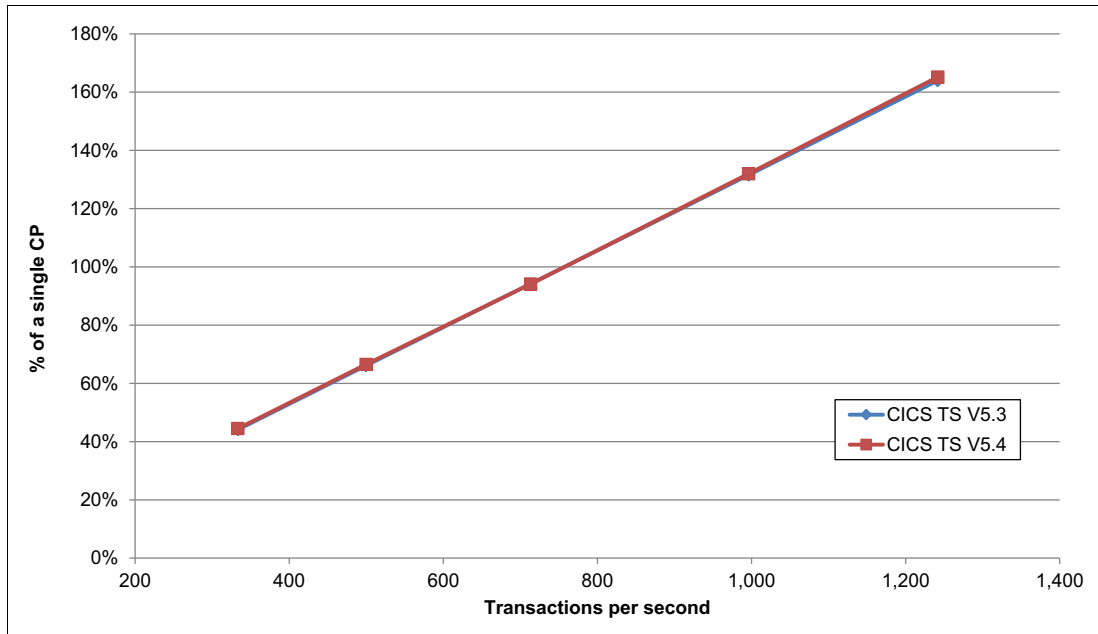


Figure 8-4 Plot of CICS TS V5.3 and V5.4 performance figures for RTW threadsafe workload

As shown in Figure 8-4, the lines are straight, which indicates linear scaling as transaction throughput increases. The lines also are overlaid, which indicates equivalent performance when the releases are compared.

8.2.5 The Java servlet that uses JDBC and VSAM

The Java servlet application is hosted in a CICS JVM server that uses the embedded WebSphere Liberty server. The workload is driven through HTTP requests by using IBM Workload Simulator for z/OS, as described in 2.4, “Driving the workload” on page 16. The servlet application accesses VSAM data by using the JCICS API and accesses DB2 by using the JDBC API. For more information about the workload, see 3.4, “WebSphere Liberty servlet with JDBC and JCICS access” on page 26.

The hardware used for the benchmarks is described in 8.1, “Introduction” on page 150. The measurement LPAR was configured with three GCPs and one zIIP, which resulted in an LSPR equivalent processor of 2964-704.

The CICS TS V5.3 and V5.4 releases were compared by using the software levels as described in 8.1, “Introduction” on page 150. Both configurations used a single CICS region and the following additional software levels and configuration options:

- ▶ DB2 V12
- ▶ Java 8 SR3 (64-bit)
- ▶ Single JVMSERVER resource with THREADLIMIT=256

Both JVM servers used the following JVM options:

- ▶ -Xgcpolicy:gencon
- ▶ -Xcompressedrefs
- ▶ -XXnosuballoc32bitmem
- ▶ -Xmx200M
- ▶ -Xms200M
- ▶ -Xmnx60M

- ▶ -Xmns60M
- ▶ -Xmox140M
- ▶ -Xmos140M

As described in 2.3.1, “Repeatability for Java workloads” on page 14, this workload requires a warm-up period of 20 minutes. After this warm-up phase completed, the request injection rate was increased every 10 minutes. CPU usage data was collected using IBM z/OS Resource Measurement Facility (RMF). An average CPU per request value was calculated using the last 5 minutes of each 10-minute interval.

Table 8-9 lists the performance results of the Java servlet workload that used the CICS TS V5.3 release, with the following columns:

- ▶ **ETR**
The average External Throughput Rate (ETR) during the measurement interval. For more details, see 1.3.1, “External throughput rate” on page 7.
- ▶ **CICS CPU not zIIP-eligible**
The fraction of a single CPU consumed by the whole address space during the measurement interval that was not eligible for execution on a zIIP engine. Extracted from the CP field in an RMF “Workload Activity” report for a report class.
- ▶ **CICS CPU zIIP-eligible**
The fraction of a single CPU consumed by the whole address space that was eligible for execution on a zIIP engine. Calculated as the sum of the IIP and IIPCP fields in an RMF “Workload Activity” report for a report class.
- ▶ **CICS CPU total**
The fraction of a single CPU consumed by the whole address space during the measurement interval. Calculated as the sum of the CP and IIP fields in an RMF “Workload Activity” report for a report class. This value is equal to the sum of the *CICS CPU not zIIP-eligible* and *CICS CPU zIIP-eligible* columns.

Table 8-9 CICS TS V5.3 results for WebSphere Liberty JDBC and VSAM workload

ETR	CICS CPU not zIIP-eligible	CICS CPU zIIP-eligible	CICS CPU total
837.32	27.45%	52.26%	79.71%
1637.60	53.71%	83.67%	137.38%
2289.29	80.11%	93.30%	173.41%

Table 8-10 lists the performance results of the Java servlet workload that used the CICS TS V5.4 release in the same format as Table 8-9.

Table 8-10 CICS TS V5.4 results for WebSphere Liberty JDBC and VSAM workload

ETR	CICS CPU not zIIP-eligible	CICS CPU zIIP-eligible	CICS CPU total
837.28	25.13%	49.53%	74.65%
1646.47	53.57%	85.10%	138.67%
2288.08	79.89%	95.61%	175.49%

The CICS CPU total values from Table 8-9 on page 157 and Table 8-10 on page 157 are plotted in Figure 8-5.

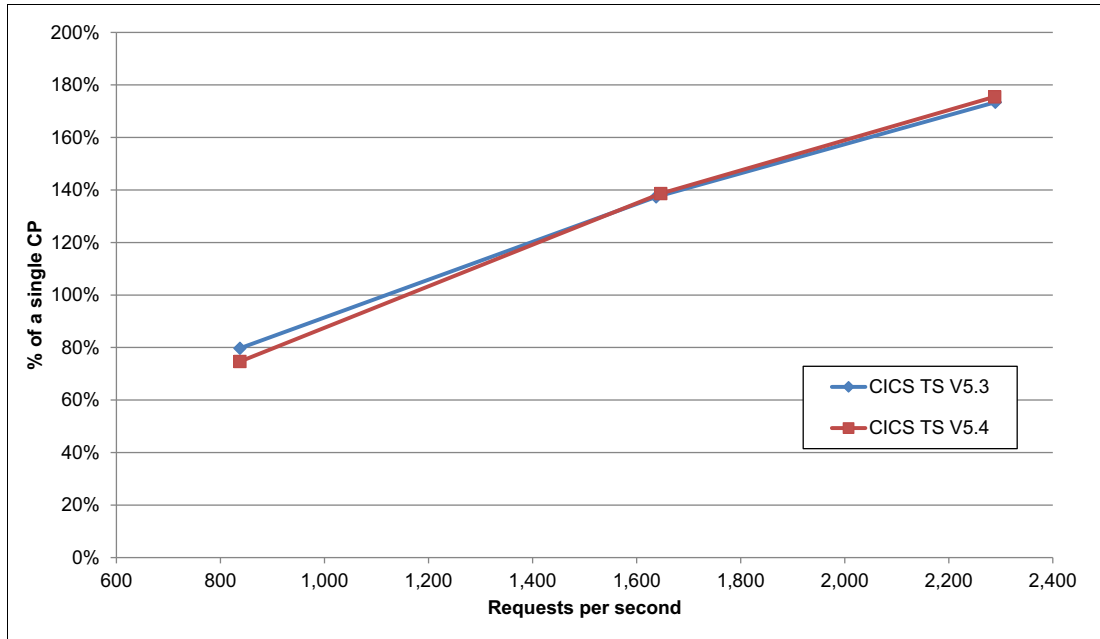


Figure 8-5 Total CPU comparison for CICS TS V5.3 and CICS TS V5.4 JDBC and VSAM workload

The offload eligibility figures are presented as a chart in Figure 8-6.

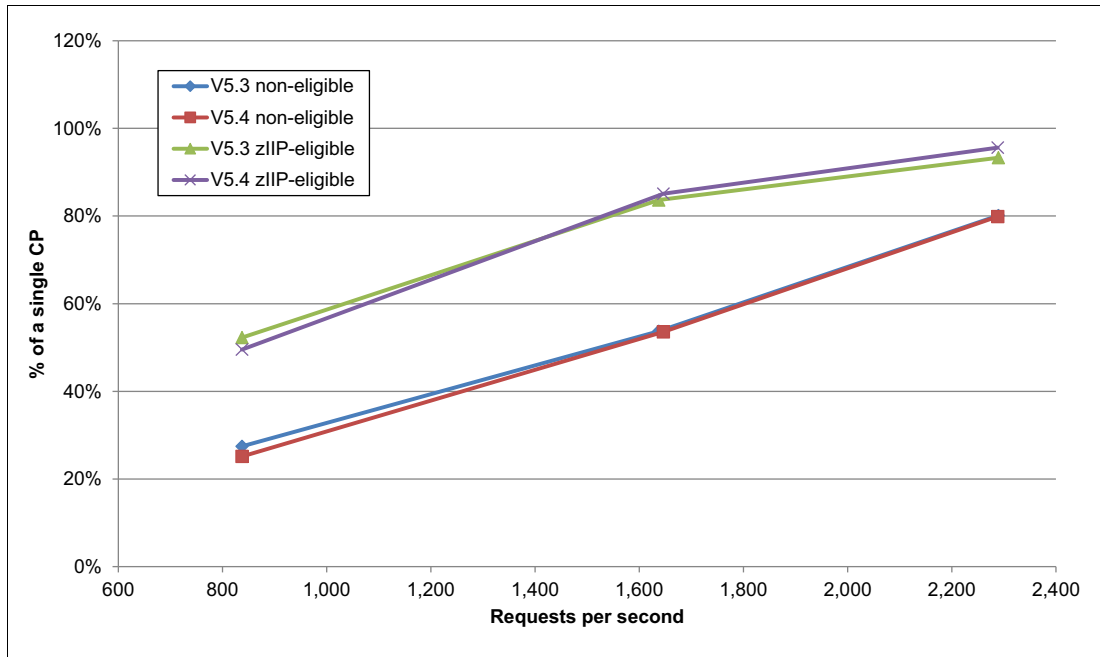


Figure 8-6 Comparing offload-eligible CPU utilization for servlet workload with CICS TS V5.3 and V5.4

The average CPU per transaction value for the JDBC and VSAM workload using the CICS TS V5.3 release is calculated to be 0.849 ms. The same value for the CICS TS V5.4 configuration is calculated to be 0.834 ms.

Table 8-9 on page 157 and Table 8-10 on page 157 present *CICS CPU zIIP-eligible* and *CICS CPU total* data, both as utilization fractions of a single CPU. From these figures, an overall zIIP eligibility figure for the workload is calculated, which is defined as the fraction of the total CPU consumed that was eligible to execute on a zIIP engine.

This value expressed mathematically is as follows:

$$\text{zIIP eligibility} = \frac{\text{CICS CPU zIIP-eligible}}{\text{CICS CPU total}}$$

The CICS TS V5.3 workload had an average zIIP eligibility value of 60.1%, and the CICS TS V5.4 configuration average value was 60.7%.

The average CPU per transaction and the zIIP eligibility calculations show that the CICS TS V5.4 release demonstrates similar performance to the CICS TS V5.3 release for this workload. This calculation is true for both the total CPU consumed and the fraction, which is eligible for offload to a zIIP engine.

Note: A performance improvement for WebSphere Liberty workloads was delivered by APAR PI54263 in CICS TS V5.3. A summary of this improvement is described in 7.15, “WebSphere Liberty zIIP eligibility” on page 143.

This performance improvement included in CICS TS V5.4. APAR PI54263 is also included in the CICS TS V5.3 configuration that was used as baseline measurements for this section. Therefore, no performance improvements are observed in this section because both releases include the zIIP eligibility improvements introduced by APAR PI54263.

8.2.6 The Java OSGi workload

The Java OSGi workload is composed of several applications, as described in 3.5, “Java OSGi workload” on page 27.

The hardware used for the benchmarks is described in 8.1, “Introduction” on page 150. The measurement LPAR was configured with three GCPs and one zIIP, which resulted in an LSPR equivalent processor of 2964-704.

The CICS TS V5.3 and V5.4 releases were compared by using the software levels as described in 8.1, “Introduction” on page 150. Both configurations used a single CICS region and the following additional software levels and configuration options:

- ▶ DB2 V12
- ▶ Java 8 SR4 (64-bit)
- ▶ Single JVMSERVER resource with THREADLIMIT=25

Both JVM servers used the following JVM options:

- ▶ -Xgcpolicy:gencon
- ▶ -Xcompressedrefs
- ▶ -XXnosuballoc32bitmem
- ▶ -Xmx100M
- ▶ -Xms100M
- ▶ -Xmnx70M
- ▶ -Xmns70M
- ▶ -Xmox30M
- ▶ -Xmos30M

As described in 2.3.1, “Repeatability for Java workloads” on page 14, this workload requires a warm-up period of 20 minutes. After this warm-up phase completed, the request injection rate was increased every 5 minutes. CPU usage data was collected using IBM z/OS Resource Measurement Facility (RMF). An average CPU per request value was calculated using the last minute of each 5-minute interval.

Table 8-11 lists the performance results of the Java OSGi workload that used the CICS TS V5.3 release.

Table 8-11 CICS TS V5.3 performance results for OSGi workload

ETR	CICS CPU not zIIP-eligible	CICS CPU zIIP-eligible	CICS CPU total
233.98	22.27%	74.51%	96.78%
467.93	43.26%	148.27%	191.53%
780.13	78.72%	249.64%	328.36%

The performance results for the CICS TS V5.4 release are shown in Table 8-12.

Table 8-12 CICS TS V5.4 performance results for OSGi workload

ETR	CICS CPU not zIIP-eligible	CICS CPU zIIP-eligible	CICS CPU total
233.98	22.68%	72.18%	94.86%
467.97	44.32%	144.65%	188.97%
779.38	81.19%	242.10%	323.29%

The CICS CPU total values from Table 8-11 and Table 8-12 are plotted in Figure 8-7.

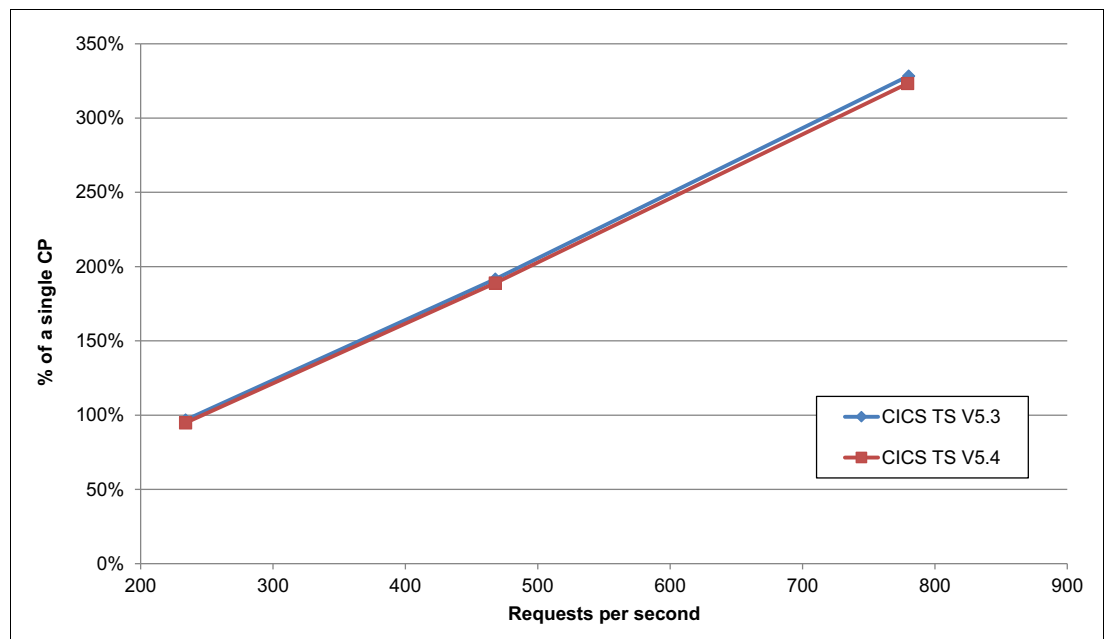


Figure 8-7 Comparing overall CPU utilization for Java OSGi workload with CICS TS V5.3 and V5.4

The offload eligibility figures are presented as a chart in Figure 8-8.

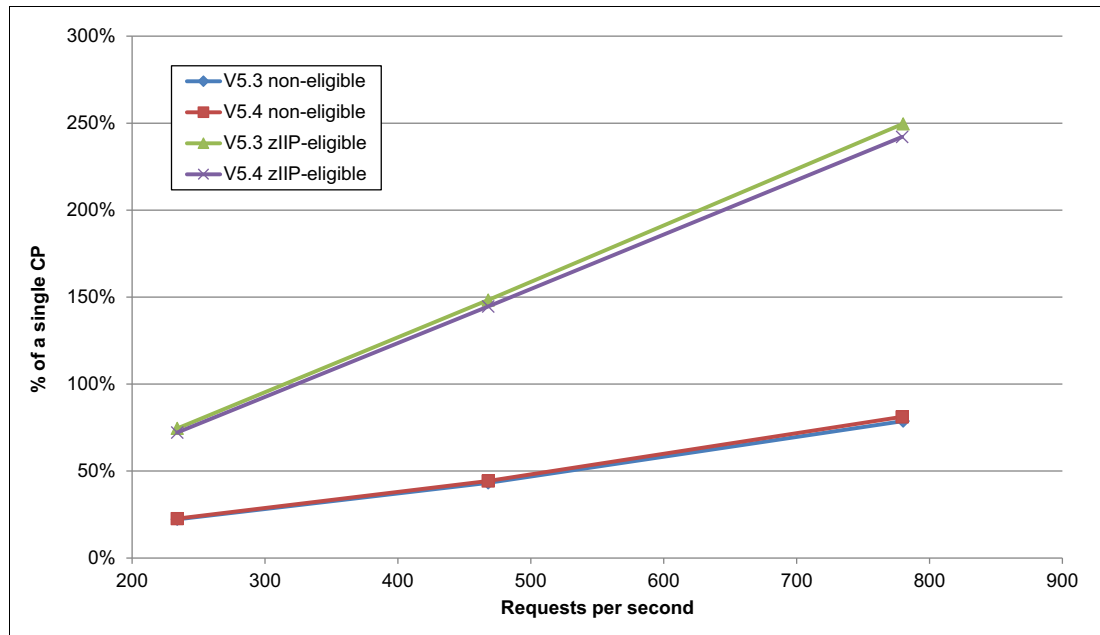


Figure 8-8 Comparing offload-eligible CPU utilization for OSGi workload with CICS TS V5.3 and V5.4

The average CPU per transaction value for this workload using the CICS TS V5.3 release is calculated to be 4.146 ms. The same value for the CICS TS V5.4 release is calculated to be 4.080 ms.

Using the methodology to calculate the zIIP eligibility of the workload described in 8.2.5, “The Java servlet that uses JDBC and VSAM” on page 156, the CICS TS V5.3 release had an average zIIP eligibility of 76.8%. The CICS TS V5.4 release had an average zIIP eligibility of 75.8%.

As observed with the Java servlet workload, the performance of Java OSGi applications is similar in CICS TS V5.4 when compared to CICS TS V5.3, both in terms of total CPU consumed and the fraction that is eligible for offload to a zIIP engine.

8.3 Improvements in threadsafety

All new CICS application programming interface (API) commands in CICS V5.4 are threadsafe. Also, an improvement for Java workloads running in an OSGi JVM server when accessing IBM MQ is implement.

8.3.1 Threadsafe API commands

The following new CICS API commands are threadsafe:

- ▶ FETCH ANY
- ▶ FETCH CHILD
- ▶ FREE CHILD
- ▶ RUN TRANSID
- ▶ TRANSFORM DATATOJSON
- ▶ TRANSFORM JSONTODATA

For more information about CICS API commands, see the “CICS command summary” topic in IBM Knowledge Center at this website:

<https://ibm.biz/Bdi6Gm>

8.3.2 Use of T8 TCB for MQ calls from Java

Java applications running in an OSGi JVM server no longer requires a task control block (TCB) to change mode operation when accessing IBM MQ, using either the MQ classes for Java or the MQ classes for JMS. In a CICS JVM server, Java applications run on T8 TCBs. In CICS TS V5.3, MQ calls from a Java environment required a switch to an L8 TCB. In CICS TS V5.4, this switch is removed.

The change in TCB switch behavior affects the results you see in CICS monitoring and statistics. TCB use for Java MQ applications is changed so that MQ CPU time is now accumulated against a T8 TCB. End-of-task sync point processing is still accumulated on an L8 TCB.

In releases prior to CICS TS V5.4, Java applications running in a WebSphere Liberty JVM server do not use the CICS MQCONN resource and, therefore, do not require TCB change mode operation, using either the MQ classes for Java or the MQ classes for JMS.

8.4 Changes to system initialization parameters

Several performance-related CICS system initialization table (SIT) parameters are changed in the CICS TS V5.4 release. This section outlines changes to the SIT parameters that have the most impact on CICS performance. All comparisons to previous limits or default values refer to CICS TS V5.3.

For a detailed view of changes to SIT parameters in the CICS TS V5.4 release, see the “Changes to SIT parameters” section of the “Changes to externals in this release” topic in IBM Knowledge Center at this website:

<https://ibm.biz/Bdi6GK>

8.4.1 The EDSA limit (EDSALIM) parameter

The minimum value for the EDSALIM parameter is increased from 48 MB to 64 MB. The value that is specified for the EDSALIM parameter can be a maximum of 2047 MB, specified in multiples of 1 MB.

For more information, see the “EDSALIM” topic in IBM Knowledge Center at this website:

<https://ibm.biz/Bdi6Ge>

8.4.2 Default runaway task time (ICVR)

The minimum value for the ICVR value is reduced from 500 ms to 250 ms. The default value is reduced from 5000 ms to 2000 ms. The value that is specified for the ICVR parameter can be a maximum of 2,700,000 ms in multiples of 250 ms.

For more information, see the “ICVR” topic in IBM Knowledge Center at this website:

<https://ibm.biz/Bdi6G8>

8.4.3 Maximum open TCBs (MAXOPENTCBS)

The minimum value for the MAXOPENTCBS parameter is increased from 1 to 32.

If the MAXOPENTCBS parameter is not specified, CICS sets the parameter automatically, based on the current value of the MXT system initialization parameter. The value of the MAXOPENTCBS parameter is calculated by using the same algorithm implemented in CICS TS V5.1, as shown in the following example:

$$\text{MAXOPENTCBS} = (2 \times \text{MXT}) + 32$$

For more information about the MAXOPENTCBS parameter, see the “MAXOPENTCBS” topic in IBM Knowledge Center at this website:

<https://ibm.biz/Bdi6GJ>

8.4.4 Maximum SSL TCBs (MAXSSLTCBS)

The default value for the MAXSSLTCBS parameter is increased from 8 to 32.

For more information about the MAXSSLTCBS parameter, see the “MAXSSLTCBS” topic in IBM Knowledge Center at this website:

<https://ibm.biz/Bdi6GA>

8.4.5 RACF synchronization (RACFSYNC)

The RACFSYNC system initialization parameter specifies whether CICS listens for type 71 Event Notification Facility (ENF) events.

RACF sends a type 71 ENF signal to listeners when a **CONNECT**, **REMOVE**, or **REVOKE** command changes a user’s resource authorization. If RACFSYNC=YES is specified in a CICS TS V5.4 environment, when CICS receives a type 71 ENF event for a user ID, all cached user tokens for the user ID are invalidated, irrespective of the setting of the USRDELAY parameter.

Subsequent requests from that user ID force a full **RACF RACROUTE VERIFY** request, which results in a refresh of the user’s authorization level. User tokens for tasks that are currently running are not affected. In addition to the **RACF RACROUTE VERIFY** processing, a type 71 ENF signal will also make DB2 threads for the associated user ID issue a full sign on when they are next reused.

Note: In the configuration where type 71 signals are issued for large numbers of users simultaneously, combined with large numbers of connections to DB2, the temporary performance overhead can be significant while the full sign on processing across all affected DB2 threads is completed.

To reduce the impact of type 71 ENF processing, make updates to large numbers of RACF users during off-peak periods.

For more information about the RACFSYNC parameter, see the “RACFSYNC” topic in IBM Knowledge Center at this website:

<https://ibm.biz/Bdi6G9>

8.4.6 Sign on for a preset user ID (SNPRESET)

The new SNPRESET parameter is provided to allow the sharing of access control environment element (ACEE) control blocks. This process can save 31-bit storage and CPU. The SNPRESET parameter takes one of the following values:

- ▶ UNIQUE

When signing on a preset user ID terminal, the ACEE is built with entry port information. Every preset terminal has a unique ACEE that is associated with the user ID and terminal. This is the default.

- ▶ SHARED

When signing on a preset user ID terminal, the ACEE is built without entry port information. All preset terminals with the same user ID use the same ACEE.

If you audit data that is based on the terminal of a preset user ID, use SNPRESET=UNIQUE.

If you do not need information that is based on the terminal of a preset user ID, you can save storage by selecting SNPRESET=SHARED.

Note: In the event of a security violation with SNPRESET=SHARED, the netname of the terminal will not show in the DFHXS1111 message.

For more information about the SNPRESET parameter, see the “SNPRESET” topic in IBM Knowledge Center at this website:

<https://ibm.biz/Bdi6G3>

8.4.7 z/OS Workload Manager Health API (WLMHEALTH)

CICS TS V5.4 introduces the new SIT parameter WLMHEALTH. For more details about the use of the z/OS Workload Manager Health API in CICS, see 8.8, “z/OS WLM Health API” on page 173.

For more information about the WLMHEALTH parameter, see the “WLMHEALTH” topic in IBM Knowledge Center at this website:

<https://ibm.biz/Bdi55P>

8.5 Changes to resource definition attribute default values

Several performance-related CICS resource definition attributes are changed in the CICS TS V5.4 release. This section outlines changes to the default attribute values that have the most impact on CICS performance. All comparisons to previous limits or default values refer to CICS TS V5.3.

For a detailed view of changes to SIT parameters in the CICS TS V5.4 release, see the “Changes to resource definitions” section of the “Changes to externals in this release” topic in IBM Knowledge Center at this website:

<https://ibm.biz/Bdi6GK>

8.5.1 The PROGRAM resource

The default value for the DATALOCATION attribute changed from BELOW to ANY. Commands that use the **SET** option can return a data address to an application program; this operand specifies the location of the data. For example, in the **EXEC CICS RECEIVE SET(ptr-ref)** command, *ptr-ref* is in 24-bit virtual storage if DATALOCATION(BELOW) is specified but might be in 31-bit virtual storage if DATALOCATION(ANY) is specified. DATALOCATION does not affect the operation of the **GETMAIN** command.

For more information about the DATALOCATION attribute, see the “PROGRAM attributes” topic in IBM Knowledge Center at this website:

<https://ibm.biz/Bdi5zj>

8.5.2 The TRANSACTION resource

The minimum value that can be specified for the RUNAWAY attribute is reduced from 500 ms to 250 ms. The value that is specified for the RUNAWAY attribute can be a maximum of 2,700,000 ms in multiples of 250 ms.

The default value for the TASKDATALOC attribute has changed from BELOW to ANY. A value of ANY indicates storage acquired by CICS for the duration of the transaction can be located in 31-bit virtual storage. These areas, which relate to specific CICS tasks, include the EXEC interface block (EIB) and the transaction work area (TWA).

The default value for the SPURGE attribute has changed from NO to YES. A value of YES indicates a transaction is initially system purgeable.

The default value for the TPURGE attribute has changed from NO to YES. A value of YES indicates, for non-z/OS Communications Server terminals only, that the transaction can be purged because of a terminal error.

For more information about the RUNAWAY, TASKDATALOC, SPURGE, and TPURGE attributes, see the “TRANSACTION attributes” topic in IBM Knowledge Center at this website:

<https://ibm.biz/Bdi5zY>

8.6 Enhanced instrumentation

The CICS TS V5.4 release enhances the information reported by the CICS monitoring and statistics components. This section describes the extra and changed fields that are now available in the CICS monitoring and statistics SMF records.

For more information about changes in monitoring fields across a range of CICS releases, see the “Changes to CICS monitoring” topic in IBM Knowledge Center at this website:

<https://ibm.biz/Bdi5UZ>

For more information about changes in statistics fields across a range of CICS releases, see the “Changes to CICS statistics” topic in IBM Knowledge Center at this website:

<https://ibm.biz/Bdi5UY>

8.6.1 The DFHCICS performance group

The following fields were added to the DFHCICS performance group to provide support for policy system rules:

- ▶ Policy system rules evaluation count (field MPSRECT)
The number of times that policy system rules have been evaluated for the task.
- ▶ Policy system rules trigger count (field MPSRACT)
The number of times that policy system rules that have evaluated to true and have triggered either a message or an event.

For more information about policy system rules, see the topic “Policy system rules” in IBM Knowledge Center at this website:

<https://ibm.biz/Bdi5U2>

Several fields have been added to provide enhanced transaction tracking information when using the asynchronous services (AS) domain:

- ▶ PTSTART
- ▶ PTTRANNO
- ▶ PTTRAN
- ▶ PTCOUNT

For more information about monitoring enhancements for the AS domain, see 8.9, “Asynchronous API” on page 173.

Note: The previous transaction information is also available for any tasks that do not create a new point of origin.

For more information about counters that are available in the DFHCICS performance group, see the topic “Performance data in group DFHCICS” in IBM Knowledge Center at this website:

<https://ibm.biz/Bdi5Uz>

8.6.2 The DFHPROG performance group

The ABCODE0 and ABCODEC monitoring fields can now contain the following abend codes:

- ▶ ASPF
- ▶ ASPN
- ▶ ASP0
- ▶ ASPP
- ▶ ASPQ
- ▶ ASPR
- ▶ ASP1
- ▶ ASP2
- ▶ ASP3
- ▶ ASP7
- ▶ ASP8

Note: Due to the circumstances under which transactions are called, a transaction dump might not be taken when these abends occur.

For more information about data fields that are available in the DFHPR0G performance group, see the “Performance data in group DFHPR0G” topic in IBM Knowledge Center at this website:

<https://ibm.biz/Bdi5Uf>

8.6.3 The DFHTASK performance group

The following field was added to the DFHTASK performance group to aid system identification:

- ▶ Logical partition name (field LPARNAME)
The name, in EBCDIC, of the logical partition (LPAR) on the processor where the CICS region is running.

The following fields are added to the DFHTASK performance group to support the asynchronous services (AS) domain:

- ▶ ASTOTCT
- ▶ ASRUNCT
- ▶ ASFTCHCT
- ▶ ASFREECT
- ▶ ASFTCHWT
- ▶ ASRNATWT

For more information about monitoring enhancements for the AS domain, see 8.9, “Asynchronous API” on page 173.

For more information about data fields that are available in the DFHTASK performance group, see the “Performance data in group DFHTASK” topic in IBM Knowledge Center at this website:

<https://ibm.biz/Bdi6G5>

8.6.4 Transaction resource class data

The following fields are added to the transaction resource class data to support the AS domain:

- ▶ MNR_PTD_ATTACH_TIME
- ▶ MNR_PTD_TRANNUM
- ▶ MNR_PTD_TRANID
- ▶ MNR_PTD_COUNT

For more information about monitoring enhancements for the AS domain, see 8.9, “Asynchronous API”.

For more information about fields that are available in the transaction resource class data, see the “Transaction resource class data: Listing of data fields” topic in IBM Knowledge Center at this website:

<https://ibm.biz/Bdi5UP>

8.6.5 Identity class data

The following fields are added to the identity class data to support the AS domain:

- ▶ MNI_PTD_ATTACH_TIME
- ▶ MNI_PTD_TRANNUM

- ▶ MNI_PTD_TRANID
- ▶ MNI_PTD_COUNT

For more information about monitoring enhancements for the AS domain, see 8.9, “Asynchronous API” on page 173.

For more information about fields that are available in the identity class data, see the “Identity class data: Listing of data fields” topic in IBM Knowledge Center at this website:

<https://ibm.biz/Bdi5Uy>

8.6.6 Asynchronous services global statistics

A new statistics report is added to support the AS domain. This report contains the following fields:

- ▶ ASG_RUN_COUNT
- ▶ ASG_FETCH_COUNT
- ▶ ASG_FREE_COUNT
- ▶ ASG_RUN_DELAY_COUNT
- ▶ ASG_PARENTS_DELAYED_CUR
- ▶ ASG_PARENTS_DELAYED_PEAK
- ▶ ASG_CHILDREN_CUR
- ▶ ASG_CHILDREN_PEAK

For more information about statistics enhancements for the AS domain, see 8.9, “Asynchronous API”.

8.6.7 IBM MQ monitor statistics

New statistics are provided in the CICS TS V5.4 for the MQMONITOR resource. The following fields are available in the new IBM MQ monitor resource statistics report:

- ▶ Monitor name (field MQR_NAME)
 - The name of an installed MQMONITOR definition in the CICS region.
- ▶ Monitor start date and time (field MQR_START_TIME_LOCAL)
 - The local date and time when the most recent instance of the MQ monitor was started.
- ▶ Monitor stop date and time (field MQR_STOP_TIME_LOCAL)
 - The local date and time when the most recent instance of the MQ monitor was stopped.
- ▶ Queue name (field MQR_QNAME)
 - The name of the MQ queue monitored by the MQ monitor.
- ▶ Monitor status (field MQR_MONSTATUS)
 - The status of the MQ monitor, which can be one of the following states:
 - STARTED
 - STARTING
 - STOPPED
 - STOPPING
- ▶ Monitor user ID (field MQR_MONUSERID)
 - The user ID used by the transaction monitoring the MQ queue.

- ▶ Task number (field MQR_TASKNUM)
Task number of the transaction monitoring the MQ queue.
- ▶ Transaction ID (field MQR_TRANID)
The ID of the CICS transaction used by the MQ monitor.
- ▶ User ID (field MQR_USERID)
The user ID to be used by the monitor transaction when issuing the start request for the application transaction if a suitable user ID is not available.
- ▶ Number of OPEN requests (field MQR_TOPEN)
The number of **MQOPEN** calls issued.
- ▶ Number of CLOSE requests (field MQR_TCLOSE)
The number of **MQCLOSE** calls issued.
- ▶ Number of GET requests (field MQR_TGET)
The number of **MQGET** calls issued.
- ▶ Number of GET with wait requests (field MQR_TGETWAIT)
The number of **MQGET** calls issued with the MQGMO_WAIT option.
- ▶ Number of PUT requests (field MQR_TPUT)
The number of **MQPUT** calls issued.
- ▶ Number of PUT1 requests (field MQR_TPUT1)
The number of **MQPUT1** calls issued.
- ▶ Number of INQ requests (field MQR_TINQ)
The number of **MQINQ** calls issued.
- ▶ Number of INQL requests (field MQR_TINQL)
The number of **MQINQL** calls issued.
- ▶ Number of SET requests (field MQR_TSET)
The number of **MQSET** calls issued.
- ▶ Number of committed units of work (field MQR_TCOMMUOW)
The number of UOWs that were in doubt at adapter startup that are now resolved by committing.
- ▶ Number of backed-out units of work (field MQR_TBACKUOW)
The number of UOWs that were in doubt at adapter startup that are now resolved by backing out.
- ▶ Number of other requests (field MQR_TOTHER)
The number of other calls.
- ▶ Monitor GMT start date and time (field MQR_START_TIME_GMT)
The Greenwich mean time (GMT) when the most recent instance of the MQ monitor was started.
- ▶ Monitor GMT stop date and time (field MQR_STOP_TIME_GMT)
The Greenwich mean time (GMT) when the most recent instance of the MQ monitor was stopped.

For more information about IBM MQ Monitor statistics, see the “IBM MQ Monitor statistics” topic in IBM Knowledge Center at this website:

<https://ibm.biz/Bdi5UM>

8.6.8 TCP/IP global statistics

The following fields were added to the TCP/IP global statistics report:

- ▶ Current non-persistent inbound sockets (field SOG_CURR_NPERS_INB_SOCKETS)
The current number of non-persistent inbound sockets.
- ▶ Peak non-persistent inbound sockets (field SOG_PEAK_NPERS_INB_SOCKETS)
The peak number of non-persistent inbound sockets.
- ▶ Peak persistent inbound sockets (field SOG_PEAK_PERS_INB_SOCKETS)
The peak number persistent inbound sockets.
- ▶ Total non-persistent inbound sockets created (field SOG_NPERS_INB_SOCKETS_CREATED)
Total number of non-persistent inbound sockets created.
- ▶ Peak outbound sockets (field SOG_PEAK_BOTH_OUTB_SOCKETS)
Peak number of outbound sockets.
- ▶ Total outbound sockets reused (field SOG_TIMES_OUTB_REUSED)
Total number of times outbound sockets reused.
- ▶ Total persistent outbound sockets created (field SOG_PERS_OUTBOUND_CREATED)
Total number of persistent outbound sockets created.

Example 8-1 shows a sample DFHSTUP report that contains the new fields.

Example 8-1 Extract of sample TCP/IP global statistics report produced by CICS TS V5.4 DFHSTUP

Current number of inbound sockets	109
Current number of non-persistent inbound sockets.	0
Peak number of inbound sockets.	109
Peak number of non-persistent inbound sockets	0
Peak number of persistent inbound sockets	109
Total number of inbound sockets created	0
Total number of non-persistent inbound sockets created.	0
Current number of non-persistent outbound sockets	500
Current number of persistent outbound sockets	0
Peak number of outbound sockets	500
Peak number of non-persistent outbound sockets.	500
Peak number of persistent outbound sockets.	0
Total number of times outbound sockets reused	301381
Total number of outbound sockets created.	0
Total number of persistent outbound sockets created	0
Total number of outbound sockets closed	0
Total number of inbound and outbound sockets created.	0

For more information about TCP/IP global statistics, see the “TCP/IP: Global statistics” topic in IBM Knowledge Center at this website:

<https://ibm.biz/Bdi5US>

8.6.9 TCP/IP services resource statistics

The following fields were added to the statistics for each installed TCP/IP service:

- ▶ **Current Maximum Backlog (field SOR_CURR_MAX_BACKLOG)**
The maximum number of connection requests that the TCP/IP service currently allows in its backlog queue, summed over all appropriate stacks if the TCP/IP service is listening on multiple stacks.
- ▶ **TCP/IPSERVICE Backlog Setting (field SOR_BACKLOG)**
The initial backlog setting for the TCP/IP service. The setting controls the maximum number of connection requests that are allowed to queue in the backlog queue for the TCP/IP service before it starts to reject incoming connections. This is per stack if the TCP/IP service is listening on multiple stacks.
- ▶ **Total connections (field SOR_TOTAL_CONNS)**
The total number of connections made for the TCP/IP service.
- ▶ **Requests processed (field SOR_REQUESTS)**
The number of requests processed by the TCP/IP service.
- ▶ **Made non-persistent at MAXPERSIST (field SOR_NONP_AT_MAXPERSIST)**
The number of times a new persistent connection was made non-persistent because MAXPERSIST was reached.
- ▶ **Disconnected after maximum uses (field SOR_DISC_AT_MAX_USES)**
The number of times a persistent HTTP connection was disconnected because its number of uses had exceeded the limit.
- ▶ **Made non-persistent at task limit (field SOR_NONP_AT_TASK_LIMIT)**
The number of times a new persistent HTTP connection was made non-persistent because the number of tasks in the region has exceeded the limit.
- ▶ **Disconnected at task limit (field SOR_DISC_AT_TASK_LIMIT)**
The number of times an existing persistent HTTP connection was closed because the number of tasks in the region has exceeded the limit.
- ▶ **Current backlog (field SOR_CURR_BACKLOG)**
The current number of connection requests waiting in the backlog queue, summed over all appropriate stacks if the TCP/IP service is listening on multiple stacks.
- ▶ **Connections dropped (field SOR_CONNS_DROPPED)**
The total number of connections that were dropped because the backlog queue was full.
- ▶ **Time connection last dropped (field SOR_CONN_LAST_DROPPED)**
The time that a connection was last rejected because the backlog queue of the TCP/IP service was full.

For more information about TCP/IP resource statistics, see the “TCP/IP services: Resource statistics” topic in IBM Knowledge Center at this website:

<https://ibm.biz/Bdi5UG>

8.6.10 z/OS Communications Server global statistics

CICS TS V5.4 introduces the BMS 3270 Intrusion Detection Service that allows CICS to detect if a 3270 emulator has invalidly modified a protected field generated by a BMS map.

For more details about the functionality, see the “BMS 3270 Intrusion Detection Service” topic in IBM Knowledge Center at this website:

<https://ibm.biz/Bdi5Uv>

The following fields were added to the z/OS Communications Server collected statistics to report the status of BMS 3270 validation:

- ▶ BMS 3270 validation status (field A03BMVL)
Indicates whether the BMS 3270 validation User Replaceable Module (URM) is ON or OFF.
- ▶ Number of BMS 3270 validation failures abended (field A03BMAB)
Number of times the BMS 3270 validation URM has detected invalid 3270 data, issued a DFHTF0200 message to log the event, and terminated the transaction with an ABMX abend code.
- ▶ Number of BMS 3270 validation failures ignored (field A03BMIG)
Number of times the BMS 3270 validation URM has detected invalid 3270 data but ignored the detection in response.
- ▶ Number of BMS 3270 Validation Failures Logged (field A03BMLG)
Number of times the BMS 3270 validation URM has detected invalid 3270 data and issued a DFHTF0200 message to log the event.

For more information about z/OS Communication Server global statistics, see the “z/OS Communications Server: Global statistics” topic in IBM Knowledge Center at this website:

<https://ibm.biz/Bdi5Um>

8.7 CICS tasks for WebSphere Liberty applications

For a WebSphere Liberty application to use the JCICS API and other CICS resources, such as a JDBC data source with type 2 connectivity, requests must run under a CICS task. The point at which this task is created is optimized, is dependent upon the type of request, as follows:

- ▶ HTTP requests
A CICS task is created before the WebSphere Liberty application is invoked
- ▶ Non-HTTP requests, such as message-driven beans (MDBs), inbound Java Connector Architecture (JCA) requests, or remote Enterprise Java Beans (EJBs)
A CICS task is created on first use of a CICS resource

As described in 2.6.1, “Collecting Java performance data” on page 19, CPU time spent on a TCB before the CICS task is created will not be captured in the CICS monitoring data. When analyzing CICS performance class monitoring data for non-HTTP tasks, it is important to be aware that *all* monitoring data (including CPU and response time) start only at the point in the application that first accessed a CICS resource.

For more information about the processing of CICS tasks for WebSphere Liberty workloads, see the “CICS tasks for Liberty applications” section of the “Java applications in a Liberty JVM server” topic in IBM Knowledge Center at this website:

<https://ibm.biz/Bdi6ni>

8.8 z/OS WLM Health API

CICS TS V5.4 uses the z/OS Workload Manager (WLM) health API (IWM4HLTH) as a means of controlling the flow of work into a CICS region. This service is used to inform z/OS WLM about the state of health of a server, in this context a CICS region. The health indicator is a number that shows, in percentage terms, how well the server is performing. It can be an integer value between 0 and 100. A value of 100 means the server is fully capable to do work without any health problems, whereas a value of 0 means it is not able to do any work.

As described in 8.4.7, “z/OS Workload Manager Health API (WLMHEALTH)” on page 164, CICS TS V5.4 introduces the new SIT parameter WLMHEALTH. This SIT parameter controls how a CICS region reports health to z/OS WLM both during and just after the startup period.

For more information about how CICS TS V5.4 uses the z/OS WLM Health API to control the flow of work into a CICS region, see the following article in the CICS Developer Center:

<https://developer.ibm.com/cics/2017/05/16/controlling-flow-work-cics/>

Note: The improvements to CICSplex SM workload routing introduced in CICS TS V5.5 was also made available in CICS TS V5.4 with APAR PI90147. See 9.7.1, “CICSplex SM workload routing” on page 211 for more details.

8.9 Asynchronous API

The CICS TS V5.4 release introduces a set of CICS commands which can be used to develop applications that start one or more child tasks. These child tasks execute asynchronously to the parent task and the new commands provide the ability for the parent task to fetch responses from the child tasks. The following API commands now support this programming model:

▶ **EXEC CICS RUN TRANSID**

Initiates a local child transaction that runs asynchronously with the parent transaction.

▶ **EXEC CICS FETCH CHILD**

Used by a parent task to inquire on the status of a specific child task.

▶ **EXEC CICS FETCH ANY**

Used by a parent task to inquire on the status of any completed child task which has not yet been fetched.

▶ **EXEC CICS FREE CHILD**

Used by a parent task which no longer requires the response of a child task, freeing resources associated with that child when it completes.

All of the new API commands are threadsafe. For more information about using the new API commands, see the “Developing for asynchronous requests” topic in IBM Knowledge Center at this website:

<https://ibm.biz/Bdi6nZ>

For an in-depth look at the Asynchronous API, refer to *IBM CICS Asynchronous API: Concurrent Processing Made Simple*, SG24-8411.

8.9.1 Enhanced transaction tracking information

The transaction tracking information in the CICS monitoring facility records is enhanced to provide greater insight into the relationship between parent and child tasks. Existing origin and previous hop data is extended with the concept of previous transaction data.

When work first enters the CICSplex, for example in a web service request or an MQ message, details of its point of origin is placed into task context information called “origin data,” part of its task association data, for the first task that is created to process it. This data, the origin data, flows with the work as it moves around the CICSplex.

As work moves from one CICS region to another, either through a distributed program link (DPL) or through a function ship (FS) request, the previous hop data in the target region is updated to reference the CICS task in the previous CICS region.

Using the **EXEC CICS RUN TRANSID** command and some **EXEC CICS START** commands, child tasks are created which have the parent task in the previous transaction data. Previous transaction data is created for a task when a new point of origin is *not* created. A task that is a new point of origin will not contain previous transaction data.

Figure 8-9 demonstrates the relationship between origin data, previous hop data, and previous transaction data.

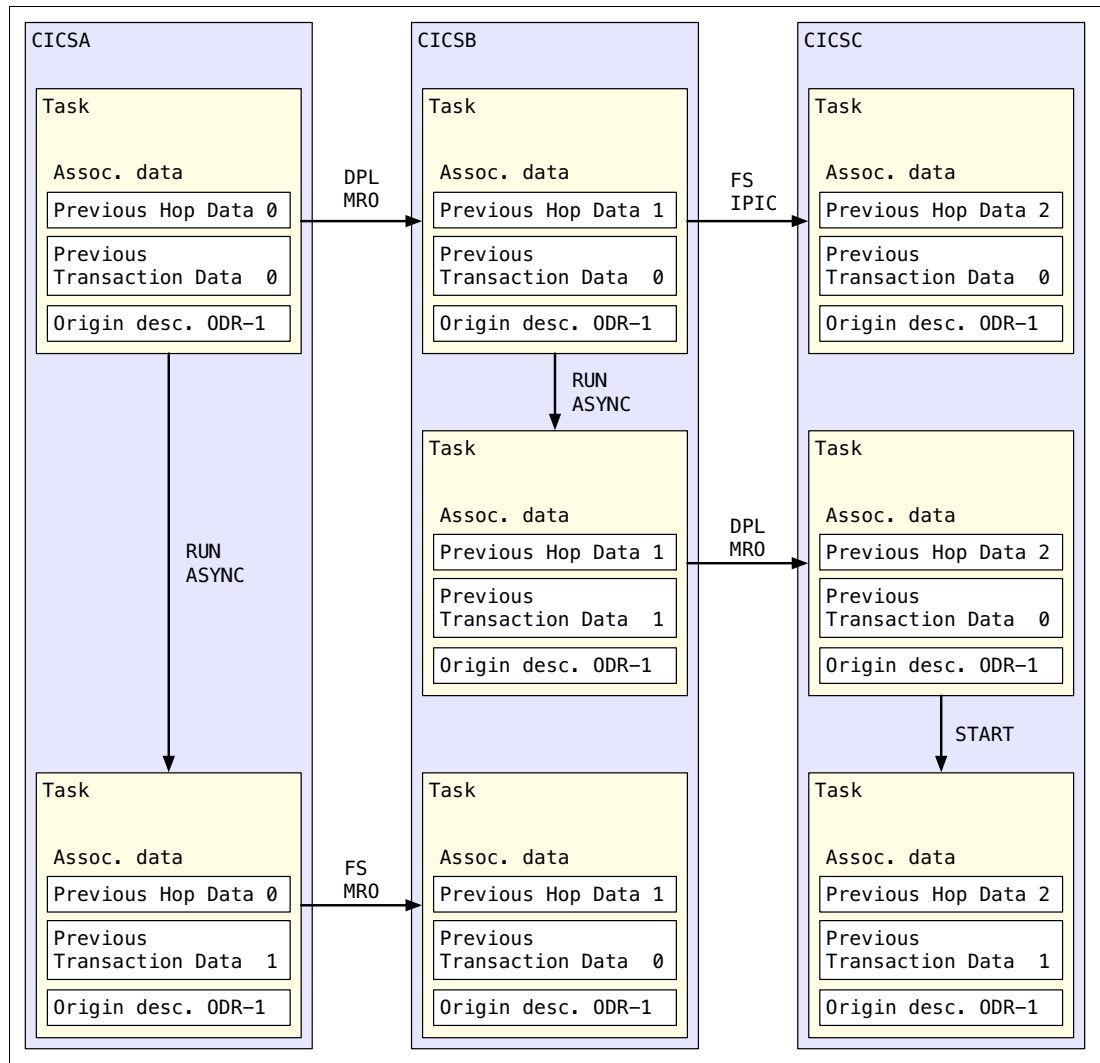


Figure 8-9 Relationship between origin data, previous hop data, and previous transaction data

Several fields have been added to the CICS monitoring fields to record the previous transaction information, and these are discussed in more depth in 8.9.3, “CICS monitoring enhancements” on page 176.

For more information about the previous transaction information, see the “Previous transaction data characteristics” topic in IBM Knowledge Center at this website:

<https://ibm.biz/Bdi6n2>

8.9.2 Workload management

The asynchronous API can result in a large number of concurrent tasks within a CICS system. CICS will automatically begin workload management should a region reach the maximum tasks (MXT) limit, and you can regulate performance yourself using transaction classes.

By specifying a TRANCLASS for parent transactions, you can control the maximum number of parent tasks that will run in a system at any given time, and by extension the number of child tasks that will be created by those parents. Use the MAXACTIVE attribute of a TRANCLASS to ensure that the combined number of parent and child transactions is less than the MXT value for your system.

If using a TRANCLASS for child transactions, the MAXACTIVE value for your child tasks should be higher than the MAXACTIVE value for parent tasks, assuming that a given parent task will create multiple children.

Note: If using a TRANCLASS for child transactions, don't use the same TRANCLASS for child transactions and parent transactions. Otherwise, you can end up with a system full of parent tasks and no space for child tasks to attach.

The workload initiated by **EXEC CICS RUN TRANSID** commands is managed automatically by CICS when a region is under stress. If there are too many child tasks in the system, the parent tasks which are issuing **EXEC CICS RUN TRANSID** commands is suspended and resumed as necessary, limiting the number of new child tasks being created.

For more information about how CICS regulates workflow with asynchronous API commands, see the "Managing performance with the asynchronous API" topic in IBM Knowledge Center at this website:

<https://ibm.biz/Bdi6nz>

8.9.3 CICS monitoring enhancements

The CICS monitoring facility (CMF) data is enhanced to support the new commands that are provided by the asynchronous API.

The DFHCICS performance group

To provide the enhanced transaction tracking information described in 8.9.1, "Enhanced transaction tracking information" on page 174, the following fields are added to the DFHCICS performance group:

- ▶ Previous transaction start time (field PTSTART)
The start time of the immediately previous or parent task in the same CICS system with which the task is associated.
- ▶ Previous transaction number (field PTTRANNO)
The task number of the immediately previous or parent task in the same CICS system with which the task is associated.
- ▶ Previous transaction (field PTTRAN)
The transaction ID of the immediately previous or parent task in the same CICS system with which the task is associated.
- ▶ Previous transaction count (field PTCOUNT)
The number of times there has been a request from one task to initiate another task in the same CICS system with which this task is associated, such as by an **EXEC CICS RUN TRANSID** or **EXEC CICS START** command. This is effectively the task depth in the local region when using the **EXEC CICS RUN TRANSID** command, or the **EXEC CICS START** command when a new point of origin is not created.

The OTRANFLG field has a new transaction origin type for asynchronous transactions.

The DFHTASK performance group

The following fields are added to DFHTASK performance group to provide task-level information when using the asynchronous API commands.

- ▶ Total asynchronous API commands count (field ASTOTCT)
The total number of **EXEC CICS** asynchronous API commands that were issued by the user task.
- ▶ **EXEC CICS RUN TRANSID** command count (field ASRUNCT)
The number of **EXEC CICS RUN TRANSID** commands that were issued by the user task.
- ▶ **EXEC CICS FETCH** command count (field ASFTCHCT)
The number of **EXEC CICS FETCH CHILD** and **EXEC CICS FETCH ANY** commands that were issued by the user task.
- ▶ **EXEC CICS FREE CHILD** command count (field ASFREECT)
The number of **EXEC CICS FREE CHILD** commands that were issued by the user task.
- ▶ Asynchronous API fetch wait time (field ASFTCHWT)
The elapsed time for which the user task waited as the result of an **EXEC CICS FETCH CHILD** or **EXEC CICS FETCH ANY** command.
- ▶ Asynchronous API delay time (field ASRNATWT)
The elapsed time that the user task was delayed as a result of asynchronous child task limits managed by the asynchronous services domain.

The TRANFLAG field has a new transaction origin type for asynchronous API transactions.

Transaction resource class data

To provide the enhanced transaction tracking information described in 8.9.1, “Enhanced transaction tracking information” on page 174, the following fields are added to the transaction resource class data:

- ▶ Previous task attach time (field MNR_PTD_ATTACH_TIME)
The start time of the immediately previous or parent task in the same CICS system with which this task is associated.
- ▶ Task number of previous task (field MNR_PTD_TRANNUM)
The task number of the immediately previous or parent task in the same CICS system with which this task is associated.
- ▶ Transaction ID of previous task (field MNR_PTD_TRANID)
The transaction ID of the immediately previous or parent task in the same CICS system with which this task is associated.
- ▶ Previous transaction depth count (field MNR_PTD_COUNT)
The number of times there has been a request from one task to initiate another task in the same CICS system with which this task is associated, such as by an **EXEC CICS RUN TRANSID** or **EXEC CICS START** command when a new point of origin is not created.

Identity class data

To provide the enhanced transaction tracking information described in 8.9.1, “Enhanced transaction tracking information” on page 174, the following fields are added to the identity class data:

- ▶ Previous task attach time (field MNI_PTD_ATTACH_TIME)
The start time of the immediately previous or parent task in the same CICS system with which this task is associated.
- ▶ Task number of previous task (field MNI_PTD_TRANNUM)
The task number of the immediately previous or parent task in the same CICS system with which this task is associated.
- ▶ Transaction ID of previous task (field MNI_PTD_TRANID)
The transaction ID of the immediately previous or parent task in the same CICS system with which this task is associated.
- ▶ Previous transaction depth count (field MNI_PTD_COUNT)
The number of times there has been a request from one task to initiate another task in the same CICS system with which this task is associated, such as by an **EXEC CICS RUN TRANSID** or **EXEC CICS START** command when a new point of origin is not created.

8.9.4 CICS statistics enhancements

A new statistics report is added to provide information about how work is processed by the AS domain and the effect the AS workload management has on parent and child tasks.

The following fields are available in the new asynchronous services global statistics report:

- ▶ RUN command count (field ASG_RUN_COUNT)
The total number of **EXEC CICS RUN TRANSID** API commands issued.
- ▶ FETCH command count (field ASG_FETCH_COUNT)
The total number of **EXEC CICS FETCH CHILD** and **EXEC CICS FETCH ANY** API commands issued.
- ▶ FREE command count (field ASG_FREE_COUNT)
The total number of **EXEC CICS FREE CHILD** API commands issued.
- ▶ Times EXEC CICS RUN command being delayed (field ASG_RUN_DELAY_COUNT)
The total number of times that **EXEC CICS RUN TRANSID** API commands delayed by flow control.
- ▶ Current parents being delayed (field ASG_PARENTS_DELAYED_CUR)
The current number of tasks that are delayed by flow control when issuing an **EXEC CICS RUN TRANSID** API command.
- ▶ Peak parents being delayed (field ASG_PARENTS_DELAYED_PEAK)
The peak number of tasks that were delayed by flow control when issuing an **EXEC CICS RUN TRANSID** API command.
- ▶ Current number of child tasks (field ASG_CHILDREN_CUR)
The current number of active tasks that were started by **EXEC CICS RUN TRANSID** API commands.

- ▶ Peak number of child tasks (field ASG_CHILDREN_PEAK)

The peak number of active tasks that were started by **EXEC CICS RUN TRANSID** API commands.

Example 8-2 shows an extract of a sample DFHSTUP report for a workload that uses the asynchronous API.

Example 8-2 Asynchronous services global statistics report produced by CICS TS V5.4 DFHSTUP

RUN commands	:	40587368
FETCH commands	:	40587290
FREE commands	:	0
Current active children	:	1
Peak active children	:	120
Times RUN commands being delayed	:	823
Current parents being delayed	:	0
Peak parents being delayed	:	5

For more information about asynchronous services domain statistics, see the “Asynchronous services domain: Global statistics” topic in IBM Knowledge Center at this website:

<https://ibm.biz/Bdi6nf>

8.9.5 Asynchronous service domain performance comparison

To understand the overhead of the new API commands, a simple workload was created. This workload compared the new **EXEC CICS RUN TRANSID** command with another common method of executing child tasks in CICS: **EXEC CICS START**.

Two transactions were created, each of which have an initial program that performs the following sequence of CICS commands:

1. EXEC CICS RECEIVE

Receives control data from the terminal.

2. EXEC CICS PUT CONTAINER

Configures a container to be passed to the child task, indicating the length of time for which the child task will suspend.

3. EXEC CICS PUT CONTAINER

Configures a container in the channel to pass dummy data to the child task.

4. EXEC CICS START (transaction ASGS) or **EXEC CICS RUN TRANSID** (transaction ASGX)

Creates a new child task using the selected CICS command. This step is repeated multiple times, based on input received from the terminal.

5. EXEC CICS SEND

Sends a completion message to the terminal.

6. EXEC CICS RETURN

Returns control to CICS.

Steps 2 to 4 are repeated multiple times to create the required number of child tasks. Both programs create the same child transaction, ACHL, passing 4 KB of data as input. Neither transaction performs any other business logic, nor is any logic used to return information from the child tasks to the parent task. The suspend time configured in step 2 is used to simulate

CICS waiting for an external service, without introducing unnecessary dependencies on non-CICS resources.

The transactions were initiated from a terminal using the methodology described in 2.4, “Driving the workload” on page 16. The transactions were executed approximately 270 times with CICS performance class monitoring enabled, and averages of the resulting data points recorded. Table 8-13 shows the CICS performance class monitoring data collected for each of the transactions in the workload.

Table 8-13 CICS monitoring data for the asynchronous API command comparison

Transaction	Average response time (ms)	Average user CPU time (ms)	Average QR CPU time (ms)	Average QR dispatch time (ms)	Average TCB change mode operations
ASGS	0.631	0.184	0.125	0.137	30
ASGX	0.284	0.140	0.030	0.038	10
ACHL	0.163	0.036	0.036	0.037	0

As shown in the table, the average CPU time for the ASGS transaction was 0.184 ms, compared to 0.140 ms for the ASGX transaction. Both the ASGS and ASGX transactions create ten ACHL transactions, with the remainder of the application logic identical. We can therefore conclude that the EXEC CICS START command costs approximately 0.004 ms of CPU more per request than the EXEC CICS RUN TRANSID command.

The “Average TCB change mode operations” column shows an average of 30 change mode operations for the ASGS transaction (EXEC CICS START), with the ASGX transaction (EXEC CICS RUN TRANSID) requiring only 10 change mode operations.

The programs used in the ASGS and ASGX transactions are both defined with the CONCURRENCY attribute of REQUIRED. As described in 4.3.2, “Programs specifying JVM(NO) and API(CICSAPI)” on page 34, specifying the value of REQUIRED will cause the program to always run on an Open TCB.

The ASGX transaction has a baseline of 10 TCB switch operations, which are accumulated as follows:

- ▶ Initial program invocation
 - 1 TCB change mode (QR L8)
- ▶ EXEC CICS RECEIVE (a non-threadsafes command)
 - 2 TCB change modes (L8 QR L8)
- ▶ EXEC CICS PUT CONTAINER (a threadsafes command)
 - 0 TCB change modes
- ▶ EXEC CICS RUN TRANSID (a threadsafes command)
 - 0 TCB change modes
- ▶ EXEC CICS SEND (a non-threadsafes command)
 - 2 TCB change modes (L8 QR L8)
- ▶ End of task sync point and task-related user exit (TRUE) processing
 - 4 TCB change modes
- ▶ Transaction termination
 - 1 TCB change mode (L8 QR)

The ASGS transaction follows a similar pattern as the ASGX baseline, but the **EXEC CICS START** command is not threadsafe. Therefore, invoking the program switches from an L8 TCB to the QR TCB and back for each of the 10 **EXEC CICS START** commands that are executed. The ASGS transaction has 20 TCB change mode operations more than the ASGX baseline. Table 8-13 on page 180 also shows the extra QR TCB dispatch and CPU time when using the **EXEC CICS START** command.

Table 8-13 on page 180 also shows a response time improvement when using **EXEC CICS RUN TRANSID** and this is due primarily to the removal of time spent waiting for dispatch on the QR TCB.

8.9.6 Asynchronous API performance summary

The results in 8.9.5, “Asynchronous service domain performance comparison” on page 179 show that initiating child tasks using the **EXEC CICS RUN TRANSID** command is approximately the same CPU cost as using an **EXEC CICS START** command. The results show a slight overhead for **EXEC CICS START** due to TCB change mode processing, but if the user application were not defined with the **CONCURRENCY** attribute of **REQUIRED**, then this overhead would disappear.

Using the asynchronous API can provide modern applications with a method of initiating child tasks and retrieving their results using an API provided by CICS. Alternative implementations of managing communication between parent and child tasks include:

- ▶ Using event control blocks (ECBs) or the **EXEC CICS ENQ** and **EXEC CICS DEQ** commands to synchronize access to common storage. This can be error-prone and places the responsibility of task management with application code, rather than the system. It is particularly difficult to gracefully handle timeout processing in all cases. Child tasks and common storage can easily become orphaned in the event of application abends.
- ▶ Polling of CICS services (such as temporary storage). Polling a resource to check for a response can have two undesired consequences:
 - It can waste CPU when the polling time is short and the average response time is long
 - It can cause unnecessary delays in response when the polling time is long and the average response time is short
- ▶ Using CICS services (such as intrapartition transient data trigger queues) to notify of child task completion. This has a lower overhead than the polling approach and maintains application responsiveness, but still requires some application management of tasks. Handling timeouts can also be problematic.
- ▶ Using external resource manager (such as IBM MQ). A common application pattern is for a parent task to start a child task and then use an **MQ GET** command to wait for notification of completion. Calling an external resource manager in another address space adds additional overhead in terms of CPU time. Use of external products also adds to application complexity, increasing management and monitoring overhead.

All of these implementation alternatives require either an explicit or an implicit **EXEC CICS START** command to initiate the child task. The performance results discussed previously demonstrate that the **EXEC CICS RUN TRANSID** is no more expensive to execute than a simple **EXEC CICS START** command yet provides access to the full range of advantages of the CICS asynchronous API:

- ▶ A fully-supported API, removing the need for complex infrastructure code in the application.
- ▶ Automated workflow management which allows transactions to complete, even during periods when CICS is under stress.

- ▶ Communication between parent and child tasks is via the channels and containers API, which uses 64-bit virtual storage for data. This reduces pressure on virtual storage in the UDSA and EUDSA dynamic storage areas.
- ▶ Advanced task-level and system-level information provided alongside the existing CICS statistical and performance monitoring data.

The CICS asynchronous API provides a simple, supported method of coordinating parent and child tasks, and remains no more expensive than alternative, application-specific implementations.

8.10 EXCI support for channels and containers

The CICS TS V5.4 release introduces the ability to pass data in channels and containers when invoking CICS applications using the external CICS interface (EXCI). The use of channels and containers provides two main benefits:

- ▶ Multiple, named data areas can be passed in a single call
- ▶ Each container can hold significantly a larger data area than possible with a COMMAREA

Note: Only the client (EXCI) application needs to use CICS TS V5.4 libraries to take advantage of this function. Any release of CICS that supports channels and containers can receive data from a client application using the channels and containers interface.

For more information about EXCI, see the “Introduction to the external CICS interface” topic in IBM Knowledge Center at this website:

<https://ibm.biz/Bdi6nM>

A common use-case for EXCI applications is to send data into a CICS program, but the data to be transmitted is larger than the COMMAREA limit of 32 KB. For these cases, the application *chunks* the data into areas less than 32 KB in size. Multiple requests are used to send the data into CICS, and logic in the CICS program reassembles the data into a contiguous block.

The channels and containers support for EXCI allows applications to send large areas of data into CICS, removing the complexity of this chunking logic and the overhead of multiple calls to the server. This section presents a comparison between two applications, one that uses the chunking method with COMMAREAs, and another that uses channels and containers.

The client EXCI application was implemented using the C language, and the CICS applications were implemented using COBOL.

8.10.1 Application design (COMMAREA)

The COMMAREA test case used the following logic in the client application:

1. Allocate storage and initialize data to be sent to the server. Note that the storage is initialized to random values to remove any potential optimizations achieved with large data areas consisting solely of zeros.
2. Open the EXCI pipe to the server using the **Initialize_User**, **Allocate_Pipe**, and **Open_Pipe** commands.
3. Start the timers for recording elapsed time in the client application.
4. Send the data to the server in chunks:

- a. Initialize the current offset pointer to zero.
 - b. Send a maximum of 32,763 bytes of data using the **DPL_Request** command.
 - c. Advance the current offset pointer.
 - d. Repeat from step b until no data remains.
5. Stop the timers for recording elapsed time in the client application.
 6. Shut down the EXCI pipe using calls to the **Close_Pipe** and **Deallocate_Pipe** commands.

The COMMAREA test case used long-running mirrors in the server to allow the receiving transaction to utilize task-related storage for reassembly of the chunked data. The following logic was used in the server application:

1. For the first chunk of data received:
 - a. Issue **EXEC CICS GETMAIN** to obtain storage for the total data length required, supplied as the first four bytes of the COMMAREA.
 - b. Save the obtained storage address and the current offset pointer in the transaction work area (TWA).
 - c. Copy received chunk of data to the obtained storage.
 - d. Advance current offset pointer.
2. For subsequent chunks of data received:
 - a. Copy received chunk of data to the obtained storage.
 - b. Advance the current offset pointer.
3. For the final chunk of data received:
 - a. Copy received chunk of data to the obtained storage.
 - b. Application would normally perform business logic at this point.
 - c. Issue **EXEC CICS FREEMAIN** to release allocated storage.

In both the client and server application code, to minimize overheads no data was copied unnecessarily. As described in 2.2, “Workload design” on page 12, the application was designed to include as little business logic as possible, to more clearly determine the costs of the CICS infrastructure.

8.10.2 Application design (channels)

The channels test case used the following logic in the client application:

1. Allocate storage and initialize data to be sent to the server. Note that the storage is initialized to random values to remove any potential optimizations achieved with large data areas consisting solely of zeros.
2. Open the EXCI pipe to the server using the **Initialize_User**, **Allocate_Pipe**, and **Open_Pipe** commands.
3. Start the timers for recording elapsed time in the client application.
4. Issue **EXEC CICS PUT CONTAINER** to store the data in a container.
5. Invoke the CICS program using the **DPL_Request** command, passing the channel name specified in the previous step.
6. Issue **EXEC CICS DELETE CONTAINER** to free storage in the client application.
7. Stop the timers for recording elapsed time in the client application.

8. Shut down the EXCI pipe using calls to the **Close_Pipe** and **Deallocate_Pipe** commands.

The channels test case also used long-running mirrors in the server to simplify application configuration, although this was not a functional requirement. The following logic was used in the server application:

1. Issue **EXEC CICS GET CONTAINER(...) SET(...)** to obtain a reference to the received data.
2. Application would normally perform business logic at this point.

The server application logic is considerably simplified when using channels and containers. As described in the COMMAREA test case, no data was copied unnecessarily to minimize overheads. As described in 2.2, “Workload design” on page 12, the application was designed to include as little business logic as possible, to more clearly determine the costs of the CICS infrastructure.

8.10.3 Payload comparisons

The performance benchmark compared response time and CPU consumption for several sizes of payload. Table 8-14 lists the payload sizes used, along with the number of COMMAREAs required to transmit the payload in its entirety.

Table 8-14 List of payload sizes compared

Payload size label	Bytes transmitted	COMMAREAs required
32	32	1
1,024	1,024	1
32,760	32,760	1
32 KB	32,768	2
512 KB	524,288	17
1 MB	1,048,576	33
2 MB	2,097,152	65

8.10.4 CPU time comparison

Client application CPU was measured using the z/OS TIMEUSED service. CPU consumed by the server application was measured using CICS performance class monitoring data. Total CPU measurements include CPU consumed by both the client application and the receiving CICS task.

Table 8-15 lists the total CPU time consumed by the client and server application for both the COMMAREA and channel scenarios.

Table 8-15 CPU time comparison for various payloads

Payload	COMMAREA total CPU time (ms)	Channel total CPU time (ms)
32	0.016	0.057
1,024	0.016	0.057
32,760	0.035	0.079

Payload	COMMAREA total CPU time (ms)	Channel total CPU time (ms)
32 KB	0.053	0.080
512 KB	0.589	0.351
1 MB	1.171	0.655
2 MB	2.323	1.251

The results in Table 8-15 can be plotted on a chart to demonstrate the behavior of the two scenarios as the payload size increases. Figure 8-10 plots the results for payloads of size 32 bytes to 32 KB.

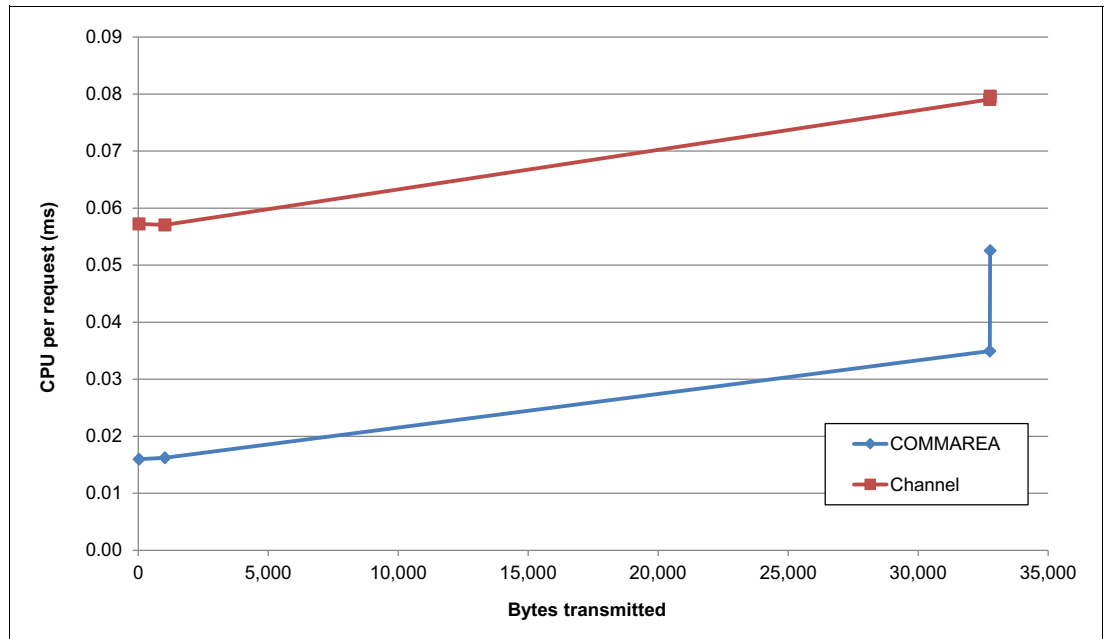


Figure 8-10 Plot of total CPU time comparing COMMAREA and channel scenarios for small payloads

Note the increase in CPU time when increasing the payload from 32,760 bytes to 32,768 bytes in the COMMAREA scenario. This sudden increase is the overhead of requiring a second send request, because the data no longer fits within a single COMMAREA.

Figure 8-11 plots the results for payloads of size 32 KB to 2 MB.

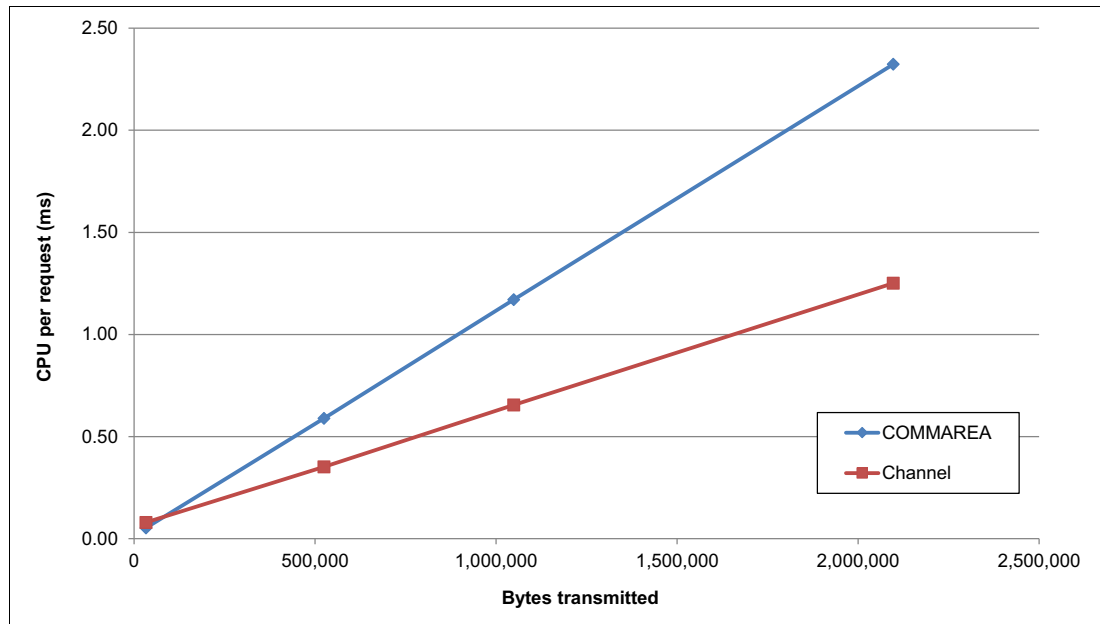


Figure 8-11 Plot of total CPU time comparing COMMAREA and channel scenarios for large payloads

Both scenarios scale linearly as the payload size increases, but the channel implementation is more efficient from a relatively low payload size.

8.10.5 Response time comparison

Overall application response time was measured in the client using the z/Architecture STCK instruction. Table 8-16 lists the application response times for both the COMMAREA and channel scenarios.

Table 8-16 Response time comparison for various payloads

Payload	COMMAREA response time (ms)	Channel response time (ms)
32	0.024	0.072
1,024	0.024	0.072
32,760	0.084	0.141
32 KB	0.108	0.144
512 KB	1.365	1.113
1 MB	2.711	2.154
2 MB	5.390	4.230

The results in Table 8-16 can be plotted on a chart to demonstrate the behavior of the two scenarios as the payload size increases. Figure 8-12 on page 187 plots the response times for payloads of size 32 bytes to 32 KB.

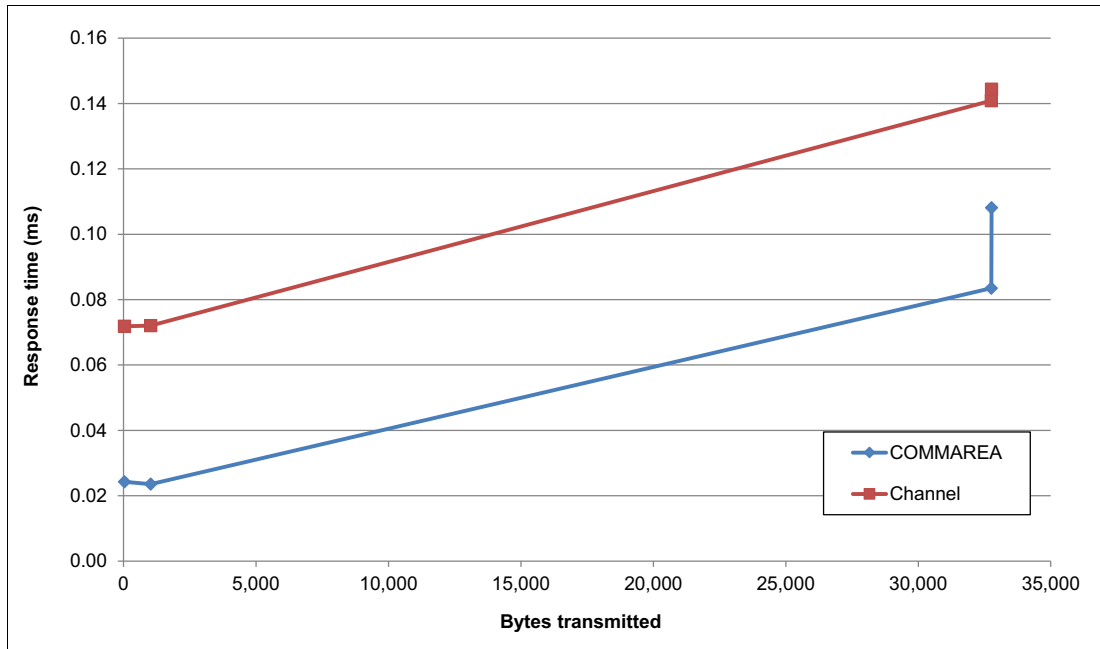


Figure 8-12 Plot of response time comparing COMMAREA and channel scenarios for small payloads

As observed for CPU time in Figure 8-10 on page 185, there is relatively large increase in response time when the payload requires a second send request.

Figure 8-13 plots the response times for payloads of size 32 KB to 2 MB.

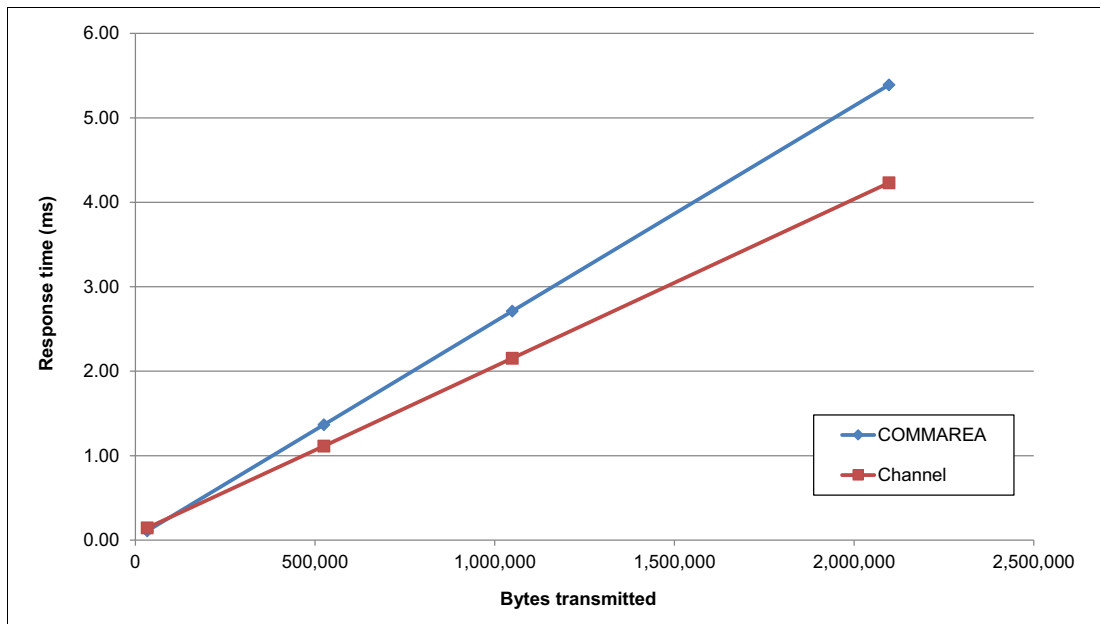


Figure 8-13 Plot of response time comparing COMMAREA and channel scenarios for large payloads

As observed for the CPU time comparisons, both scenarios scale linearly as the payload size increases, but again the channel implementation produces lower response times from a relatively low payload size.

8.10.6 EXCI support for channels and containers performance summary

For smaller payloads, the channels and containers implementation can be shown to have a minor overhead when compared to the COMMAREA implementation in terms of both CPU consumption and response time. As the payload size increases, this CPU and response time overhead is reduced dramatically when the payload requires more than one send operation.

Using linear regression analysis shows that channels and containers are the most efficient approach for payloads larger than 64 KB (that is, those that require three or more `DPL_Request` operations).

As payload sizes increase, CPU and response time increase linearly for both scenarios.

The overhead of channels and containers for this test case was at most 40 μ s of CPU time and 50 μ s in response time, even for the smallest of payloads. This is a small value in the context of a real-world application with business logic. The advantages of the channels and containers implementation over a COMMAREA approach are:

- ▶ Reduced client and server code complexity when sending more than 32 KB of data.
- ▶ Increased flexibility, due to the ability to send multiple data areas in one call.
- ▶ Potential for future expansion without redesigning the application.

The code maintenance advantages listed here are significant, and these offset the small CPU benefits of a COMMAREA solution. Therefore, any new EXCI applications invoking CICS should use the channels and containers interface.

8.11 CICS support for IBM Health Checker for z/OS

IBM Health Checker for z/OS (Health Checker) is a z/OS component that helps simplify and automate the identification of potential configuration problems before they impact availability or cause outages. CICS TS supports Health Checker rules that define preferred practices for CICS system configuration.

Each CICS region providing support for Health Checker executes the system transaction `CHCK` as a long-running task. This task wakes up every 30 minutes to check and report on compliance to preferred practices. To ensure this task does not consume unnecessary CPU, two idle CICS regions were compared.

One CICS region had the Health Checker reporting enabled and the other had the Health Checker reporting disabled. Both CICS regions had security enabled (SIT parameter `SEC=YES`), and 25 transient data queues were installed. To ensure the most accurate reporting of CPU consumption, the `CHCK` transaction was specially modified to wake up every 5 seconds.

Using the overnight automation environment as described in 2.3, “Repeatable measurements” on page 13, performance of the two CICS regions were measured while idle for a 5-minute measurement period. CPU consumption data was extracted from the CICS statistics report and is presented in Table 8-17 on page 189. All data is presented in CPU seconds.

Table 8-17 Summary of CICS statistics for Health Checker CPU measurement

	CHCK disabled	CHCK enabled	Difference
Address space TCB time	0.003888	0.035986	0.032098
Address space SRB time	0.000349	0.000538	0.000189
Total CPU time	0.004237	0.036524	0.032287

The CICS statistics interval was 305 seconds; therefore, we can conclude the CHCK task woke up around 60 times during that time period. Also, we can calculate that the CHCK task cost approximately 540 μ s of CPU per iteration.

When using the CICS support for Health Checker configuration with the standard checking frequency of 30 minutes, this functionality consumes approximately 1 ms of CPU per hour for a completely idle CICS region. In contrast, the results presented in Table 8-17 for the CICS region with Health Checker support disabled show a CPU consumption of approximately 50 ms per hour for other CICS background processing. Thus, we can conclude that CICS support for the IBM Health Checker for z/OS adds no significant overhead to a CICS region.

Note: CICS support for the IBM Health Checker for z/OS is also available for releases prior to CICS TS V5.4. See APAR PI76963 for CICS TS V4.2 and APAR PI76965 for CICS TS V5 releases.

For more information about the CICS support for the Health Checker, see the topic “Checking CICS configuration with IBM Health Checker for z/OS” in IBM Knowledge Center:

<https://ibm.biz/BdjdQk>

8.12 Web services performance

This section examines the performance of the CICS TS V5.4 web services support, using several variants of the web services workload detailed in 3.6, “Web services” on page 28. All sections present the data in a similar tabular format:

- ▶ Request rate
 - The number of SOAP requests per second
- ▶ CICS CPU %
 - The amount of CPU consumed by the CICS provider region, expressed as a percentage of a single CP
- ▶ TCP/IP CPU %
 - The amount of CPU consumed by the TCP/IP address space, expressed as a percentage of a single CP
- ▶ CPU per request
 - A calculation of the total CPU consumed per request, which includes the CICS provider region and the TCP/IP address space, expressed as milliseconds
- ▶ Response time
 - Where applicable, the response time is obtained from RMF monitoring data, expressed in milliseconds

8.12.1 Persistent and non-persistent connections

This section looks at the performance of the web services workload when running with non-persistent and persistent connections. Neither scenario used SSL.

The data in Table 8-18 shows the performance characteristics of CICS web services support when using persistent connections.

Table 8-18 Web services workload with persistent connections and no SSL

Request rate	CICS CPU %	TCP/IP CPU %	CPU per request (ms)
417	12.58%	0.48%	0.313
500	15.14%	0.58%	0.314
555	16.99%	0.69%	0.319
714	22.10%	0.79%	0.321
999	31.20%	0.98%	0.322

Table 8-19 shows the performance characteristics of the same workload when using non-persistent connections.

Table 8-19 Web services workload with non-persistent connections and no SSL

Request rate	CICS CPU %	TCP/IP CPU %	CPU per request (ms)
417	14.66%	0.85%	0.372
500	17.62%	1.03%	0.373
556	19.63%	1.16%	0.374
714	25.29%	1.45%	0.375
1000	36.19%	1.94%	0.381

For both configurations and at all measured transaction rates, the response times were less than 1 ms therefore the response time columns have been omitted.

These results are summarized by the chart in Figure 8-14.

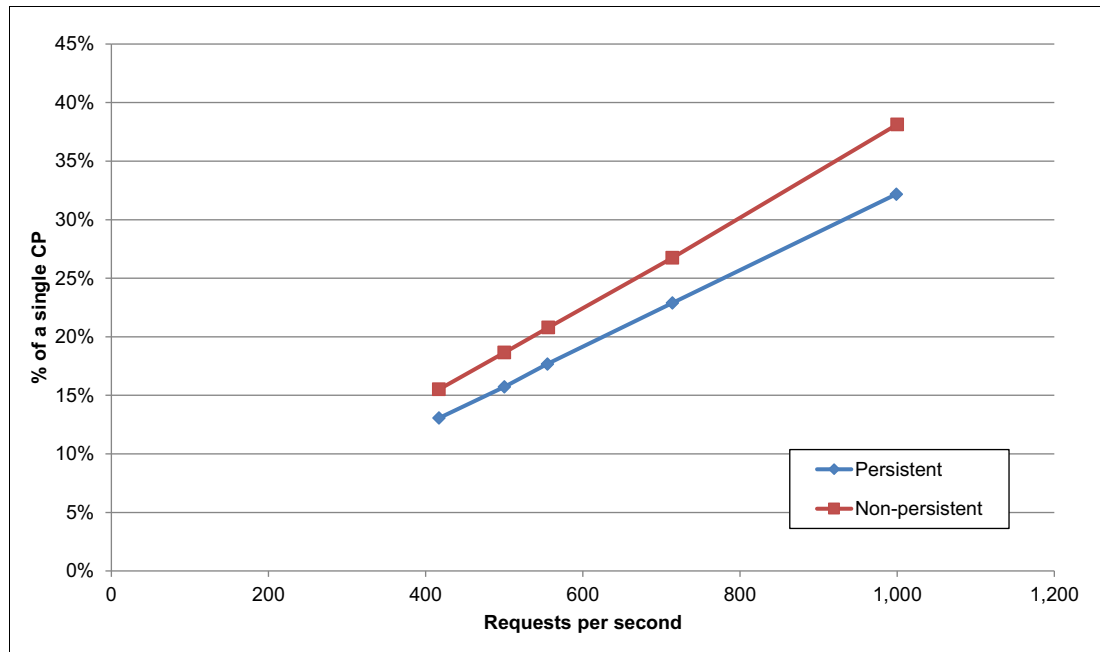


Figure 8-14 Comparison of persistent and non-persistent connections without SSL

On average the CICS provider region and TCP/IP address space consumed 0.318 ms of CPU per request with persistent sessions, and 0.375 ms of CPU per request with non-persistent sessions. This overhead is the result of the small extra processing required by the CICS provider region and network layer when creating and destroying an HTTP session.

8.12.2 Overhead of using SSL with persistent connections

As described in 3.6.1, “Web services variations” on page 29 the workload can use SSL to encrypt request and response data. To determine the overhead of this encryption, CICS SSL support was enabled to use the TLS_RSA_WITH_AES_256_CBC_SHA cipher suite and persistent sessions.

The results of this configuration are presented in Table 8-20 and can be compared with the performance data presented in Table 8-18 on page 190.

Table 8-20 Web services workload using SSL with persistent connections

Request rate	CICS CPU %	TCP/IP CPU %	CPU per request (ms)
417	13.85%	0.38%	0.341
500	16.51%	0.44%	0.339
555	18.05%	0.46%	0.334
714	23.45%	0.57%	0.336
1000	33.95%	0.75%	0.347

In both configurations and at all transaction rates, the response time was 1 ms or less and therefore the response time column has been omitted for clarity. These measurements are summarized by the chart in Figure 8-15 on page 192.

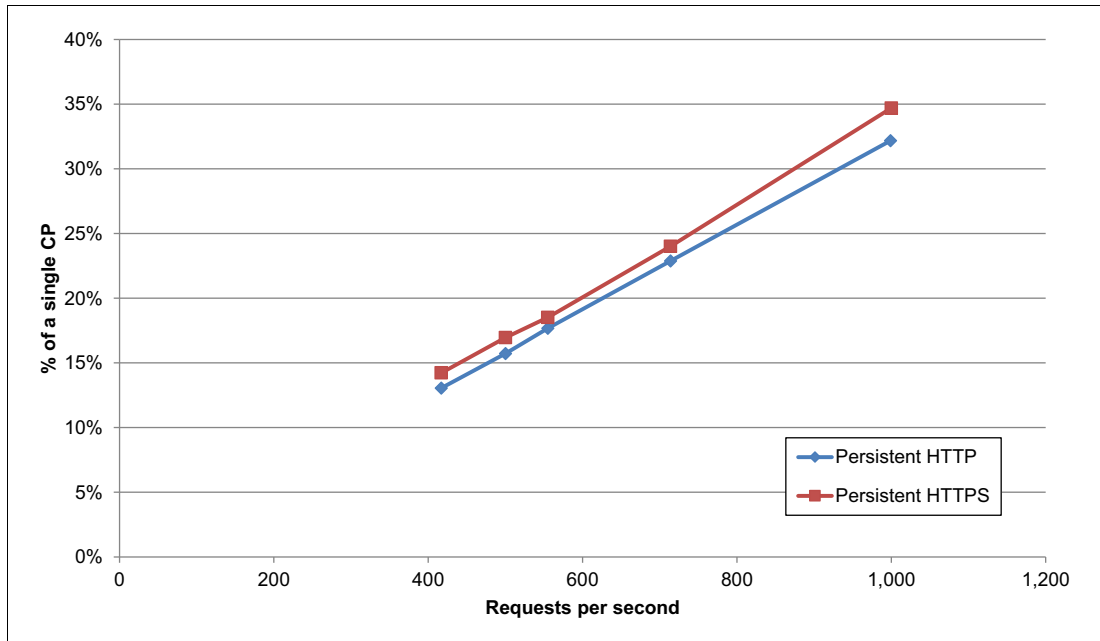


Figure 8-15 Comparison of non-SSL and SSL connections using persistent sessions

As documented in 8.12.1, “Persistent and non-persistent connections” on page 190 the average CPU per request figure for non-SSL communications was 0.318 ms. When using CICS SSL support, this figure rises by 0.057 ms per request to 0.375 ms.

8.12.3 SSL handshake overhead

Section 8.12.2, “Overhead of using SSL with persistent connections” on page 191 documents the overhead of using encryption when no SSL handshakes are required. This section looks in more detail at the handshake options available.

A *full* SSL handshake occurs when SSL partners negotiate an SSL session without the benefit of any previously-agreed information. A *partial* SSL handshake occurs when SSL partners establish an SSL connection using a token cached from a previously-negotiated connection. This partial handshake allows SSL partners to avoid the overhead of a full SSL session negotiation while retaining the integrity of SSL.

The first scenario obtained performance data in a steady-state situation where no SSL handshakes were taking place by enabling persistent sessions. This data was presented in section 8.12.2, “Overhead of using SSL with persistent connections” on page 191 and is available in Table 8-20 on page 191.

The second scenario measured performance when SSL partial handshakes were being used and the performance data is presented in Table 8-21.

Table 8-21 Web services workload using SSL with partial handshakes

Request rate	CICS CPU %	TCP/IP CPU %	CPU per request (ms)	Response time (ms)
417	20.13%	1.12%	0.510	2
500	23.87%	1.33%	0.504	2
556	26.35%	1.43%	0.500	2

Request rate	CICS CPU %	TCP/IP CPU %	CPU per request (ms)	Response time (ms)
715	34.20%	1.75%	0.503	2
1000	48.36%	2.47%	0.508	3

The third scenario measured performance of SSL full handshakes, and the data is presented in Table 8-22.

Table 8-22 Web services workload using SSL with full handshake

Request rate	CICS CPU %	TCP/IP CPU %	CPU per request (ms)	Response time (ms)
416	25.73%	1.62%	0.657	8
500	29.76%	1.92%	0.634	6
555	32.99%	2.11%	0.632	6
713	43.64%	2.57%	0.648	7
986	64.75%	3.12%	0.688	11

The results of these three scenarios are summarized in Figure 8-16.

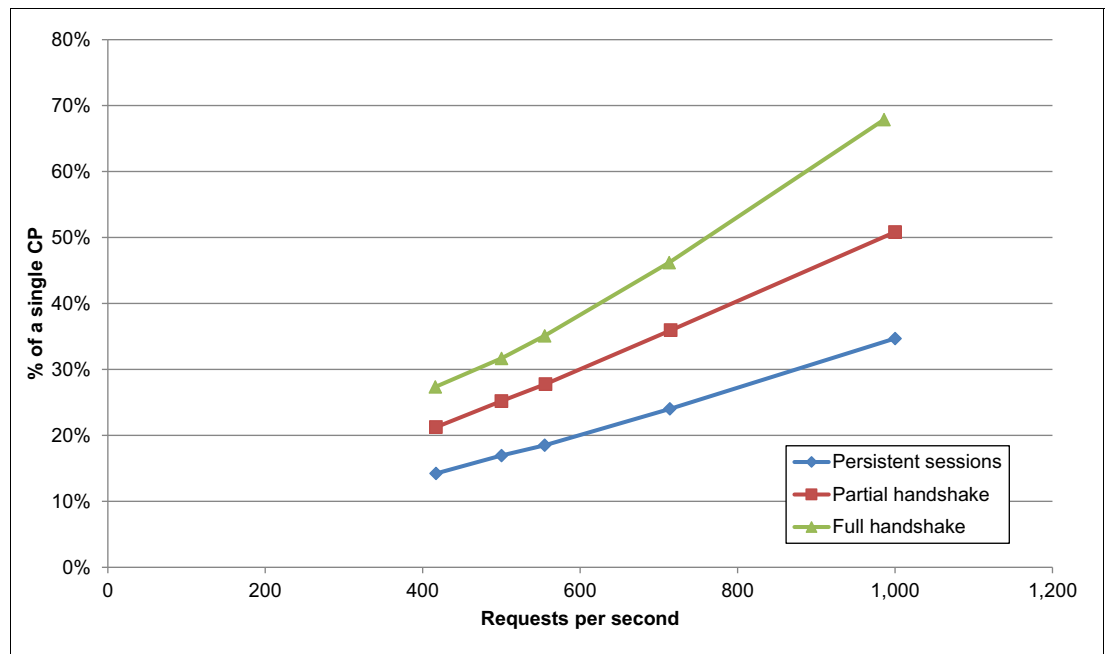


Figure 8-16 Comparison of web services performance using SSL with various handshake options

The average CPU per request for persistent sessions, partial handshakes, and full handshakes is 0.339 ms, 0.505 ms, and 0.652 ms respectively. This data shows that the CPU overhead of a partial handshake is therefore around 0.165 ms of CPU, which includes the cost of establishing a new connection and the renegotiation of a previously-established session. The CPU overhead of a full handshake is approximately 0.150 ms in addition to the cost of a partial handshake.

The CPU overhead for SSL full and partial handshakes is dependant on several factors, including:

- ▶ The key exchange algorithm
- ▶ The size of the and complexity of the public keys
- ▶ Use of client authentication

8.12.4 CICS SSL and AT-TLS

Table 8-23 presents the performance results of the web services workload using AT-TLS with persistent sessions. By comparing the data in Table 8-20 on page 191 with the data in Table 8-23 we can determine the relative difference in performance between using the CICS SSL support and the AT-TLS support provided by IBM Communications Server for z/OS.

Table 8-23 Web services workload using AT-TLS with persistent sessions

Request rate	CICS CPU %	TCP/IP CPU %	CPU per request (ms)	Response time (ms)
417	12.59%	0.39%	0.311	1
500	15.15%	0.46%	0.312	1
556	16.63%	0.50%	0.308	1
714	21.62%	0.64%	0.312	1
1000	30.12%	0.89%	0.310	1

When using the SSL support provided by CICS the average CPU per transaction was 0.339 ms. When using the AT-TLS support the same value was 0.311 ms. This difference in CPU per request can be attributed to two key factors:

- ▶ The removal of the requirement for the CWXN transaction. When using CICS SSL support the CWXN transaction is required, but using AT-TLS this is no longer required. See 7.7, “Web support and web service optimization” on page 122 for more details.
- ▶ A reduction in the number of TCB switches in the user transaction. CICS SSL support requires several task switches to and from S8 TCBs, but AT-TLS does not have this requirement.

The chart in Figure 8-17 on page 195 presents the data from the two scenarios for comparison.

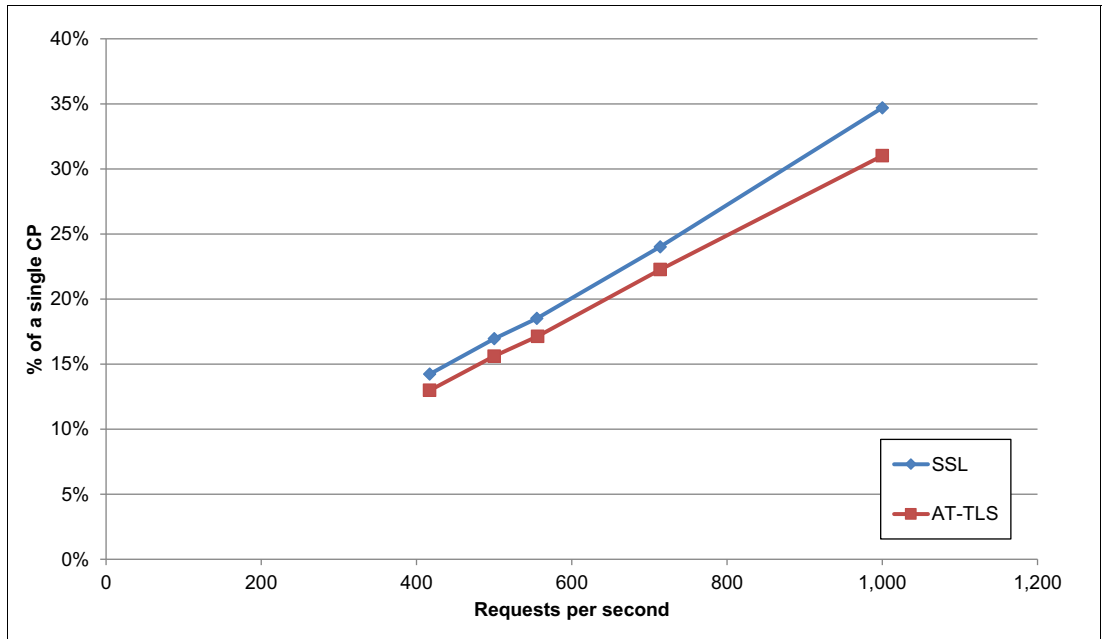


Figure 8-17 Web services workload comparing CICS SSL and AT-TLS with persistent sessions



CICS TS for z/OS V5.5

IBM CICS Transaction Server for z/OS (CICS TS) V5.5 release introduces various technical and operational capabilities. Included in these updates are improvements that provide performance benefits over previous CICS releases. The performance report CICS TS V5.5 includes the following subject areas:

- ▶ Key performance benchmarks that are presented as a comparison with the CICS TS V5.4 release.
- ▶ An outline of improvements made regarding the threadsafe characteristics of the CICS TS run time.
- ▶ Details of the changes that are made to performance-critical CICS initialization parameters, and the effect of these updates.
- ▶ A description of all the updated statistics and monitoring fields.
- ▶ High-level views of new functionality that was introduced in the CICS TS V5.5 release, including performance benchmark results where appropriate.
- ▶ Studies of several areas that have received frequent performance questions from customers, including the use of:
 - CICS policies
 - encrypted zFS file systems with CICS
 - multiple JVM servers in a single CICS region
 - shared class cache to improve JVM server startup performance

This chapter includes the following topics:

- ▶ 9.1, “Introduction” on page 198
- ▶ 9.2, “Release-to-release comparisons” on page 198
- ▶ 9.3, “Improvements in threadsafety” on page 206
- ▶ 9.4, “Changes to system initialization parameters” on page 207
- ▶ 9.5, “Enhanced instrumentation” on page 208
- ▶ 9.6, “Virtual storage constraint relief” on page 210
- ▶ 9.7, “z/OS WLM Health API” on page 211
- ▶ 9.8, “Disabling of VSAM dynamic buffer addition” on page 212
- ▶ 9.9, “USS processes associated with L8, L9, X8, and X9 TCBs” on page 212
- ▶ 9.10, “Channels performance improvement” on page 212

- ▶ 9.11, “Threadsafe Coupling Facility Data Tables” on page 217
- ▶ 9.12, “CICS policy rules” on page 221
- ▶ 9.13, “Encrypted zFS file systems” on page 224
- ▶ 9.14, “Multiple Liberty JVM servers in a single CICS region” on page 226
- ▶ 9.15, “Liberty JVM server and application startup times” on page 233

9.1 Introduction

When we compiled the results for this chapter, the workloads were executed on an IBM z14™ model M04 (machine type 3906). A maximum of 32 dedicated CPUs were available on the measured LPAR, with a maximum of six dedicated CPUs available to the LPAR used to simulate users. These LPARs are configured as part of a parallel sysplex. An internal coupling facility was co-located on the same central processor complex (CPC) as the measurement and driving LPARs, connected by using internal coupling peer (ICP) links. An IBM System Storage DS8870 (machine type 2424) was used to provide external storage.

This chapter presents the results of several performance benchmarks when executed in a CICS TS for z/OS V5.5 environment. Unless otherwise stated in the results, the CICS TS V5.5 environment was the code available at GA time. Several of the performance benchmarks are presented in the context of a comparison against CICS TS V5.4. The CICS TS V5.4 environment contained all PTFs issued before 20 November 2017. All LPARs used z/OS V2.3.

For a definition of performance terms used in this chapter, see Chapter 1, “Performance terminology” on page 3. A description of the test methodology that is used can be found in Chapter 2, “Test methodology” on page 11. For a full description of the workloads used, see Chapter 3, “Workload descriptions” on page 21.

Where reference is made to an *LSPR processor equivalent*, the indicated machine type and model can be found in the large systems performance reference (LSPR) document. For more information about obtaining and using LSPR data, see 1.3, “Large Systems Performance Reference” on page 6.

9.2 Release-to-release comparisons

This section describes some of the results from a selection of regression workloads that are used to benchmark development releases of CICS TS. For more information about the use of regression workloads, see Chapter 3, “Workload descriptions” on page 21.

9.2.1 Data Systems Workload static routing

The static routing variant of the Data Systems Workload (DSW) is described in 9.2.1, “Data Systems Workload static routing”. This section presents the performance figures that were obtained by running this workload. Table 9-1 lists the results of the DSW static routing workload that used the CICS TS V5.4 release.

Table 9-1 CICS TS V5.4 results for DSW static routing workload

ETR	CICS CPU	CPU per transaction (ms)
4180.81	76.37%	0.183
4938.15	89.21%	0.181

ETR	CICS CPU	CPU per transaction (ms)
6054.32	108.09%	0.179
6582.22	116.78%	0.177
7143.83	126.76%	0.177

Table 9-2 on page 199 lists the same figures for the CICS TS V5.5 release.

Table 9-2 CICS TS V5.5 results for DSW static routing workload

ETR	CICS CPU	CPU per transaction (ms)
4180.34	73.22%	0.175
4948.82	85.95%	0.174
6057.17	103.90%	0.172
6591.39	112.31%	0.170
7151.20	121.70%	0.170

The average CPU per transaction value for CICS TS V5.4 is calculated to be 0.179 ms. The same value for CICS TS V5.5 is calculated to be 0.172 ms. This is a relative performance improvement of 4% for this workload. However, this is only an absolute improvement of around 7 μ s. Such a small change in CPU consumption is unlikely to be measurable outside of lab conditions. However, we can conclude that the performance between CICS TS V5.4 and CICS TS V5.5 is not degraded for this workload.

The figures from Table 9-1 and Table 9-2 are plotted in the chart in Figure 9-1.

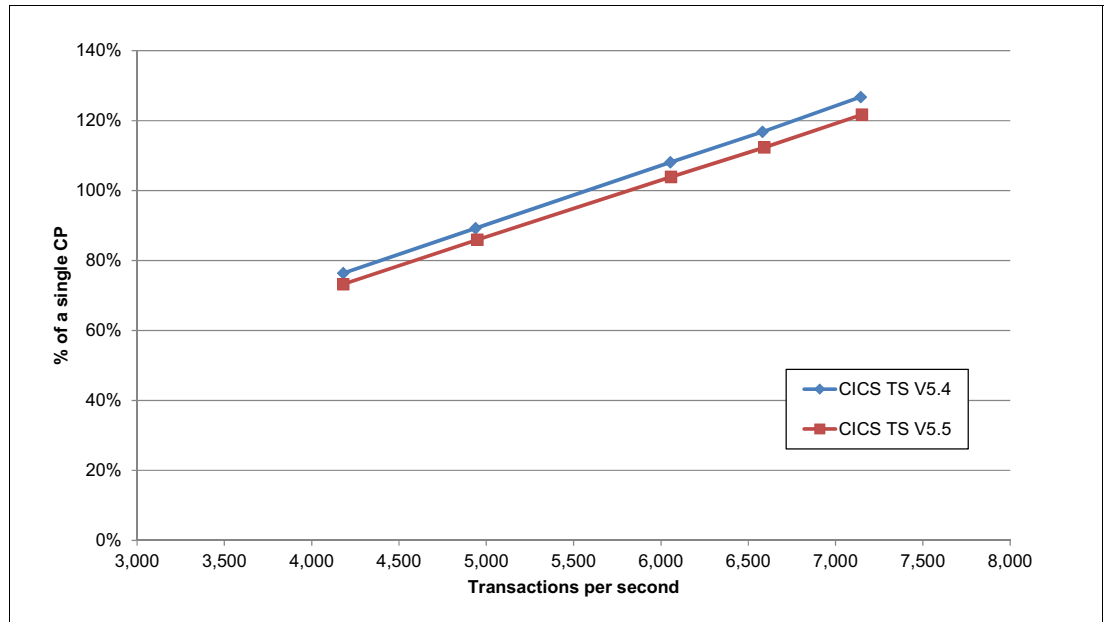


Figure 9-1 Plot of CICS TS V5.4 and V5.5 performance figures for DSW static routing workload

The measured CPU cost for each transaction rate scales linearly in both cases, with CICS TS V5.5 showing a slight improvement as described above.

9.2.2 Java servlet that uses JDBC and VSAM

The Java servlet application is hosted in a CICS JVM server that uses the embedded WebSphere Liberty server. The workload is driven through HTTP requests by using IBM Workload Simulator for z/OS, as described in 2.4, “Driving the workload” on page 16. The servlet application accesses VSAM data by using the JCICS API and accesses IBM Db2® by using the JDBC API. For more information about the workload, see 3.4, “WebSphere Liberty servlet with JDBC and JCICS access” on page 26.

The hardware that is used for the benchmarks is described in 9.1, “Introduction” on page 198. The measurement LPAR was configured with three GCPs and one zIIP, which resulted in an LSPR equivalent processor of 3906-704.

The CICS TS V5.4 and CICS TS V5.5 releases were compared by using the software levels as described in 9.1, “Introduction” on page 198. The CICS TS V5.4 configuration also had the PTF applied for APAR PI99650 to ensure that both configurations used WebSphere Liberty V18.0.0.2. Both configurations used a single CICS region and the following additional software levels and configuration options:

- ▶ IBM Db2 V12
- ▶ Java 8 SR5 FP25 (64-bit)
- ▶ Single JVMSERVER resource with THREADLIMIT=256

Both JVM servers used the following JVM options:

- ▶ -Xgcpolicy:gencon
- ▶ -Xcompressedrefs
- ▶ -XXnosuballoc32bitmem
- ▶ -Xmx200M
- ▶ -Xms200M
- ▶ -Xmnx60M
- ▶ -Xmns60M
- ▶ -Xmox140M
- ▶ -Xmos140M

As described in 2.3.1, “Repeatability for Java workloads” on page 14, this workload requires a warm-up period of 20 minutes. After this warm-up phase completed, the request injection rate was increased every 10 minutes. CPU usage data was collected by using IBM z/OS Resource Measurement Facility (RMF). An average CPU per request value was calculated by using the last 5 minutes of each 10-minute interval.

Table 9-3 lists the performance results of the Java servlet workload that used the CICS TS V5.4 release. This data is presented in the same format as described in 8.2.5, “The Java servlet that uses JDBC and VSAM” on page 156.

Table 9-3 CICS TS V5.4 results for WebSphere Liberty JDBC and VSAM workload

ETR	CICS CPU not zIIP-eligible	CICS CPU zIIP-eligible	CICS CPU total
837.63	20.56%	27.55%	48.11%
1664.88	42.38%	55.29%	97.67%
3002.68	85.17%	94.56%	179.73%

Table 9-4 lists the performance results of the JDBC and VSAM workload that used the CICS TS V5.5 release, presented in the same format as Table 9-3.

Table 9-4 CICS TS V5.5 results for WebSphere Liberty JDBC and VSAM workload

ETR	CICS CPU not zIIP-eligible	CICS CPU zIIP-eligible	CICS CPU total
838.90	20.95%	28.66%	49.61%
1662.18	42.82%	55.07%	97.89%
2980.13	85.66%	95.07%	180.73%

The CICS CPU total values from Table 9-3 and Table 9-4 are plotted in Figure 9-2 on page 201.

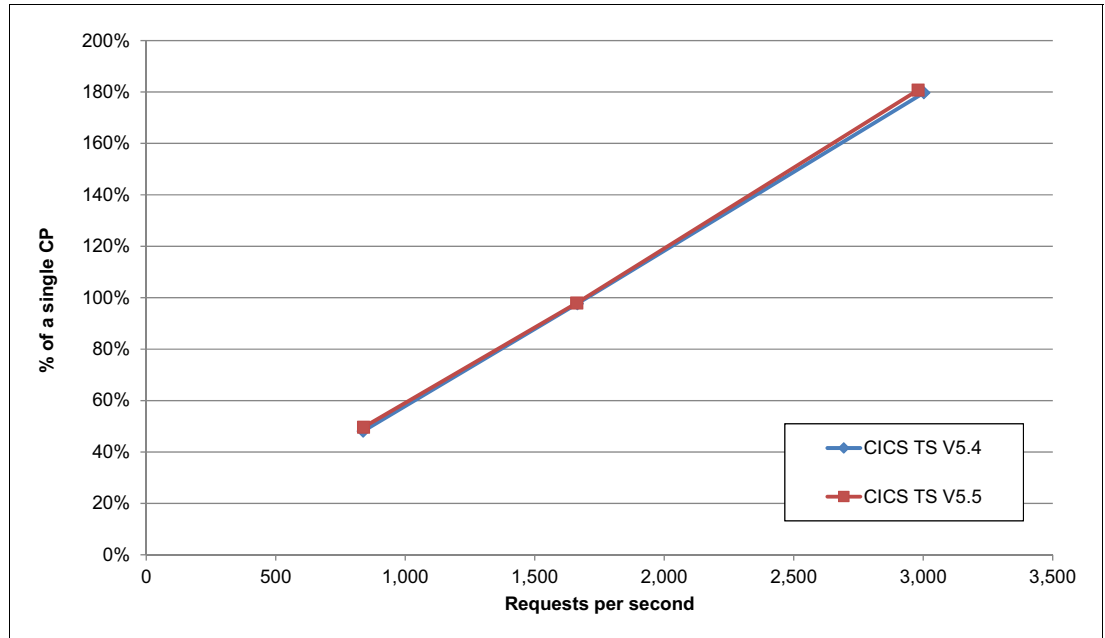


Figure 9-2 Total CPU comparison for CICS TS V5.4 and V5.5 JDBC and VSAM workload

The zIIP-eligibility figures are presented as a chart in Figure 9-3.

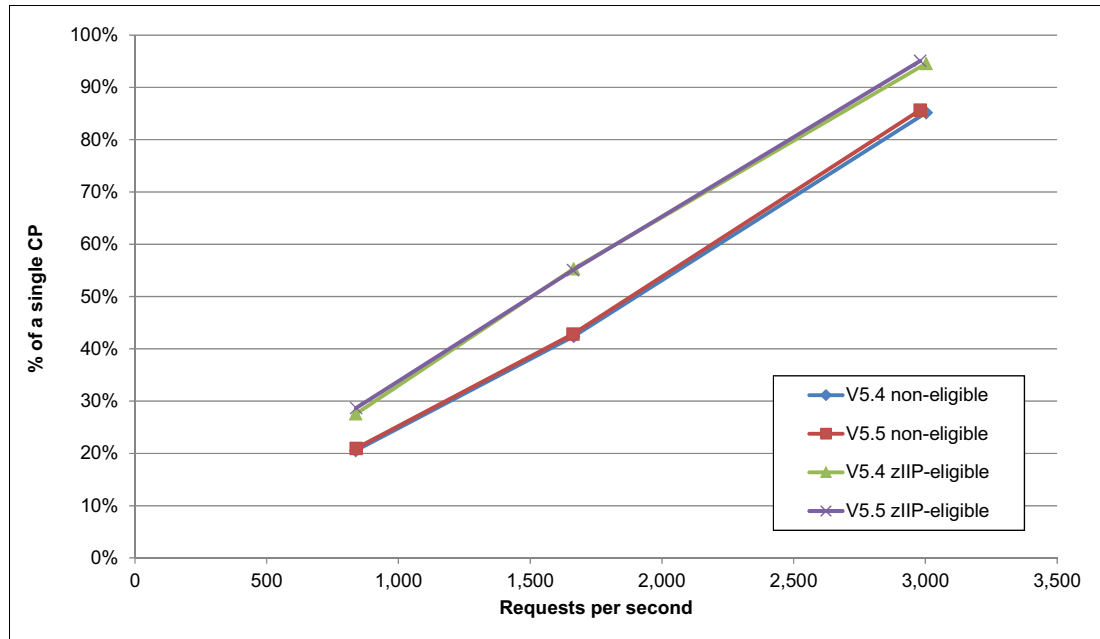


Figure 9-3 zIIP-eligibility comparison for JDBC and VSAM workload with CICS TS V5.4 and V5.5

The average CPU per transaction value for the JDBC and VSAM workload using the CICS TS V5.4 release is calculated to be 0.587 ms. The same value for the CICS TS V5.5 configuration is calculated to be 0.596 ms. The method used to calculate of zIIP-eligibility is described in 8.2.5, “The Java servlet that uses JDBC and VSAM” on page 156. The average zIIP-eligibility for both the CICS TS V5.4 and CICS TS V5.5 workloads was 55.5%.

The average CPU per transaction and the zIIP eligibility calculations show that the performance for the CICS TS V5.4 and CICS TS V5.5 releases is equivalent within measurable limits. This is true for both the total CPU consumed and the fraction that is eligible for offload to a zIIP engine.

9.2.3 The Java OSGi workload

The Java OSGi workload is composed of several applications, as described in 3.5, “Java OSGi workload” on page 27.

The hardware that is used for the benchmarks is described in 9.1, “Introduction” on page 198. The measurement LPAR was configured with three GCPs and one zIIP, which resulted in an LSPR equivalent processor of 3906-704.

The CICS TS V5.4 and CICS TS V5.5 releases were compared by using the software levels as described in 9.1, “Introduction” on page 198. Both configurations used a single CICS region and the following additional software levels and configuration options:

- ▶ IBM Db2 V12
- ▶ Java 8 SR5 FP25 (64-bit)
- ▶ Single JVMSEVER resource with THREADLIMIT=25

Both JVM servers used the following JVM options:

- ▶ -Xgcpolicy:gencon
- ▶ -Xcompressedrefs
- ▶ -XXnosuballc32bitmem

- ▶ -Xmx100M
- ▶ -Xms100M
- ▶ -Xmnx70M
- ▶ -Xmns70M
- ▶ -Xmox30M
- ▶ -Xmos30M

As described in 2.3.1, “Repeatability for Java workloads” on page 14, this workload requires a warm-up period of 20 minutes. After this warm-up phase completed, the request injection rate was increased every 5 minutes. CPU usage data was collected by using IBM z/OS Resource Measurement Facility (RMF). An average CPU per request value was calculated using the last minute of each 5-minute interval.

Table 9-5 lists the performance results of the Java OSGi workload that used the CICS TS V5.4 release.

Table 9-5 CICS TS V5.4 performance results for OSGi workload

ETR	CICS CPU not zIIP-eligible	CICS CPU zIIP-eligible	CICS CPU total
233.98	20.07%	69.89%	89.96%
467.98	39.29%	139.95%	179.24%
831.07	74.65%	249.21%	323.86%

The performance results for the CICS TS V5.5 release are shown in Table 9-6 on page 203.

Table 9-6 CICS TS V5.5 performance results for OSGi workload

ETR	CICS CPU not zIIP-eligible	CICS CPU zIIP-eligible	CICS CPU total
233.98	20.42%	70.22%	90.64%
467.93	39.61%	140.40%	180.01%
822.57	75.10%	247.61%	322.71%

The CICS CPU total values from Table 9-5 and Table 9-6 are plotted in Figure 9-4 on page 204.

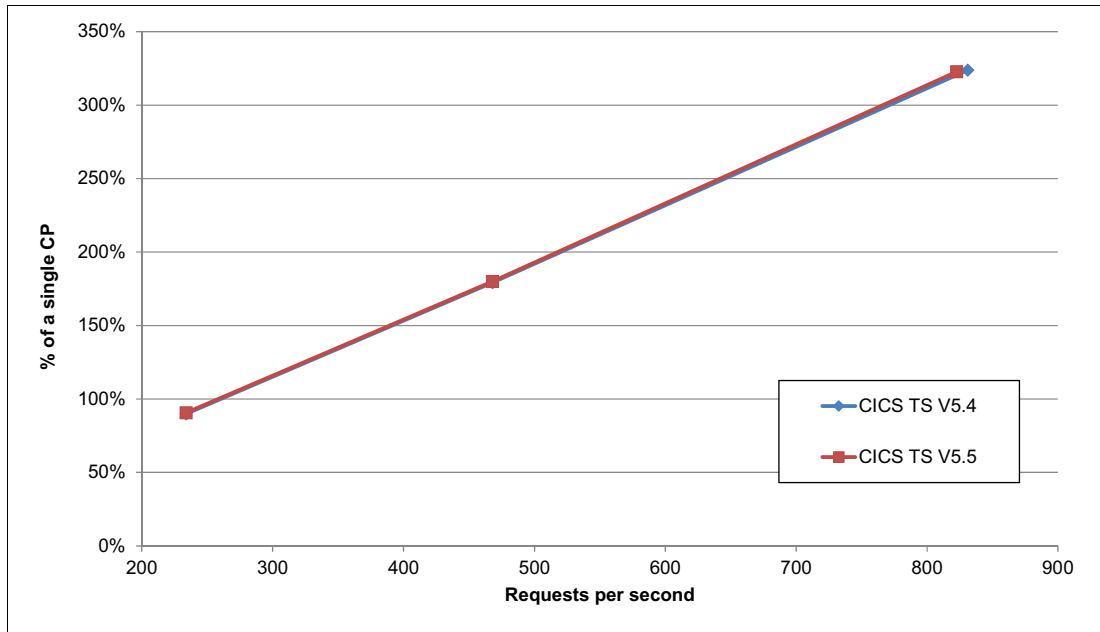


Figure 9-4 Comparing overall CPU utilization for Java OSGi workload with CICS TS V5.4 and V5.5

The offload eligibility figures are presented as a chart in Figure 9-5 on page 204.

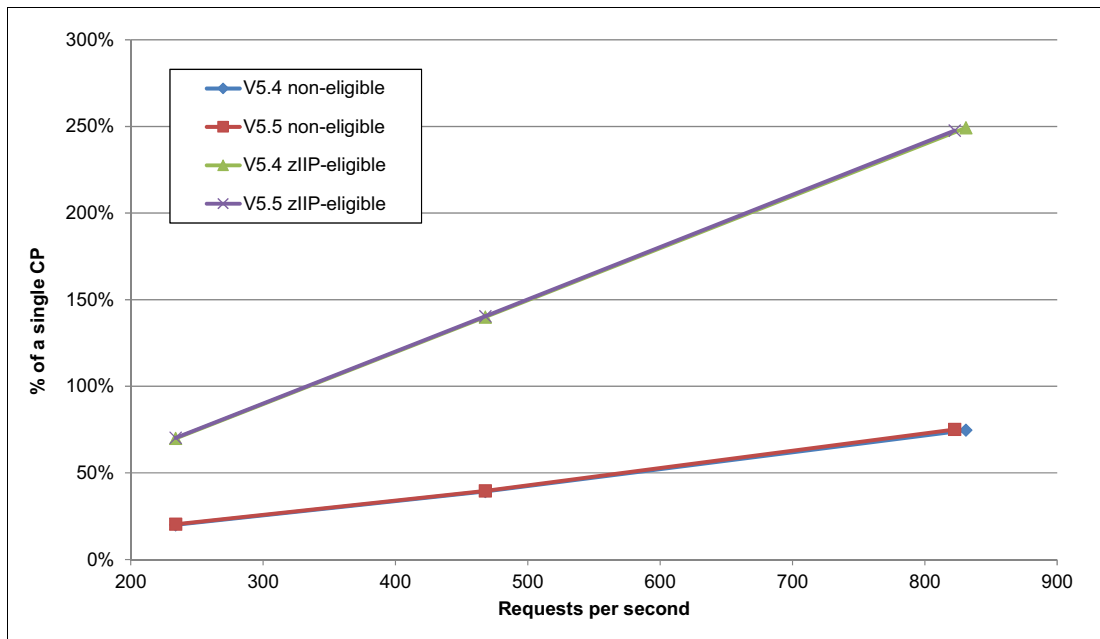


Figure 9-5 Comparing offload-eligible CPU utilization for OSGi workload with CICS TS V5.4 and V5.5

The average CPU per transaction value for this workload using the CICS TS V5.4 release is calculated to be 3.857 ms. The same value for the CICS TS V5.5 release is calculated to be 3.881 ms.

Using the methodology to calculate the zIIP eligibility of the workload described in Chapter 8, the CICS TS V5.4 release had an average zIIP eligibility of 77.6%. See 8.2.5, “The Java servlet that uses JDBC and VSAM” on page 156. The CICS TS V5.5 release had an average zIIP eligibility of 77.4%.

As observed with the Java servlet workload, the performance of Java OSGi applications is similar in CICS TS V5.5 when compared to CICS TS V5.4. This similarity includes both total CPU consumed and the fraction that is eligible for offload to a zIIP engine.

9.2.4 Relational Transactional Workload threadsafe

The Relational Transactional Workload (RTW) is described in 3.3, “Relational Transactional Workload” on page 25. This section presents the performance figures that were obtained by running this workload.

Table 9-7 on page 205 lists the performance results for the RTW threadsafe workload that used the CICS TS V5.4 release.

Table 9-7 CICS TS V5.4 results for the RTW threadsafe workload

ETR	CICS CPU	CPU per transaction (ms)
713.33	89.25%	1.251
996.88	124.43%	1.248
1417.03	177.47%	1.252
1959.66	248.73%	1.269
2401.43	309.99%	1.291

Table 9-4 lists the performance results for the RTW threadsafe workload that used the CICS TS V5.5 release.

Table 9-8 CICS TS V5.5 results for the RTW threadsafe workload

ETR	CICS CPU	CPU per transaction (ms)
713.41	88.59%	1.242
997.00	123.74%	1.241
1417.54	176.81%	1.247
1960.32	248.39%	1.267
2402.72	309.49%	1.288

The figures from Table 9-7 and Table 9-8 are shown in Figure 9-6.

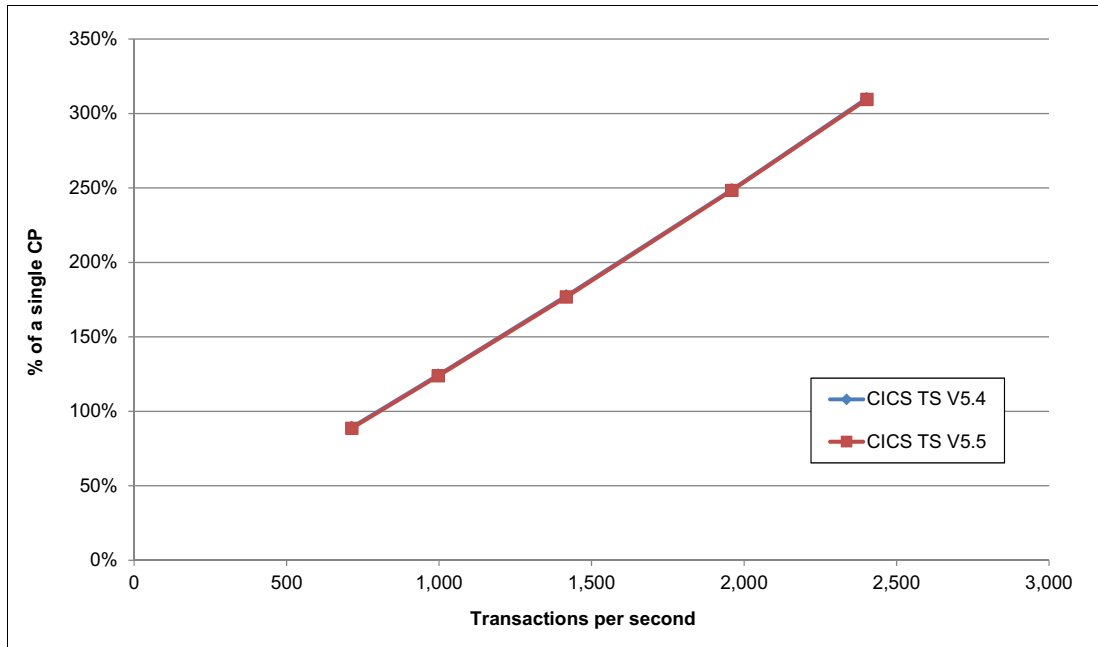


Figure 9-6 Plot of CICS TS V5.4 and V5.5 performance figures for RTW threadsafe workload

The average CPU per transaction value for CICS TS V5.4 is calculated to be 1.262 ms. The same value for CICS TS V5.5 is calculated to be 1.257 ms. Notice that this very small absolute performance improvement is in line with that observed with the DSW static routing workload in 9.2.1, “Data Systems Workload static routing” on page 198. The overall effect to the total CPU cost of the workload is negligible. And we can conclude that the performance between CICS TS V5.4 and CICS TS V5.5 has not degraded for this workload.

9.3 Improvements in threadsafety

There are three areas in CICS TS V5.5 that have improved performance by reducing the number of TCB switches required for API commands.

9.3.1 QUERY SECURITY API command

The **EXEC CICS QUERY SECURITY** command has been enhanced such that the number of TCB switches has been reduced if more than one access level is specified on the command.

For more information on the **EXEC CICS QUERY SECURITY** command, see the “**QUERY SECURITY**” topic in IBM Knowledge Center at this website:

<https://ibm.biz/BdzyuQ>

9.3.2 Coupling Facility Data Tables

Access to coupling facility data tables (CFDTs) is now threadsafe, so CFDTs can be accessed by applications that are running on open TCBs without incurring a TCB switch. Syncpoint processing of CFDTs can also run on an open TCB. However, note that the open and loading of a CFDT still occurs on the QR TCB. See 9.11, “Threadsafe Coupling Facility Data Tables” on page 217 for performance study relating to CFDTs.

For more information on CFDTs, see the “Using coupling facility data tables” topic in IBM Knowledge Center at this website:

<https://ibm.biz/BdzMRb>

9.3.3 System subtasking and auxiliary temporary storage

The SUBTSKS SIT parameter controls the use of the CO TCB when performing I/O. APAR PH05298 was released after the general availability of CICS TS V5.5. It removes the switch to the CO TCB if the application is executing on an open TCB when it uses CICS auxiliary temporary storage. If the application is currently executing on the QR TCB, then subtasking is performed as normal.

This removal of TCB switches provides a small performance benefit for applications the execute on an open TCB. The threadsafe characteristics of the relevant API commands are unaffected.

Note: APAR PH05298 also provides this optimization to all CICS V5 releases.

For more information on the SUBTSKS SIT parameter, see the “SUBTSKS” topic in IBM Knowledge Center at this website:

<https://ibm.biz/Bdzyuu>

9.4 Changes to system initialization parameters

Two CICS system initialization table (SIT) parameters that might have a performance impact have been modified in CICS TS V5.5.

For a detailed view of changes to SIT parameters in the CICS TS V5.5 release, see the “Changes to SIT parameters” section of the “Changes to externals in this release” topic in IBM Knowledge Center at this website:

<https://ibm.biz/BdzyrR>

9.4.1 High Performance Option (HPO)

The HPO parameter can now be specified in the PARM parameter on an EXEC PGM=DFHSIP statement or in the SYSIN data set.

For more information, see the “HPO” topic in IBM Knowledge Center at this website:

<https://ibm.biz/Bdzyrs>

9.4.2 Minimum TLS level (MINTLSLEVEL)

The default value for the MINTLSLEVEL parameter has changed from TLS10 to TLS12.

For more information, see the “MINTLSLEVEL” topic in IBM Knowledge Center at this website:

<https://ibm.biz/Bdzyrf>

9.5 Enhanced instrumentation

The CICS TS V5.5 release continues the expansion of information that is reported by the CICS monitoring and statistics component. This section describes the extra data that is now available in the CICS statistics and monitoring SMF records.

9.5.1 The DFH SOCK performance group

The following new field was added to the DFH SOCK performance group:

- ▶ **New connection indicator** (field SOCONMSG)

Indicates whether the task processed the first message for establishing a new connection for a client.

For more information about counters that are available in the DFH SOCK performance group, see the “Performance data in group DFH SOCK” topic in IBM Knowledge Center at the following website:

<https://ibm.biz/BdzyLf>

9.5.2 The DFH WEBB performance group

The following fields were added to the DFH WEBB performance group:

- ▶ **WEB OPEN URIMAP** request elapsed time (field WBURIOPN)

The total elapsed time that the user task was processing **WEB OPEN URIMAP** requests that are issued by the user task.

- ▶ **WEB RECEIVE** and **WEB CONVERSE** receive portion elapsed time (field WBURIRCV)

The total elapsed time that the user task was processing **WEB RECEIVE** requests and the receiving side of **WEB CONVERSE** requests that are issued by the user task. The sessions these requests target to are opened by the **WEB OPEN URIMAP** command.

- ▶ **WEB SEND** and **WEB CONVERSE** send portion elapsed time (field WBURISND)

The total elapsed time that the user task was processing **WEB SEND** requests and the sending side of **WEB CONVERSE** requests that are issued by the user task. The sessions these requests target to are opened by the **WEB OPEN URIMAP** command.

- ▶ **Node.js application name** (field NJSAPPNM)

Node.js application name from which the task was started.

For more information about counters that are available in the DFH WEBB performance group, see the “Performance data in group DFH WEBB” topic in IBM Knowledge Center at this website:

<https://ibm.biz/BdzyLq>

9.5.3 The DFH WEBC performance group

A new performance group has been created with the following field:

- ▶ **INVOKE SERVICE** request elapsed time (field WBSVINVK)

The total elapsed time that the user task was processing **INVOKE SERVICE** requests for WEBSERVICES.

For more information about counters that are available in the DFHWEBC performance group, see the “Performance data in group DFHWEBC” topic in IBM Knowledge Center at this website:

<https://ibm.biz/BdzyLz>

9.5.4 ISC/IRC system entry resource statistics

The following new field was added to the collected ISC/IRC system entry resource statistics:

- ▶ Peak aids in chain (field A14EAHWM)

The peak number of automatic initiate descriptors (AID) that were present in the AID chain at any one time.

A fragment of a sample DFHSTUP report that shows the new statistics field is shown in Example 9-1.

Example 9-1 Fragment of ISC/IRC system entry resource statistics report produced by CICS TS V5.5 DFHSTUP

Connection name.	:	FOR
Connection netname	:	IYCUZC27
Access Method / Protocol	:	XM /
Autoinstalled Connection Create Time	:	
Send session count	:	400
Aids in chain.	:	0
Peak aids in chain	:	74
ATIs satisfied by contention losers.	:	0
Current contention losers.	:	0

For more information about ISC/IRC system entry statistics, see the topic “ISC/IRC system and mode entry statistics” in IBM Knowledge Center at this website:

<https://ibm.biz/BdzyZJ>

9.5.5 Policy statistics

Statistics are now available for CICS policy rules. CICS collects resource statistics for each rule that is defined in a policy, and supplies a summary report.

Example 9-2 shows a sample DFHSTUP report for an installed policy.

Example 9-2 Extract of sample policy statistics report produced by CICS TS V5.5 DFHSTUP

Policy name.	:	file_v51
Policy user tag.	:	
Bundle name.	:	PLCY51FC
Bundle directory	:	/u/iburnet/git/cics-perf-workload-dsw-lsr/bu : ndles/com.ibm.cics.perf.workload.dsw.lsr.pol : icy.V51.file/
Rule name.	:	READ
Rule type.	:	filerequest
Rule subtype	:	read
Action type.	:	abend
Action count	:	0
Action time.	:	

For more information about CICS policy statistics, see the “Policy statistics” topic in IBM Knowledge Center at this website:

<https://ibm.biz/BdzyLP>

9.5.6 Transaction resource statistics

The following field was added to the collected transaction resource statistics:

- ▶ Abend count (field XMRAENDC)

The number of times that this transaction has abended.

For more information about transaction resource statistics, see the topic “Transaction statistics” in IBM Knowledge Center at this website:

<https://ibm.biz/BdzyMq>

9.5.7 Transaction resource class data

CICS monitoring is enhanced with new monitoring records URIMAP and WEBSERVICE in the resource monitoring class. Multiple URIMAP or WEBSERVICE records can be monitored for one task.

The following fields are now available for each URIMAP entry in a transaction resource monitoring record:

- ▶ MNR_URIMAP_NAME
- ▶ MNR_URIMAP_CIPHER
- ▶ MNR_URIMAP_WEBOPEN
- ▶ MNR_URIMAP_WEBRECV
- ▶ MNR_URIMAP_WESEND

The following fields are now available for each WEBSERVICE entry in a transaction resource monitoring record:

- ▶ MNR_WEBSVC_NAME
- ▶ MNR_WEBSVC_PIPE
- ▶ MNR_WEBSVC_INVK

For more information about fields that are available in the transaction resource class data, see the “Transaction resource class data: Listing of data fields” topic in IBM Knowledge Center at this website:

<https://ibm.biz/BdzyLM>

9.6 Virtual storage constraint relief

The Web domain (WB) now uses internal 64-bit buffer storage when it sends and receives HTTP outbound messages. This change relieves constraint on 31-bit virtual storage and enables more 31-bit application use in a CICS region.

Minor improvements in 24-bit storage usage were also introduced in CICS TS V5.5. The amount of 24-bit storage that is used by the CICS auxiliary trace mechanism was reduced. This changes provided a small performance improvement for both the DSW static routing and the RTW single region workloads when running with auxiliary trace enabled.

9.7 z/OS WLM Health API

As described in 8.8, “z/OS WLM Health API” on page 173, CICS TS V5.4 uses the z/OS Workload Manager (WLM) Health API as a means of controlling the flow of work into a CICS region. This awareness of the z/OS WLM health value has been enhanced in CICS TS V5.5.

For more information about how CICS TS V5.5 uses the z/OS WLM Health API to control the flow of work into a CICS region, see the following article in the CICS Developer Center:

<https://developer.ibm.com/cics/2017/05/16/controlling-flow-work-cics/>

9.7.1 CICSplex SM workload routing

The z/OS WLM health value of a region is now a more effective factor in CICSplex SM workload routing decisions. When it determines the target region to route workload to, CICSplex SM workload management assigns penalizing weights in the routing algorithm based on the actual health value of each region. The higher the health value, the lower the penalizing weight that is assigned, so a region with a greater health value becomes more favorable as a target. In addition, a region with a health value of zero is now deemed as ineligible to receive work.

With this enhancement to CICSplex SM workload routing, you can have better control of flow of work into regions that are in warm-up or cool-down.

Note: The refined use of z/OS WLM Health when making routing decisions was also made available in CICS TS V5.4 with APAR PI90147.

For more information on how the z/OS WLM health value affects CICSplex SM routing decisions, see the “Effect of the z/OS WLM health service on CICSplex SM workload routing” topic in IBM Knowledge Center at this website:

<https://ibm.biz/Bdzyie>

9.7.2 Throttle on number of MQGET calls issued by an MQMONITOR

In CICS TS V5.4 when the region's z/OS WLM health value is less than 100%, there is a throttle on the number of **MQGET** calls that an **MQMONITOR** can issue per second. In this way, the number of trigger tasks that are started is controlled. The throttle affects all started **MQMONITOR**s in the region. When the region's health value reaches 100%, the throttle on **MQGET** calls is removed. This behavior has been enhanced in CICS TS V5.5 by also reacting to the maximum tasks (MXT) condition.

If CICS encounters an MXT condition, the CICS-MQ Alert Monitor (CKAM) calculates the maximum number of **MQGET** calls that an **MQMONITOR** can issue per second when this condition exists. In effect, this action imposes a restriction on the number of tasks being started by **MQMONITOR** resources while CICS is at the MXT limit. While CICS is at the MXT limit the number of **MQGET** calls an **MQMONITOR** resource can issue per second is given by the calculation $MXT + 10\%$.

Note: The limit applied is a per-**MQMONITOR** resource limit and not a global limit. Tasks that are not associated with **MQMONITOR** resources will not be subject to any throttling.

For more information about how the z/OS WLM Health service affects IBM MQ resources in CICS TS V5.5, see the topic “Effect of z/OS Workload Manager health service on MQMONITORS” in IBM Knowledge Center at this website:

<https://ibm.biz/BdzMRG>

9.8 Disabling of VSAM dynamic buffer addition

From z/OS V2.2, VSAM provides a dynamic buffer addition capability that allows for the addition of extra buffers for an LSR pool if no buffer is available for a given VSAM request. For CICS, it is preferable to retry the request rather than allow uncontrolled expansion of an LSR pool, so dynamic buffer addition is not enabled for CICS LSR pools.

Note: The disabling of VSAM dynamic buffer addition was provided in all CICS V5 releases by APAR PI92486.

9.9 USS processes associated with L8, L9, X8, and X9 TCBs

CICS TS V5.5 now manages the release of USS (UNIX System Services) processes from X8, X9, L8, and L9 TCBs when the TCB is released from the CICS task and returned to the relevant CICS dispatcher pool of open TCBs.

The performance overhead of this additional USS process management was measured by using a development build of CICS TS V5.5. For each task that uses USS APIs, this overhead was measured to be approximately 410 μ s of CPU. Approximately half of the CPU overhead occurs in the CICS address space, and the remainder occurs in the OMVS address space. Of the CPU overhead measured in the CICS address space, approximately half of that is observed in the CICS performance class monitoring records.

A summary of the use of USS processes can be found in the topic “The SYS1.PARMLIB(BPXPRMxx) parameters” in IBM Knowledge Center at this website:

<https://ibm.biz/BdzMRe>

9.10 Channels performance improvement

Containers are named blocks of data that are designed for passing information between programs. Programs can pass any number of containers between each other. Containers are grouped in sets that are called *channels*. A channel is analogous to a parameter list. The CICS TS V5.5 release introduces a performance improvement that benefits applications where many containers are stored in a single channel.

CICS TS V5.5 improves performance by using a hash table to access containers, rather than searching a list. The performance improvement changes the order in which containers are returned when browsing a channel. Therefore, applications should not rely on the order in which containers are returned from calls to **EXEC CICS GETNEXT CONTAINER (CHANNEL)** commands. The CICS feature toggle `com.ibm.cics.container.hash` can be set to `false` to restore CICS to the previous behavior. For more information see the “Upgrading applications” topic in IBM Knowledge Center at this website:

<https://ibm.biz/Bdzyse>

This section presents a performance comparison of the two containers implementation options.

For more information about developing CICS applications by using channels, see the “Transferring data between programs using channels” topic in IBM Knowledge Center at this website:

<https://ibm.biz/Bdzysf>

9.10.1 Containers performance comparison

The application takes a value as input and creates the specified number of containers in a single channel. These are BIT containers 8 bytes in length. Next, the application reads each of these containers in reverse order and then the transaction completes. The overall response time and CPU per transaction is measured by using RMF. Several scenarios were tested, varying the number of containers from 10 to 750 per transaction.

The application was tested in both non-threadsafe and threadsafe configurations. The application was executed by using a development build of CICS TS V5.5 with the feature toggle `com.ibm.cics.container.hash` set to `false` and then set to `true` (the default). The use of the feature toggle provides the ability to directly compare CICS TS V5.4 and CICS TS V5.5 performance without other differences between the releases affecting the results.

The transactions were initiated from a terminal by using the methodology described in 2.4, “Driving the workload” on page 16 using 500 simulated clients. Where possible, the transaction rate was sustained at around 570 transactions per second.

Table 9-9 details the performance results obtained when running the workload in a non-threadsafe configuration with the `com.ibm.cics.container.hash` feature toggle set to `false`. This has the effect of using the CICS TS V5.4 channels and containers implementation.

Table 9-9 Performance data for a non-threadsafe configuration using V5.4 implementation

Number of containers	ETR	CPU per transaction (ms)	Response time (ms)
10	570.53	0.002	0.397
100	570.52	0.014	2.140
250	570.27	0.048	2.751
500	566.51	0.146	8.027
750	327.19	0.302	498.886

The data in Table 9-9 shows a significantly lower transaction rate and significantly higher response time for the scenario with 750 containers. During this test scenario, the QR TCB was fully utilized and became the primary bottleneck for the workload. From the 750 containers scenario in Table 9-9, we can make the observation that $327.19 \times 0.302 = 98.8\%$ utilization for the QR TCB.

Table 9-10 details the performance results when running the workload in a non-threadsafe configuration with the `com.ibm.cics.container.hash` feature toggle set to `true`. This is the default in CICS TS V5.5 and allows the use of the improved containers implementation.

Table 9-10 Performance data for a non-threadsafe configuration using V5.5 default implementation

Number of containers	ETR	CPU per transaction (ms)	Response time (ms)
10	570.54	0.002	0.448
100	570.54	0.011	1.659
250	570.55	0.027	2.687
500	570.28	0.057	2.828
750	570.21	0.093	3.093

The results of the non-threadsafe configuration are summarized for CPU time in Figure 9-7. The scenario using 750 containers has been omitted from the charts as this produces an unnecessarily large maximum value for the y-axis when measuring both CPU and response time.

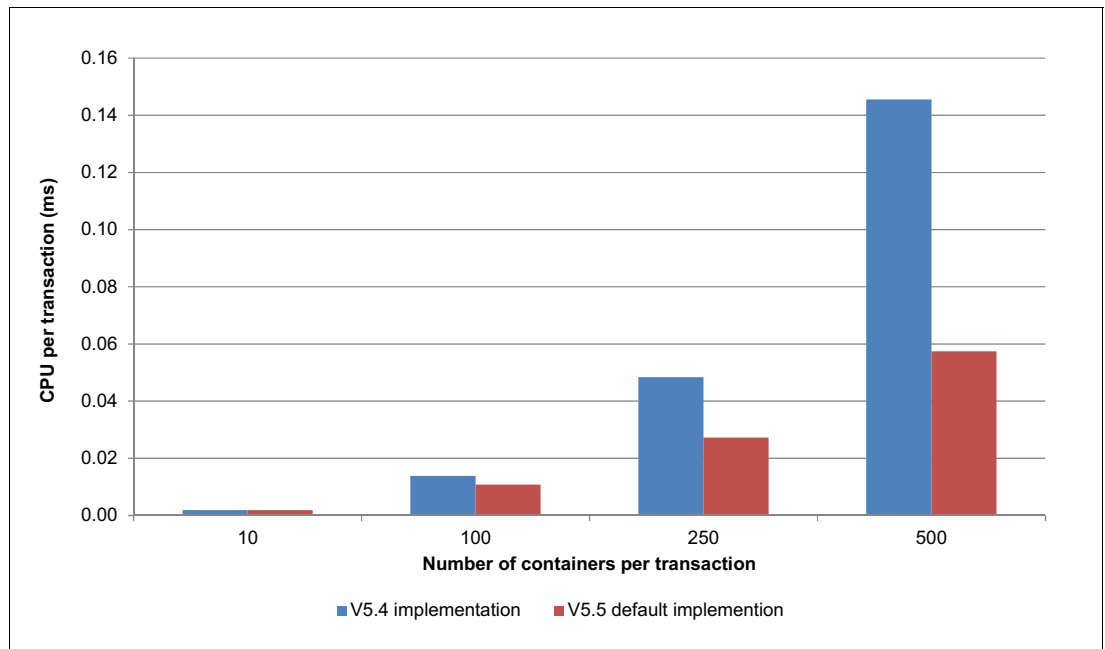


Figure 9-7 Summary of CPU per transaction for non-threadsafe configuration

The results of the non-threadsafe configuration are summarized for response time in Figure 9-8, again omitting the 750 containers scenario.

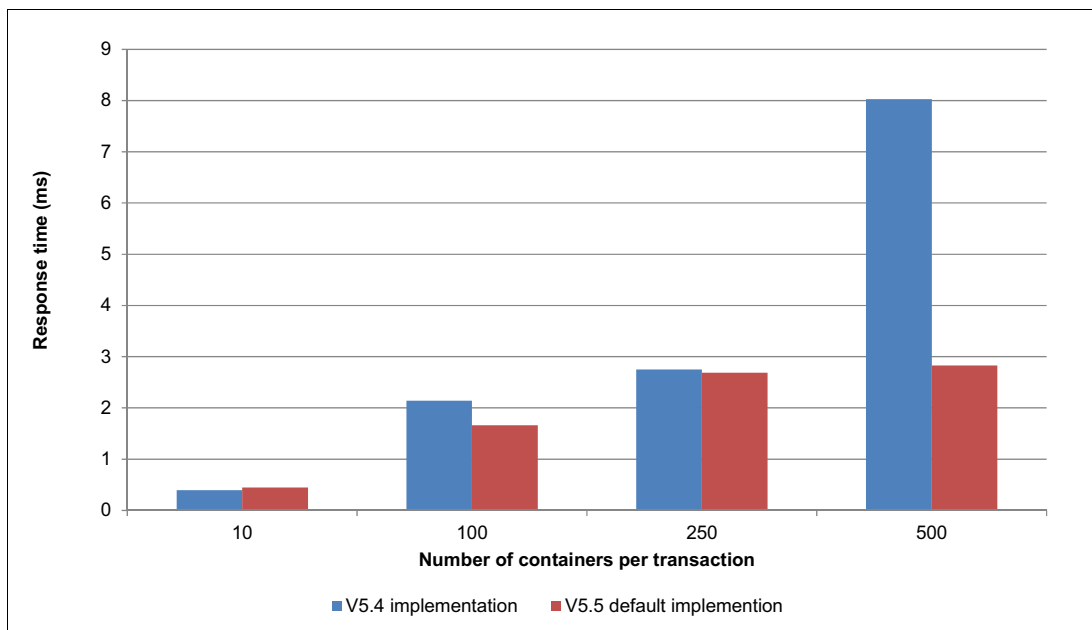


Figure 9-8 Summary of response time for non-threadsafe configuration

The application was next configured to execute using open TCBs and remove the QR TCB constraint. Table 9-11 on page 215 details the performance results when running the workload in a threadsafe configuration with the `com.ibm.cics.container.hash` feature toggle set to `false` — the equivalent of using the CICS TS V5.4 implementation.

Table 9-11 Performance data for a threadsafe configuration using V5.4 implementation

Number of containers	ETR	CPU per transaction (ms)	Response time (ms)
10	570.60	0.002	0.500
100	570.58	0.023	1.432
250	570.27	0.079	2.549
500	569.87	0.236	4.322
750	517.43	0.696	69.239

Finally, the test was repeated in a threadsafe configuration with the feature toggle set to `true` in order to use the improved CICS TS V5.5 implementation. The performance data for this test is detailed in Table 9-12.

Table 9-12 Performance data for a threadsafe configuration using V5.5 default implementation

Number of containers	ETR	CPU per transaction (ms)	Response time (ms)
10	570.54	0.002	0.523
100	570.50	0.018	1.144
250	570.19	0.055	2.248
500	569.97	0.113	3.207
750	565.04	0.244	10.917

The results of the threadsafe configuration are summarized for CPU time in Figure 9-9.

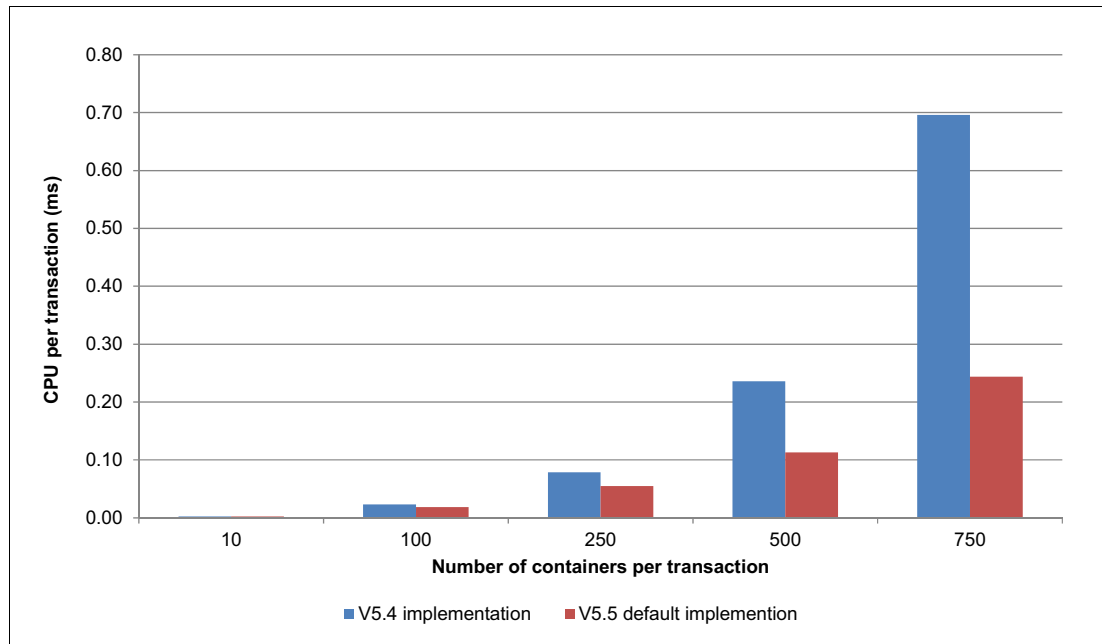


Figure 9-9 Summary of CPU per transaction for threadsafe configuration

The results of the threadsafe configuration are summarized for response time in Figure 9-10.

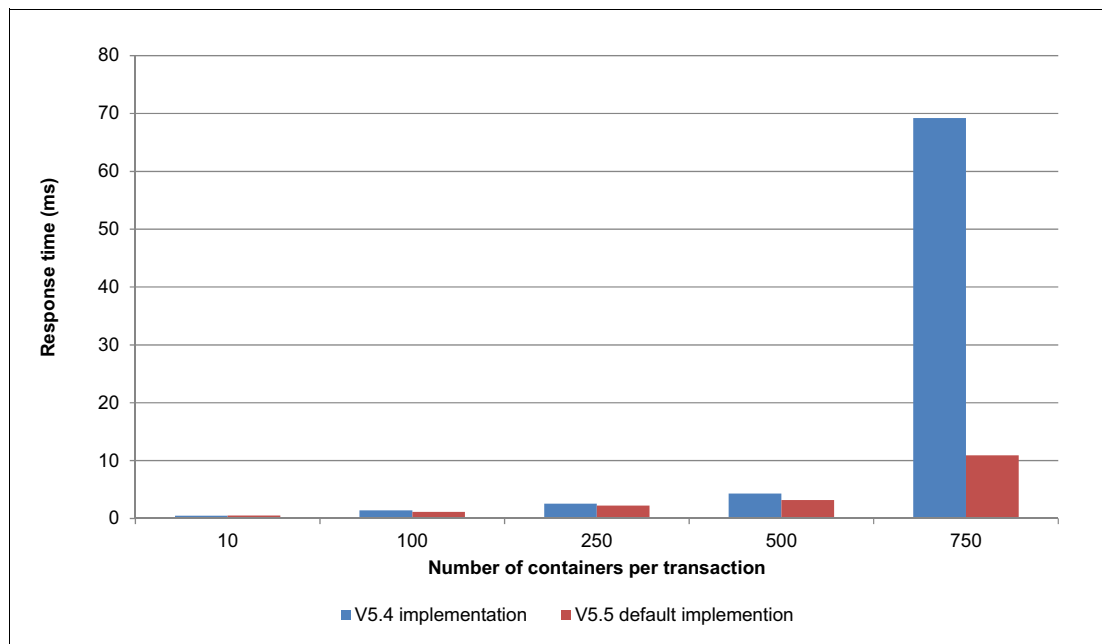


Figure 9-10 Summary of response time for threadsafe configuration

9.10.2 Containers performance summary

When using very small numbers of containers per transaction, the CPU consumed was equivalent regardless of internal implementation for both non-threadsafe and threadsafe configurations. Response times increased by a very small amount using the CICS TS V5.5 implementation. However, this increase is not expected to be significant in a real-world

application. In all test scenarios with more than 10 containers, both CPU and response times were improved when using the CICS TS V5.5 default implementation.

Scenarios with more than 750 containers were tested. However, for clarity their results are not included in this document. A maximum of 9,000 containers per transaction were tested and their results continued the trend that is demonstrated by the data in the tables above.

9.11 Threadsafe Coupling Facility Data Tables

As described in 9.3.2, “Coupling Facility Data Tables” on page 206, access to Coupling Facility Data Tables (CFDTs) is now threadsafe. Performance tests were executed comparing CICS TS V5.4 and CICS TS V5.5. The primary goal was to ensure that performance did not degrade when upgrading to the newest release, with the secondary goal to demonstrate the improved throughput available when using the threadsafe APIs.

9.11.1 CFDT performance test configuration

The threadsafe VSAM workload described in 3.7, “File control workload” on page 29 was used to validate the performance of CFDTs. Two files with record lengths of 64 bytes were defined as CFDTs and each specified the value of the UPDATEMODEL attribute to LOCKING. One CICS region was used and the workload was driven as described in 2.4, “Driving the workload” on page 16 using 1,000 simulated terminals.

The ratio of transactions used was as follows:

- ▶ 70% read only
- ▶ 30% update

Two files were defined, each with a record length of 64 bytes. Every transaction accessed 50 records in one of the defined files and this produced an average of 65 File Control requests per transaction with the following mix:

- ▶ EXEC CICS READ - 54%
- ▶ EXEC CICS READ UPDATE - 23%
- ▶ EXEC CICS REWRITE - 23%

Eight dedicated CPs were configured on the performance measurement LPAR, plus two dedicated CPs were configured in the Coupling Facility (CF). The LPAR and the CF were connected by using ICP links.

9.11.2 Non-threadsafe CFDT application performance results

The application was configured to run only on the QR TCB. Table 9-13 on page 217 presents the performance results when running the non-threadsafe application in a CICS TS V5.4 environment.

Table 9-13 CICS TS V5.4 results for non-threadsafe CFDT workload

ETR	CICS CPU	CPU per transaction (ms)	Response time (ms)
798.12	36.11%	0.452	1.312
1597.61	71.95%	0.450	2.963
2204.22	99.20%	0.450	131.748

The same non-threadsafe application test was executed using CICS TS V5.5 and the results are presented in Table 9-14.

Table 9-14 CICS TS V5.5 results for non-threadsafe CFDT workload

ETR	CICS CPU	CPU per transaction (ms)	Response time (ms)
798.09	36.21%	0.454	1.281
1597.65	72.06%	0.451	2.954
2203.78	99.32%	0.451	131.900

The CPU results for each CICS release are summarized in Figure 9-11.

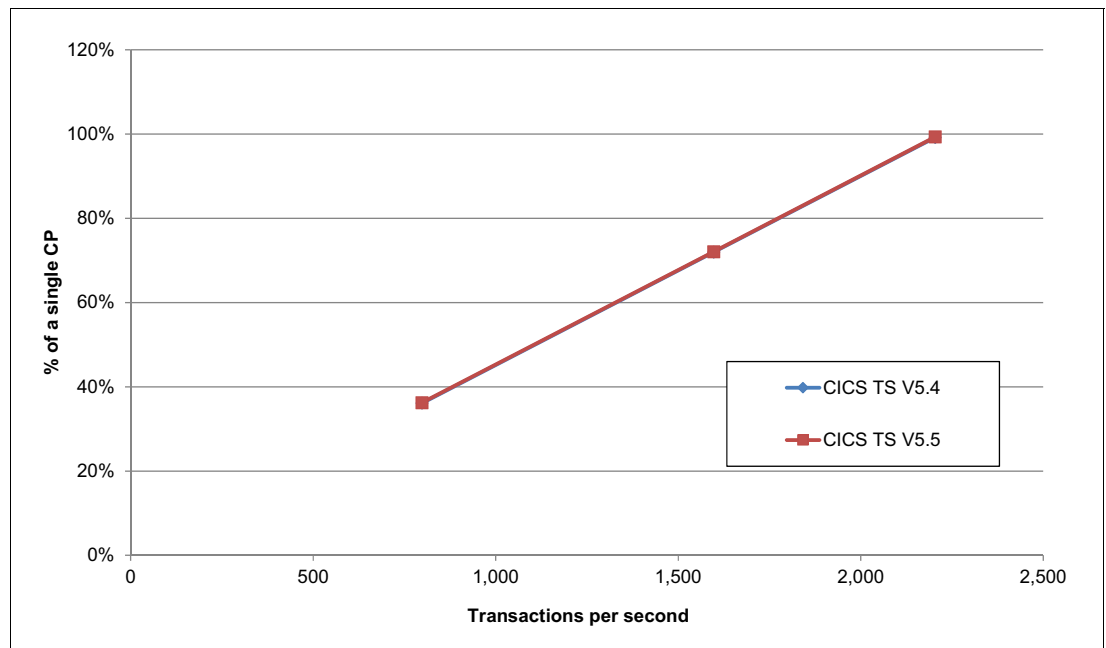


Figure 9-11 Plot of CICS TS V5.4 and V5.5 results for CFDT workload in non-threadsafe configuration

When running the benchmark in a non-threadsafe configuration, the throughput of the CICS region was limited by the capacity of the QR TCB. At peak throughput, during the 5-minute measurement interval the QR TCB was dispatched for more than 4 minutes 59.95 seconds (> 99.9%). From the data, it can be seen that the CPU per transaction for both the CICS TS V5.4 and the CICS TS V5.5 releases are equivalent, with the total CPU scaling linearly up to the throughput limit of approximately 2,200 transactions per second. Response times in both configurations increased dramatically as the transaction rate and hence the QR TCB utilization increased.

9.11.3 Threadsafes CFDT application performance results

The application program was configured with the CONCURRENCY attribute set to the value THREADSAFE. In both releases, the application started on an L8 TCB. In CICS TS V5.4 CFDT access is not threadsafe. Therefore, execution switches to the QR TCB at the time of first CFDT access and remains there until task termination. In CICS TS V5.5 CFDT access is threadsafe. Therefore, execution continues on the L8 TCB until the application writes a

completion message to the terminal. See Chapter 4, “Open transaction environment” on page 31 for a description of the TCB switching process.

Table 9-15 presents the performance results when running the application in a threadsafe configuration in CICS TS V5.4.

Table 9-15 CICS TS V5.4 results for threadsafe CFDT workload

ETR	CICS CPU	CPU per transaction (ms)	Response time (ms)
798.09	36.92%	0.463	1.474
1597.80	73.63%	0.461	2.440
2399.82	110.10%	0.459	93.556
2399.14	110.14%	0.459	243.851
2398.59	110.16%	0.459	293.903

The performance results for the CICS TS V5.5 release with a threadsafe configuration are presented in Table 9-16.

Table 9-16 CICS TS V5.5 results for threadsafe CFDT workload

ETR	CICS CPU	CPU per transaction (ms)	Response time (ms)
798.14	45.73%	0.573	0.883
1598.78	104.65%	0.655	1.295
3079.04	217.79%	0.707	1.881
5708.62	443.08%	0.776	3.285
7936.61	624.47%	0.787	4.641

The total CPU performance results are plotted in Figure 9-12 on page 220.

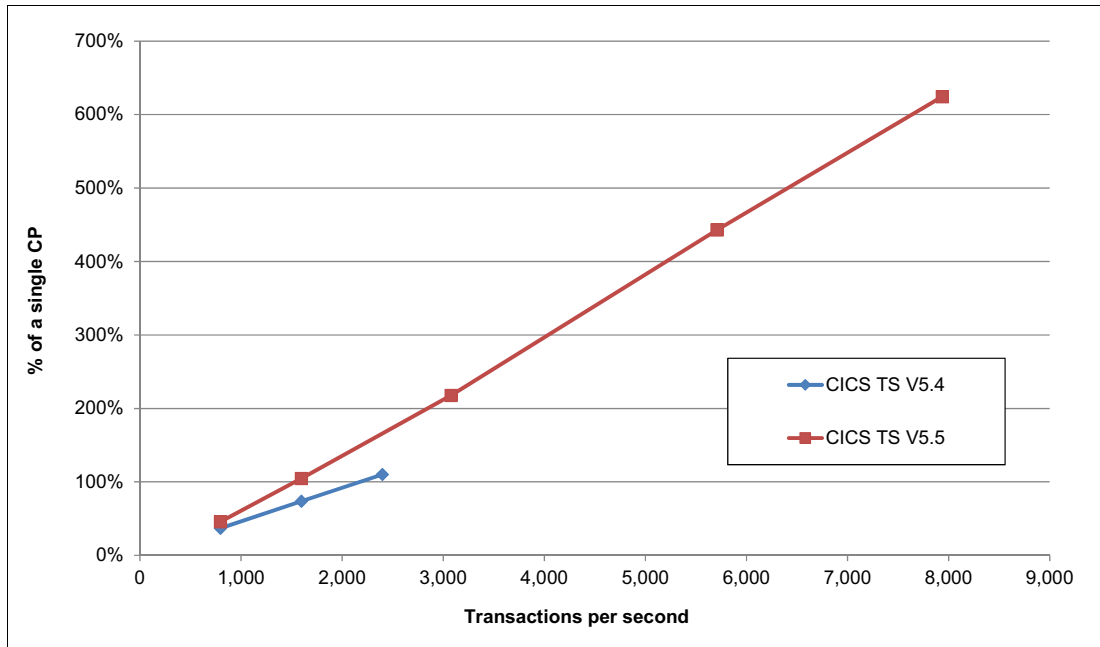


Figure 9-12 Plot of CICS TS V5.4 and V5.5 results for CFDT workload in threadsafe configuration

The response time data is plotted in Figure 9-13 on page 220.

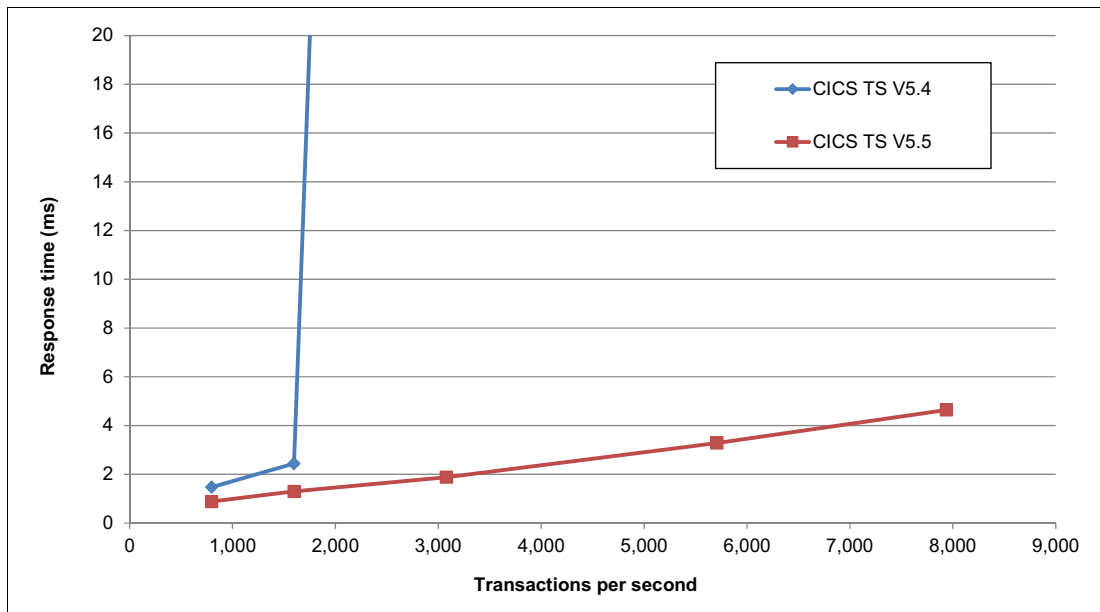


Figure 9-13 CICS TS V5.4 and V5.5 response time results for threadsafe CFDT workload

9.11.4 CFDT performance results summary

From the chart in Figure 9-12 on page 220 the following can be observed:

- ▶ As noted in the non-threadsafe configuration, the requirement to switch to the QR TCB for CFDT requests in the CICS TS V5.4 release causes a peak throughput of around 2,400 transactions per second. This QR TCB contention is removed in the CICS TS V5.5 release.

- ▶ Total CPU consumed by the CICS region scales linearly in CICS TS V5.4 up to the throughput limit.
- ▶ Total CPU consumed by the CICS region scales linearly in CICS TS V5.5 up to the tested throughput limit. Note that this testing limit was arbitrary: no CICS constraint existed at this point and could have increased further. Using this workload, a rate of 8,000 transactions per second correlates to over 500,000 CICS File Control requests per second in a single CICS region.
- ▶ The CPU cost per transaction in the CICS TS V5.5 release is greater than in the CICS TS V5.4 release. This is due to the significantly increased number of concurrent TCBs executing within the z/OS LPAR. Therefore, the CPU per transaction results cannot be directly compared. Section 9.11.2, “Non-threadsafe CFDT application performance results” on page 217 demonstrated that on a like-for-like per-API call basis, the two CICS releases provided equivalent performance.

The chart in Figure 9-13 on page 220 demonstrates the significant response time improvements that CICS TS V5.5 can provide for threadsafe workloads that access CFDTs. Non-threadsafe CFDT access in CICS TS V5.4 causes QR TCB saturation and extended response times. But the removal of the QR TCB constraint in CICS TS V5.5 provides better response times with greater scalability.

9.12 CICS policy rules

The behavior of CICS can be controlled during run time, based on predefined policies. CICS performs the action that is defined for a policy rule when all the conditions that are specified by the rule are met.

Policies define the action that CICS is to take when one of the following conditions is met:

- ▶ A CICS user task makes excessive use of system resources; for example, a user task consumes too much storage.
- ▶ A CICS system or user task changes the state of a system resource; for example, a FILE resource is closed.
- ▶ The overall system health changes; for example, the number of active tasks exceeds the maximum user tasks in the CICS system (the MXT value).

A condition and action pair make up a policy rule, and one or more policy rules can be defined within a policy. A policy is defined in a CICS bundle and a CICS bundle can consist of one or more policies. For more information on CICS policies, see the “CICS policies” topic in IBM Knowledge Center at this website:

<https://ibm.biz/Bdzy6A>

This section looks at the performance overhead of enabling policy task rules when running a standard performance benchmark application.

9.12.1 Policy task rules overhead performance study

The standard DSW static routing workload described in 3.2.1, “DSW static routing” on page 22 was used for the performance study. The benchmark was executed twice: one time with no policies installed and then again with a set of 19 policies installed. When combining all installed policies, this applied a task rule for every threshold supported by CICS TS V5.5.

The aim of this performance study is to measure the overhead of enabling task rules — not to measure the overhead executing the action. To ensure that no actions were invoked, all rules were coded with threshold values that would never be exceeded by the application. The implementation of the benchmark is such that not all rules can be triggered. For example, the DSW application does not use IBM Db2, therefore the task rules for **EXEC SQL** commands will never be used. The DSW application touches at least the following rules:

- ▶ **EXEC CICS** requests
 - Total number of **EXEC CICS** requests issued by the application
- ▶ File requests
 - **DELETE**
 - **READ**
 - **READNEXT**
 - **READ UPDATE**
 - **REWRITE**
 - **STARTBR**
 - **WRITE**
- ▶ Program requests
 - **LINK** commands
- ▶ Start requests
 - **START** commands
- ▶ Storage allocation
 - Task 24-bit storage
 - Task 31-bit storage
 - Shared 24-bit storage
 - Shared 31-bit storage
- ▶ Storage requests
 - Task 24-bit storage
 - Task 31-bit storage
 - Shared 24-bit storage
 - Shared 31-bit storage
- ▶ TD queue requests
 - **READQ**
 - **WRITEQ**
- ▶ Time
 - CPU time
 - Elapsed time
- ▶ TS queue bytes
 - **WRITEQ** all TS queue bytes
 - **WRITEQ** auxiliary TS queue bytes
- ▶ TS queue requests
 - **WRITEQ** all TS queue requests
 - **WRITEQ** auxiliary TS queue requests

RMF was used to obtain the transaction rate and CPU cost for the whole CICS region. Using this data, the average CPU per transaction value can be calculated. Table 9-17 lists the performance results for the configuration where no policies were installed.

Table 9-17 Results for DSW static routing workload with no policies installed

ETR	CICS CPU	CPU per transaction (ms)
4181.98	75.48%	0.180
4948.22	88.90%	0.180
6063.40	107.03%	0.177
6610.11	116.12%	0.176
7165.88	125.24%	0.175

The performance results for the configuration where all policies were installed are shown in Table 9-18.

Table 9-18 Results for DSW static routing workload with all 19 policies installed

ETR	CICS CPU	CPU per transaction (ms)
4176.96	75.36%	0.180
4932.58	88.96%	0.180
6057.89	107.33%	0.177
6602.11	115.99%	0.176
7176.31	126.33%	0.176

The results from Table 9-17 on page 223 and Table 9-18 on page 223 are shown in Figure 9-14.

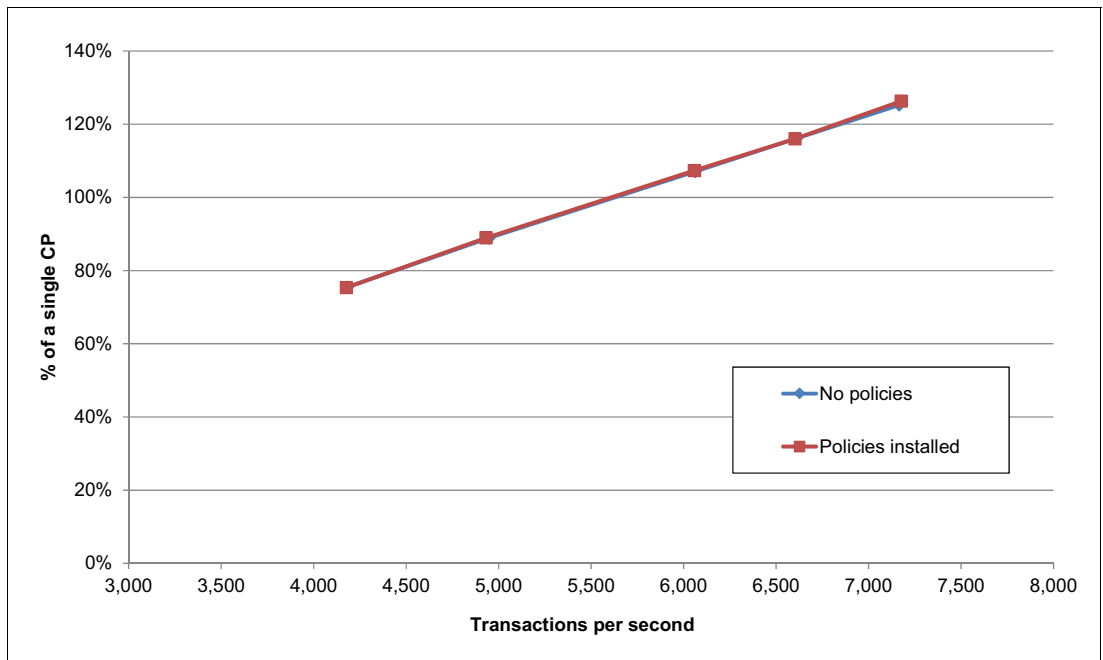


Figure 9-14 Plot of CICS TS V5.5 performance data with and without policy task rules installed

9.12.2 Policy task rules overhead performance summary

The performance data shows that there is no measurable overhead when using policy task rules to monitor user tasks. The data that is presented in Table 9-17 on page 223 and Table 9-18 on page 223 demonstrates that the CPU per transaction is equivalent within measurable limits. The data also shows that CICS continues to scale linearly as the transaction rate increases.

9.13 Encrypted zFS file systems

In z/OS V2.3 zFS added support for encrypting file system data using DFSMS access method encryption. This section presents the results of a performance test that investigated the overhead of using an encrypted zFS file system for a CICS workload. A development build of CICS TS V5.5 was used when testing encrypted zFS file system support. However, any release of CICS that uses the zFS file system can use this functionality.

For more information on encrypting zFS file system data, see the “Encrypting and compressing zFS file system data” topic in IBM Knowledge Center at this website:

<https://ibm.biz/BdzyJK>

9.13.1 zFS file system encryption performance comparison

The WebSphere Liberty workload described in 3.4, “WebSphere Liberty servlet with JDBC and JCICS access” on page 26 was used as the benchmark application. To generate significant quantities of zFS data, CICS tracing was enabled specifying the value of ALL for the SJ domain. For both the encryption disabled and the encryption enabled configurations, the transaction rate was sustained at approximately 1,650 requests per second. Enabling this level of trace at the given workload request rate resulted in approximately 30 MB of data that is written to zFS per second. Where enabled, the zFS file system used AES-256 encryption.

The performance data that is obtained during the test is listed in Table 9-19. The CPU per request is separated into that consumed by the CICS address space and that consumed by the ZFS address space. The overall zIIP eligibility is also presented for comparison.

Table 9-19 CPU cost comparison when enabling encryption for a zFS file system

Encryption	CICS CPU per request (µs)	ZFS CPU per request (µs)	zIIP eligibility
Disabled	2913.44	17.28	72.1%
Enabled	2933.98	20.18	72.4%

The CPU per request data that is presented in Table 9-19 is summarized in the chart in Figure 9-15.

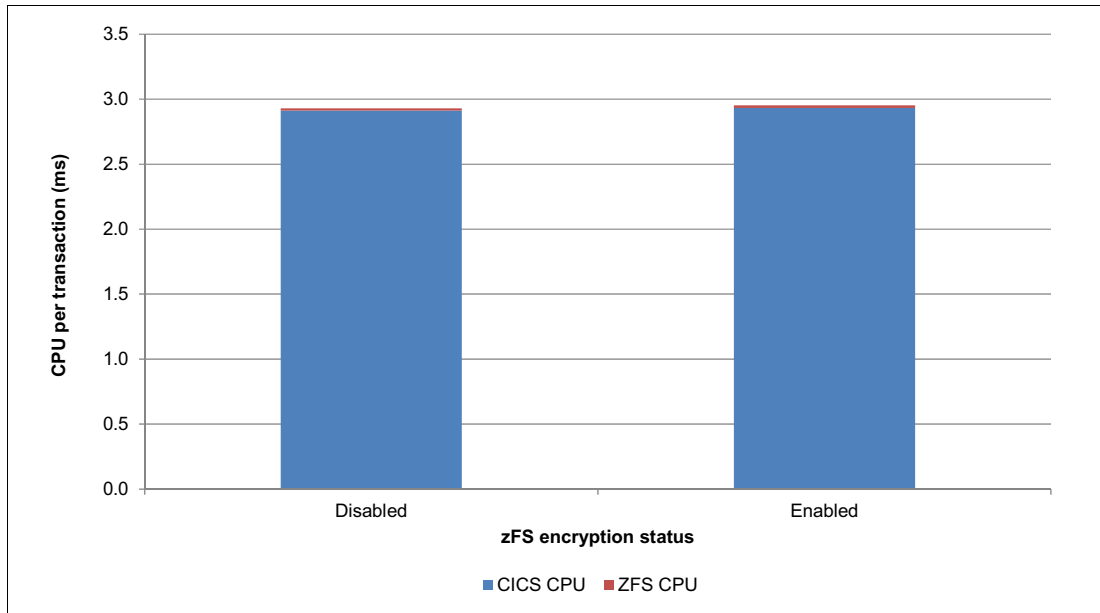


Figure 9-15 Summary of performance data for encrypted zFS

The chart in Figure 9-15 demonstrates that the CPU consumed by the ZFS address space is a very small fraction of the overall CPU consumption per request. To more clearly demonstrate the difference in CPU attributed to the ZFS address space when enabling zFS encryption, only the ZFS address space data is plotted in Figure 9-16 on page 225.

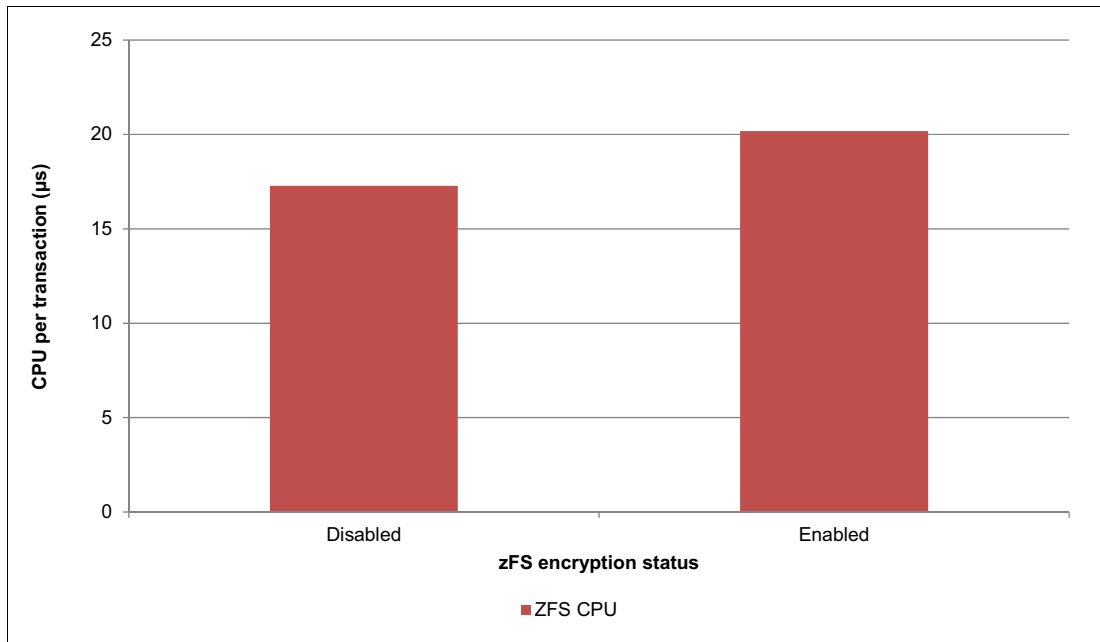


Figure 9-16 Summary of performance data for the ZFS address space when enabling zFS encryption

9.13.2 zFS file system encryption performance summary

The total CPU overhead for this workload when writing encrypted data to zFS is very small: approximately 23 µs per request. Although the ZFS address space showed a significant relative increase in CPU cost per request (+17%), the overall total cost to the workload was

negligible. As observed in Table 9-19 on page 224, the overall zIIP eligibility of the workload remained unchanged.

The use of zFS encrypted file systems is fully supported in a CICS environment and the CPU overhead is expected to be negligible in a full production workload.

9.14 Multiple Liberty JVM servers in a single CICS region

CICS TS V5.5 introduced the ability to run multiple Liberty JVM servers in a single CICS region. It is no longer necessarily to disable angel process security with the (deprecated) JVM server option `WLP_ZOS_PLATFORM=FALSE` to achieve multi-tenancy of JVM servers in a single region.

Note: APAR PI98174 enables the ability to run multiple Liberty JVM servers in a single CICS region for CICS TS V5.4.

This section examines the performance and storage characteristics of running multiple Liberty JVM servers within a single CICS region when compared with JVM servers across multiple CICS regions.

9.14.1 Shared libraries support

The shared library region is a z/OS USS feature that enables address spaces to share dynamic link library (DLL) files. This feature enables CICS regions to share the DLLs that are needed for JVMs, rather than each region loading them individually. A CICS address space utilizes shared library support if *any* of the USS processes or JVM servers within the CICS region enable shared library support. In CICS TS V5.4 and earlier this feature was enabled by default, but in CICS TS V5.5 (with APAR PH09400) it must be enabled using the `_BPXK_DISABLE_SHLIB=N0` parameter in the JVM profile.

Using shared libraries support can reduce the amount of real storage that is used by z/OS and the time it takes for the regions to load the DLL files. The disadvantage of using shared libraries is that any address space that uses this feature reserves an area of 31-bit virtual storage that is equal in size to the value of the z/OS `SHRLIBRGNSIZE` parameter, which is likely to increase the virtual storage footprint of each region.

Note: When shared libraries are enabled, the full size of the 31-bit area will be allocated, regardless of the utilization achieved by an individual address space. Therefore, it is important to adjust the `SHRLIBRGNSIZE` parameter to accommodate all the libraries, but avoid over-allocation and waste 31-bit virtual storage.

For more information on the use of the shared library region in JVM servers within a CICS environment, see the “Tuning the z/OS shared library region” topic in IBM Knowledge Center at this website:

<https://ibm.biz/Bdzv37>

9.14.2 Multiple Liberty JVM servers performance workload configuration

The hardware that is used for the benchmarks is described in 9.1, “Introduction” on page 198. The measurement LPAR was configured with three GCPs and three zIIPs running in SMT

mode 1, which resulted in an LSPR equivalent processor of 3906-706. The measurement LPAR had 16 GB of real storage allocated.

The measurement LPAR was running z/OS V2.3 and a development build of CICS TS V5.5. Java 8.0 SR5 was used with the following options:

- ▶ RMODE64 enabled
From z/OS V2.3, 64-bit residency mode for applications (RMODE64) is enabled by default. This feature allows the JIT to allocate executable code caches above the 2 GB memory bar.
- ▶ Compressed references enabled
The IBM Java SDK for z/OS can use *compressed references* on 64-bit platforms to decrease the size of Java objects and make more effective use of the available space. The result is less frequent garbage collection and improved memory cache utilization.
- ▶ Shared libraries disabled
See section 9.14.1, “Shared libraries support” on page 226 for a discussion of shared libraries.

All CICS regions set the MXT parameter to 150 and all JVM servers specified the value of 64 for the THREADLIMIT parameter.

The application used was the standard servlet workload as described in 3.4, “WebSphere Liberty servlet with JDBC and JCICS access” on page 26. CICS Liberty security was enabled and all Liberty JVM servers were connected to the same WebSphere Liberty angel process. The workload was driven through HTTP requests by using IBM Workload Simulator for z/OS, as described in 2.4, “Driving the workload” on page 16. The workload used 1,000 simulated web browsers, each supplying a username and password via HTTP basic authentication.

The application was cloned to produce five versions that can be deployed in separate Liberty JVM servers that used different TCP/IP ports. The configurations tested were:

1. One CICS region with one Liberty JVM server
2. One CICS region with three Liberty JVM servers
3. Three CICS regions each with one Liberty JVM server
4. One CICS region with five Liberty JVM servers
5. Five CICS regions each with one Liberty JVM server

9.14.3 Comparing CPU costs per request and maximum throughput

The workload was run in all five configurations. The total CPU cost and number of transactions completed was obtained by using IBM Resource Measurement Facility (RMF). Using this data, the CPU per request and the throughput rate was calculated. z/OS storage information was obtained using CICS MVS TCB statistics data.

Table 9-20 on page 228 presents the CPU per request and total throughput data for each of the five configurations.

Table 9-20 CPU per request comparison for multiple JVM server configurations

Scenario	Not zIIP-eligible CPU per request (ms)	zIIP-eligible CPU per request (ms)	Throughput (requests per sec)
1 CICS region with 1 Liberty JVM server	0.74	0.75	3,245
1 CICS region with 3 Liberty JVM servers	1.10	0.93	2,766
3 CICS regions each with 1 Liberty JVM server	0.99	1.05	2,804
1 CICS region with 5 Liberty JVM servers	1.14	0.96	2,695
5 CICS regions each with 1 Liberty JVM server	1.02	1.07	2,736

The CPU per request data from Table 9-20 is presented in Figure 9-17, separated into non-zIIP-eligible and zIIP-eligible components.

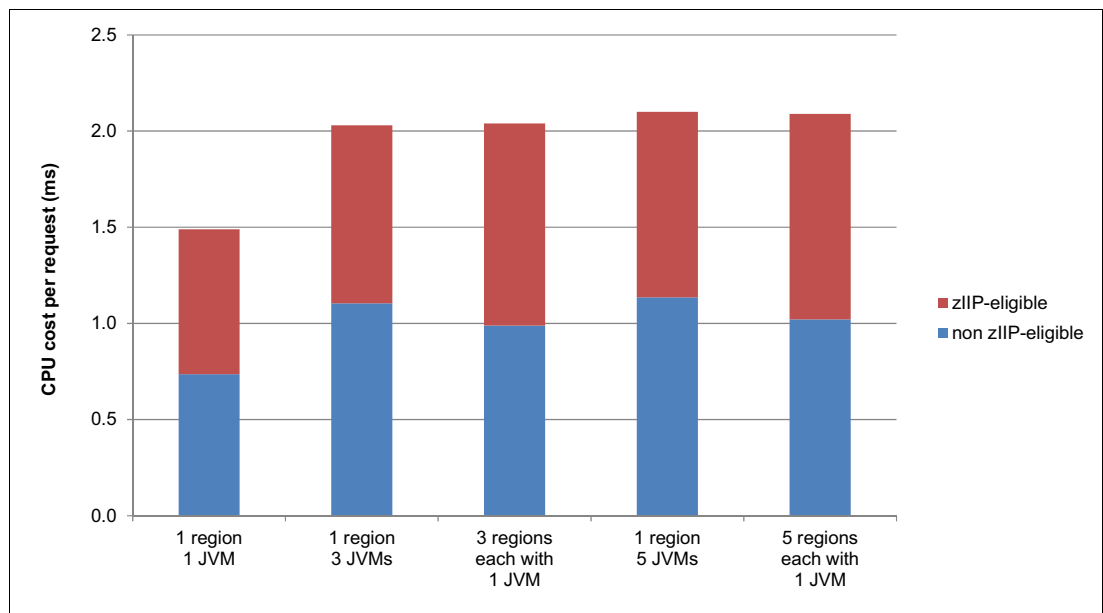


Figure 9-17 Chart showing CPU per request comparison for multiple JVM server configurations

It can be seen from the Figure 9-17 that the lowest CPU cost per request was provided by single JVM server in a single CICS region (configuration 1). This lower cost is because the JVM in configuration 1 will process more requests than each individual JVM used in configurations 2 through 5. The more requests that are processed by a JVM, the more effectively the JIT compiler can optimize the code path, resulting in a lower CPU per request.

When running at very high CPU utilization with multiple JVM servers in a single CICS region, there are a large number of TCBs active in the CICS address space. This causes increased z/OS dispatcher activity, which slightly reduces the zIIP eligibility by reducing the zIIP *lazy switching* benefit. At lower throughput rates with lower CPU utilization — which is more likely in a customer production system — the zIIP eligibility was seen to be similar for all configurations investigated.

zIIP lazy switching is described in the *IBM Systems Magazine* article *Understanding zIIP Usage in CICS*:

<http://archive.ibmssystemsmag.com/mainframe/administrator/cics/ziip-usage/>

As an example of the additional JIT optimization, at the end of the test the JVM in configuration 1 (one CICS region with one Liberty JVM server) had optimized a total of 7,079 Java methods. Conversely, one of the JVMs in configuration 5 (five CICS regions each with one Liberty JVM server) had optimized only 1,362 Java methods.

The total throughput data from Table 9-20 is presented in Figure 9-18 on page 229.

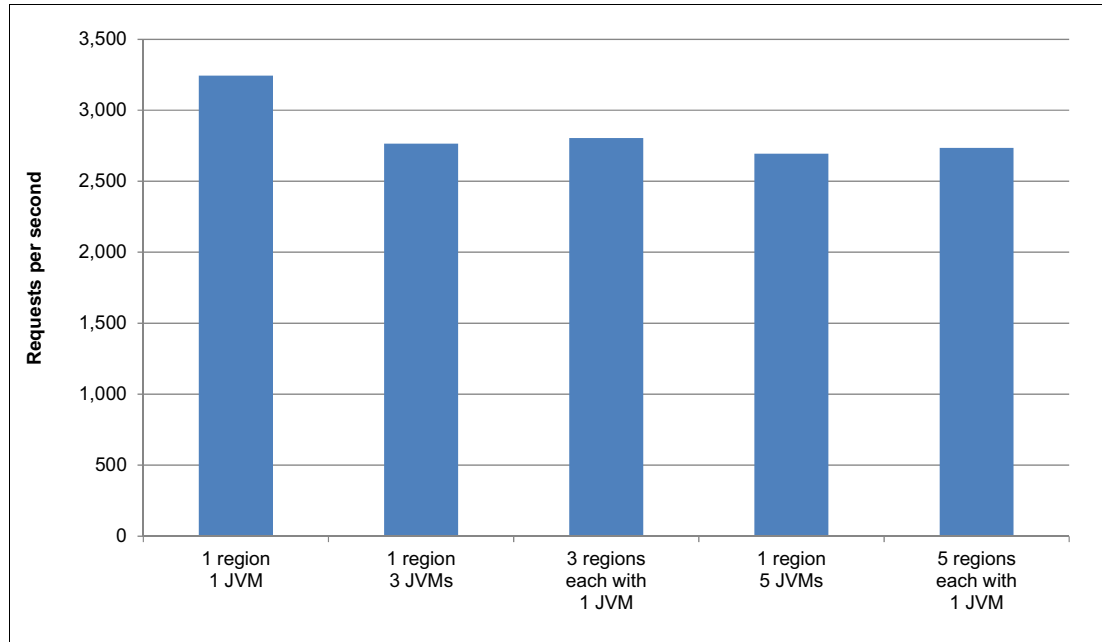


Figure 9-18 Chart showing total throughput comparison for multiple JVM server configurations

During all test scenarios, the LPAR was 97% busy and therefore the throughput was limited by the CPU cost per request.

9.14.4 Comparing 31-bit memory usage

The amount of 31-bit storage used was collected from the CICS MVS TCB statistics data and is summarized in Figure 9-19. Where a configuration used multiple CICS regions, the chart presents the average amount of 31-bit storage that was used per CICS region.

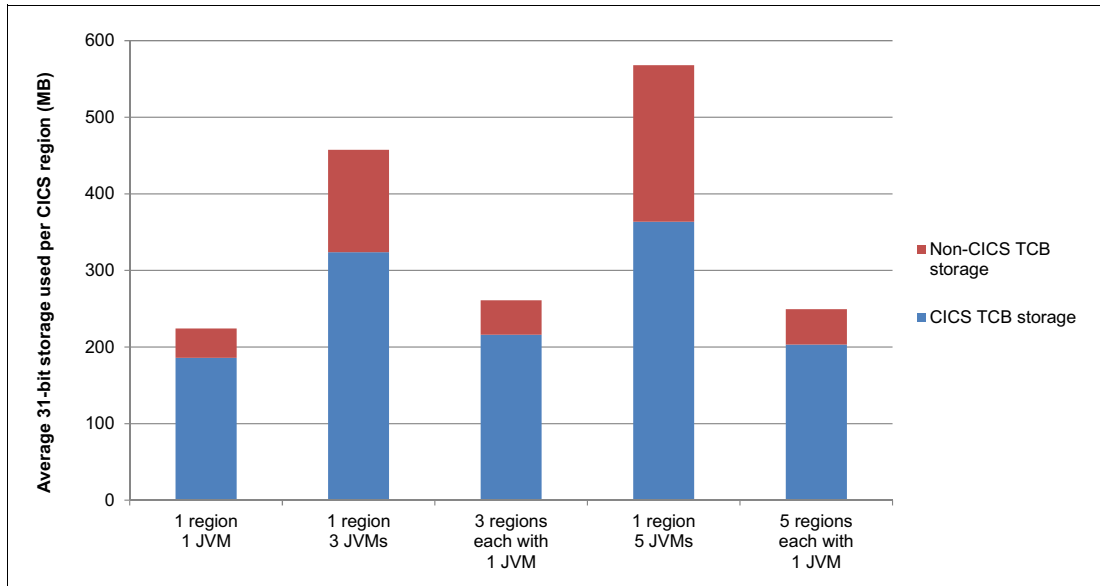


Figure 9-19 Summary of 31-bit storage used per CICS region for multiple JVM server configurations

The amount of CICS TCB storage per CICS region is related to the number of concurrent tasks and TCBs used. To restrict the number of concurrent TCBs in a CICS region for a Java workload, use the `THREADLIMIT` attribute of the `JVMSEVER` resource definition.

The amount of non-CICS TCB storage that is used per CICS region is related to the number of JVM servers. As documented in 9.14.2, “Multiple Liberty JVM servers performance workload configuration” on page 226, this test used compressed references. Disabling of compressed references reduces the amount of 31-bit storage that is used, at the expense of some CPU and 64-bit storage usage.

Each configuration that only had one JVM per CICS region shows very similar 31-bit storage usage. Where multiple JVM servers are configured per CICS region, the increased storage use is a result of each JVM having its own private copy of runtime data. These copies are mostly held in non-CICS TCB storage.

In contrast to Figure 9-19 on page 230 that presented the storage that is used per CICS region, the chart in Figure 9-20 summarizes the total 31-bit storage that is used across all CICS regions in a given configuration.

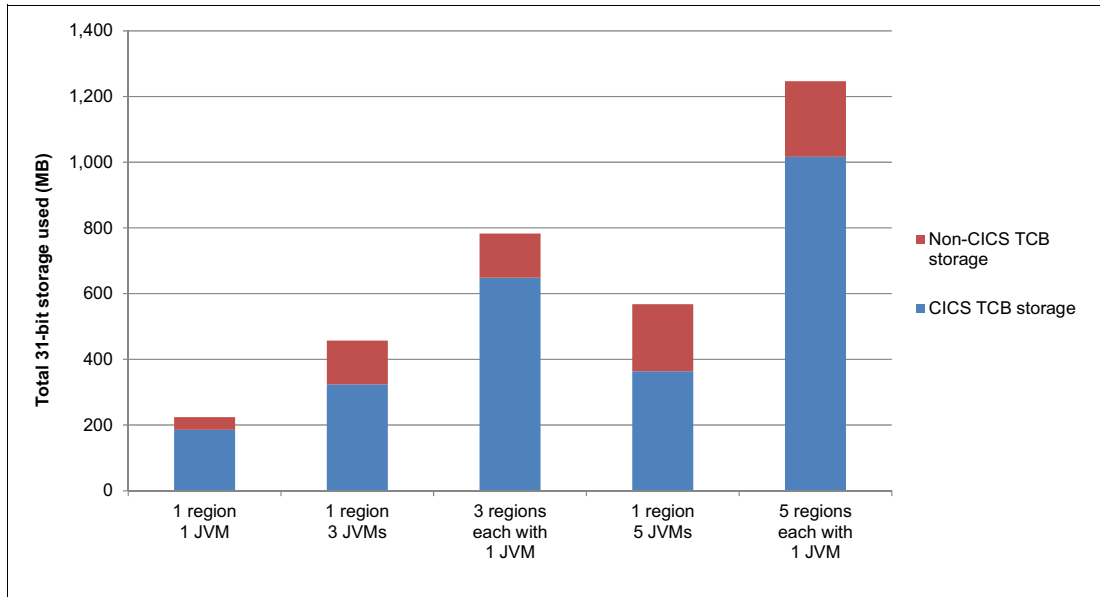


Figure 9-20 Summary of total 31-bit storage used for multiple JVM server configurations

The data in Figure 9-20 demonstrates the storage savings that are achieved by using multiple JVM servers in a single CICS region, compared to using one JVM server in multiple CICS regions.

9.14.5 Comparing 64-bit memory usage

To achieve the required concurrency, the JVM used in configuration 1 (one CICS region with one Liberty JVM server) specified a heap size of 1000 MB. All other JVMs specified a heap size of 200 MB.

The amount of 64-bit storage that was used was collected from the CICS storage statistics data and is summarized in Figure 9-21 on page 232. Where a configuration used multiple CICS regions, the chart presents the average amount of 64-bit storage that is used per CICS region. The shared class cache is held in a z/OS shared memory object. A shared class cache will be included in the 'Bytes Allocated Shared Memory Objects' data that is reported in the CICS storage overview statistics report, but does not count toward the MEMLIMIT of the CICS region.

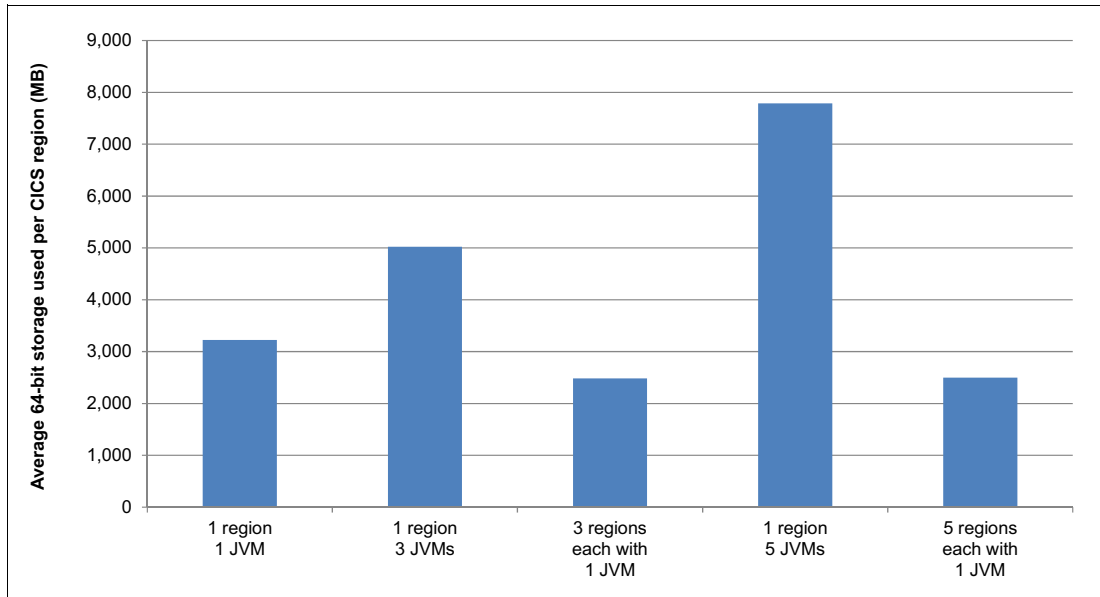


Figure 9-21 Summary of 64-bit storage used per CICS region for multiple JVM server configurations

As expected, the amount of 64-bit storage that is used per CICS region is related to the number of JVM servers configured. Each JVM server requires its own copy of 64-bit runtime data areas including heap and JIT caches.

The total 31-bit storage used was presented in Figure 9-20 on page 231 and Figure 9-22 presents a similar view of total 64-bit storage usage across all CICS regions.

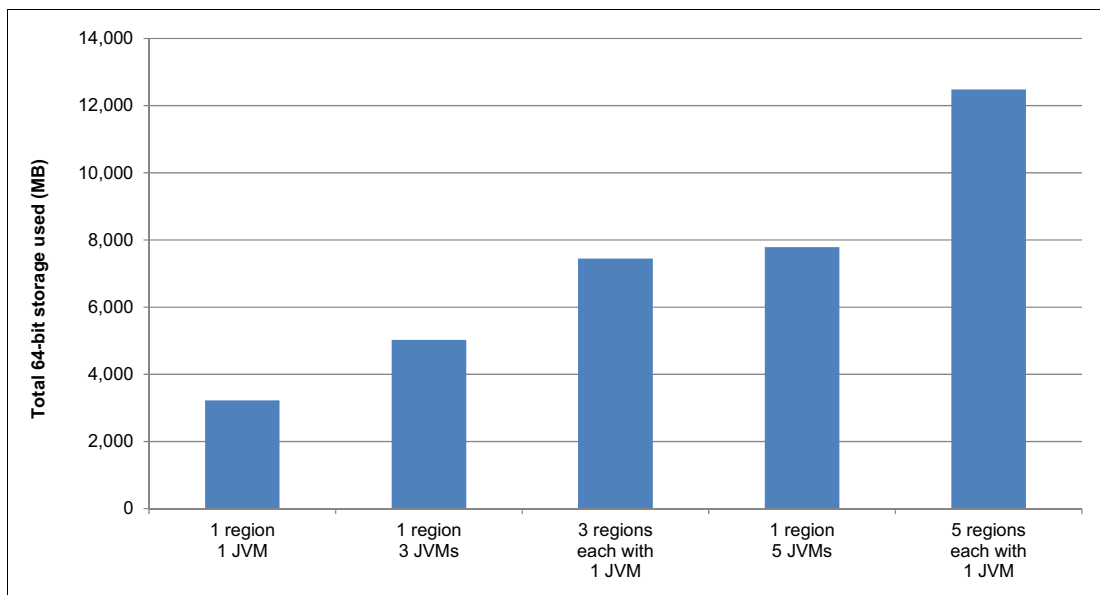


Figure 9-22 Summary of total 64-bit storage used for multiple JVM server configurations

Section 9.14.4, “Comparing 31-bit memory usage” on page 229 demonstrated how multiple JVMs in a single CICS region gives a reduction in overall 31-bit storage that is used and Figure 9-22 shows this is also true for 64-bit storage usage.

9.14.6 Multiple Liberty JVM servers performance conclusion

The most efficient configuration is a single large JVM server when you consider the following:

- ▶ CPU costs

Multiple JVM servers may not JIT methods to the same level of optimization as a more frequently used single JVM server.

Multiple JVM servers will probably use more CICS T8 TCBs and will each have their own set of JVM-related TCBs (such as JIT and GC helpers). Management of these additional TCBs introduces an extra CPU overhead.

- ▶ Throughput

The increased cost per request of using multiple JVM servers means that the maximum throughput, when all CPU resource is consumed, is lower than for a single JVM server configuration.

- ▶ Memory

The 31-bit memory usage per CICS region significantly increased when using multiple JVM servers.

The use of 31-bit memory can be minimized by using Java 8.0 SR5 with z/OS v2.3 to place JIT code caches in 64-bit memory. JIT data caches are always in 64-bit memory.

Restricting the number of CICS T8 TCBs by specifying low values for the JVMSERVER THREADLIMIT attribute reduces ECDSA use (each CICS TCBs requires 28 KB of kernel stack storage).

Using uncompressed references with the JVM profile setting `-Xnocompressedrefs` moves all Java class data to 64-bit memory.

As described in 9.14.1, “Shared libraries support” on page 226 the use of shared libraries has an impact on the amount of 31-bit storage allocated. The size of the shared library area is controlled by the z/OS SHRLIBRGNSIZE parameter.

This study does not report on response times. However, no significant difference was observed across all of the configurations measured.

Although a single large JVM server can provide the best performance, this does not provide high availability or application separation. When you deploy applications, consideration should also be given to the following requirements:

- ▶ Protection against the failure of an individual JVM server
- ▶ Protection against the failure of an individual CICS region
- ▶ Protection against the failure of a z/OS LPAR
- ▶ The ability to apply maintenance to the application, CICS or z/OS

9.15 Liberty JVM server and application startup times

After enabling a JVMSERVER resource, the Liberty environment and hosted applications require a finite amount of time to start. This section looks at minimizing this startup time by using shared class cache. This section also examines the time that is taken to start multiple Liberty JVM servers in a single CICS region.

The time taken for a CICS Liberty JVM server to start is measured from enabling the JVMSERVER resource until the CWWKF0011I message is emitted. The time taken for a Liberty application to start is reported in the CWWKZ0001I message.

9.15.1 Startup times with shared class cache

The class sharing feature offers the transparent and dynamic sharing of data between multiple JVMs. When enabled, JVMs use shared memory to obtain and store data, including information about: loaded classes, Ahead-Of-Time (AOT) compiled code, commonly used UTF-8 strings, and Java Archive (JAR) file indexes. For more information, see the “Class data sharing” topic in IBM Knowledge Center at this website:

<https://ibm.biz/BdzSkf>

Using class data sharing, the time that is required to start a CICS Liberty JVM server and Liberty applications within this server can be reduced. The use of the `-Xtune:virtualized` JVM option further improves JVM and application startup time. For more information, see the “-Xtune:virtualized” topic in IBM Knowledge Center at this website:

<https://ibm.biz/BdzSkv>

These timings are presented for each of four configurations in Table 9-21.

Table 9-21 Summary of startup timings with varying shared cache class configurations

Configuration	Liberty startup time (s)	Application startup time (s)
No class cache	9.726	0.973
Class cache (first use)	10.427	1.147
Class cache (second use)	3.939	0.414
Class cache (second use) with <code>-Xtune:virtualized</code>	3.603	0.378

The time taken for the Liberty JVM server to start is plotted in Figure 9-23.

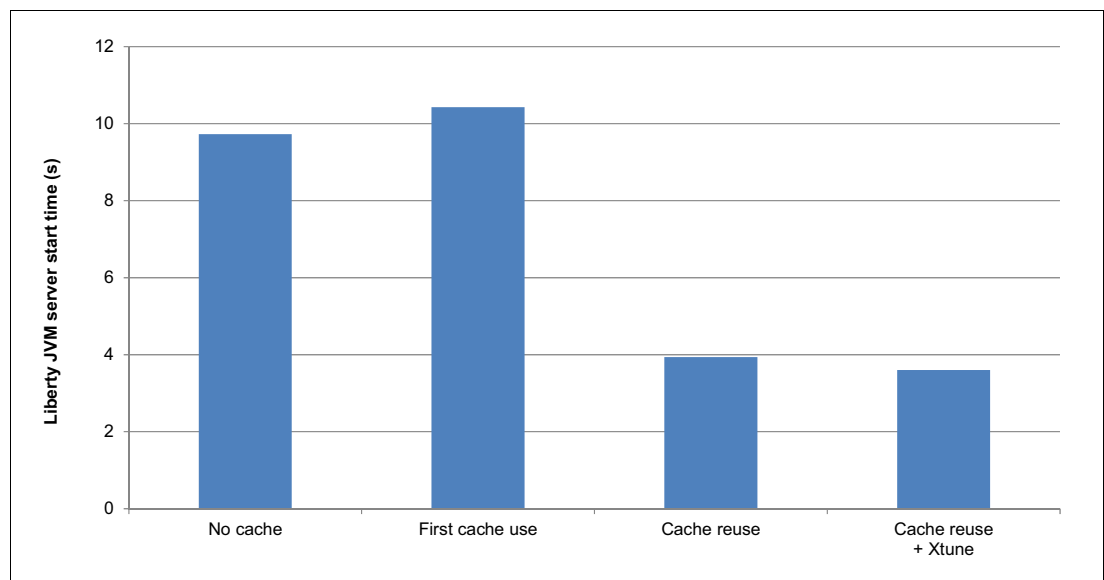


Figure 9-23 Summary of Liberty JVM server startup time with varying class cache configurations

The time taken for the application to start is plotted in Figure 9-24 on page 235.

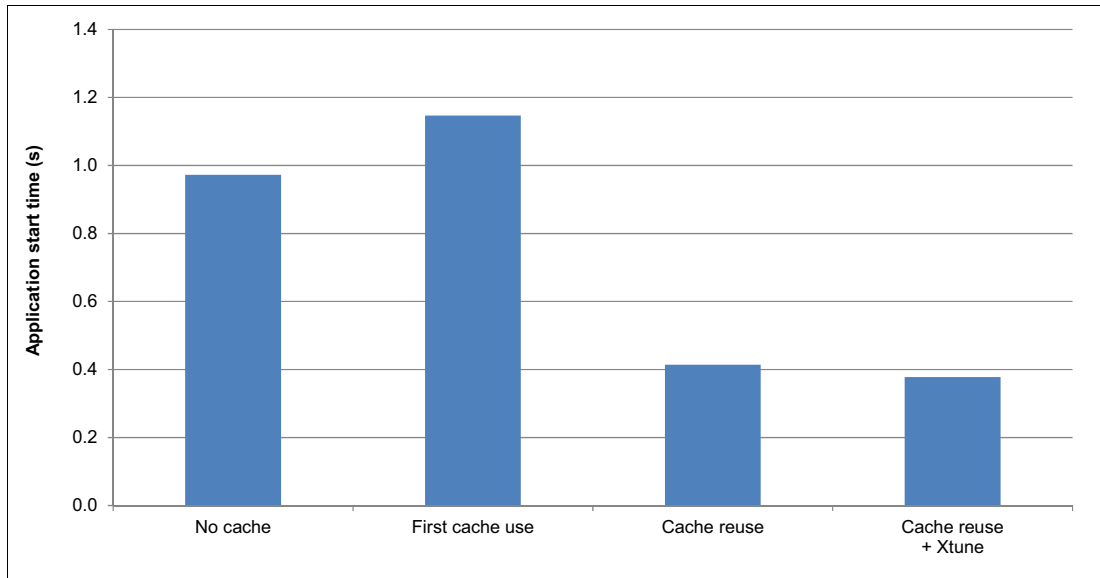


Figure 9-24 Summary of application startup time with varying class cache configurations

It can be seen that the first use of a shared class cache slightly increases startup times for both the Liberty JVM server and any applications. However, subsequent starts are significantly improved with shared class cache enabled. The use of the `-Xtune:virtualized` option slightly reduces startup times in addition to the benefits of a shared class cache.

9.15.2 Application startup times with multiple JVM servers

This section looks at application startup times along with section 9.14, “Multiple Liberty JVM servers in a single CICS region” on page 226. The time taken to start each application was recorded, firstly when running five JVMs in one CICS region, then when running one JVM in each of five CICS regions.

For all JVM configurations shared class cache was enabled and was already been populated before the test. The `-Xtune:virtualized` option was also specified. The time taken to start an application in each instance of a JVM is plotted in the chart in Figure 9-25 on page 236.

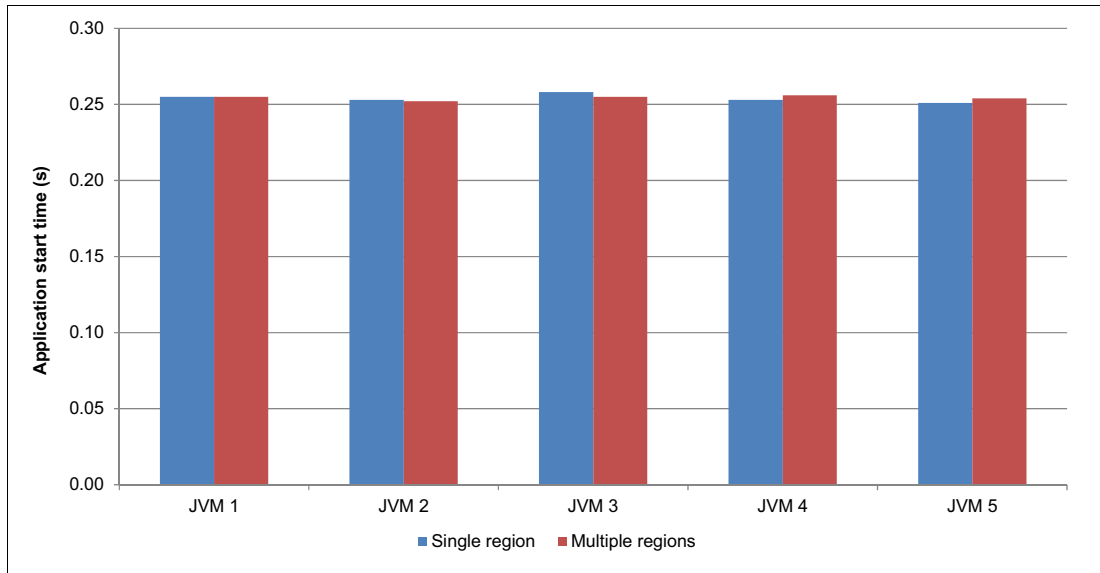


Figure 9-25 Summary of application startup times with single and multiple CICS regions

The results plotted in Figure 9-25 show that there is no significant difference in application startup time using one JVM server in multiple CICS regions, when compared to using multiple JVM servers in a single CICS region.

Related publications

The publications that are listed in this section are considered particularly suitable for a more detailed discussion of the topics that are covered in this book.

CICS performance series

This book is one in a series that is focused on CICS performance. The following IBM Redbooks publications are also part of this CICS performance series. Some publications that are referenced in this list might be available in softcopy only:

- ▶ *IBM CICS Performance Series: A CPU Utilization Study of Java EE applications running in CICS TS V5.3*, REDP-5361
- ▶ *IBM CICS Performance Series: Web Services Performance in CICS TS V5.3*, REDP-5322
- ▶ *IBM CICS Performance Series: CICS TS V5.3 Benchmark on IBM z13*, REDP-5320
- ▶ *IBM CICS Performance Series: Comparing Type 2 and Type 4 JDBC Driver Performance with IBM CICS Transaction Server for z/OS V5.2 Liberty JVM server*, REDP-5208
- ▶ *IBM CICS Performance Series: Effective Monitoring for CICS Performance Benchmarks*, REDP-5170
- ▶ *IBM CICS Performance Series: FiTeq Authenticator Benchmark*, REDP-5114
- ▶ *IBM CICS Performance Series: A Processor Usage Study of Ways into CICS*, REDP-4906
- ▶ *IBM CICS Performance Series: CICS and VSAM RLS*, REDP-4905
- ▶ *IBM CICS Performance Series: CICS, DB2, and Thread Safety*, REDP-4860
- ▶ *IBM CICS Performance Series: CICS TS V4.2 and Java Performance*, REDP-4850

Other IBM Redbooks publications

The IBM Redbooks publication *Setting Up and Using the IBM System z CPU Measurement Facility with z/OS*, REDP-4727, provides more information about the topic in this document.

You can search for, view, download or order this documents and other Redbooks, Redpapers, Web Docs, draft and other materials, at the following website:

<http://www.redbooks.ibm.com/>

Other publications

The publication *z/Architecture Principles of Operation*, SA22-7832, is also relevant as a further information source and is available at this website:

<http://www.ibm.com/support/docview.wss?uid=isg2b9de5f05a9d57819852571c500428f9a>

The article *Understanding zIIP Usage in CICS* provides background on how zIIP processors interact with CICS and z/OS and is available on the IBM Systems Magazine website:

<http://archive.ibmssystemsmag.com/mainframe/administrator/cics/ziip-usage/>

Online resources

The following websites are also relevant as further information sources:

- ▶ IBM CICS Family home page:
<https://www.ibm.com/it-infrastructure/z/cics>
- ▶ Developer Center for CICS:
<https://developer.ibm.com/cics/>
- ▶ CICS performance resources
<https://developer.ibm.com/cics/cics-performance-resources/>
- ▶ CICS Transaction Server for z/OS V5.1 in IBM Knowledge Center:
<https://ibm.biz/Bd4zFv>
- ▶ CICS Transaction Server for z/OS V5.2 in IBM Knowledge Center:
<https://ibm.biz/BdXUWA>
- ▶ CICS Transaction Server for z/OS V5.3 in IBM Knowledge Center:
<https://ibm.biz/Bd4zFm>
- ▶ CICS Transaction Server for z/OS V5.4 in IBM Knowledge Center:
<https://ibm.biz/BdiuGX>
- ▶ CICS Transaction Server for z/OS V5.5 in IBM Knowledge Center:
<https://ibm.biz/BdznLZ>
- ▶ Large Systems Performance Reference for IBM z Systems:
<https://www.ibm.com/servers/resourceLink/lib03060.nsf/pages/lspindex>

Help from IBM

IBM Support and downloads:

[ibm.com/support](https://www.ibm.com/support)

IBM Global Services:

[ibm.com/services](https://www.ibm.com/services)



SG24-8298-02

ISBN 0738457930

Printed in U.S.A.

Get connected

