

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

Graham Rawson



z Systems



Introduction

This IBM® Redpaper™ publication is a study of the performance of a Java Platform, Enterprise Edition (Java EE) application that is hosted by an IBM CICS® Transaction Server for z/OS® (CICS) V5.2 Liberty Java virtual machine (JVM) server that connects to an IBM DB2® for z/OS database server. The use of type 2 and type 4 Java Database Connectivity (JDBC) drivers supplied by DB2 for z/OS is described, and performance comparisons are made using processor (CPU) resource use and response time data.

CPU usage data includes measurement of offload eligibility using IBM System z® Integrated Information Processors (zIIPs).

The target audience for this IBM Redpaper publication is as follows:

- ▶ Systems Architects considering the performance characteristics of using either the type 2 or type 4 JDBC driver in a CICS Liberty environment.
- ▶ Capacity Planners and Performance Analysts trying to understand how to configure the JDBC drivers and measure the resource usage and response time characteristics of the JDBC drivers.

This paper contains a description of the Java EE application used in this performance study, and a description of the environment on which the performance measurements were made. The results from the performance measurements and an analysis of these results are presented.

This paper covers the following topics:

- ▶ A description of the Java EE application used, and a description of some minor changes made to extend the application functionality
- ▶ A description of the hardware and software comprising the measurement environment
- ▶ An overview of the JDBC driver types used in this study
- ▶ A description of database connection sharing performance optimizations used in this study
- ▶ An explanation of the configuration of a JDBC driver type for use in a CICS Liberty JVM server
- ▶ An explanation of the performance metrics used in this study
- ▶ Presentation and analysis of the results in terms of CPU cost, zIIP eligibility, and request response time

This paper is one in a series 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.

Disclaimer: All performance data contained in this paper was obtained in the specified operating environment and configurations, and should be considered an illustration. Performance characteristics of other operating environments might differ.

IBM does not represent, warrant, or guarantee that the same or similar results as those results reported in this paper will be achieved in a user's environment.

Executive summary

This paper uses the TradeLite Java EE benchmarking application, featured in numerous IBM WebSphere® Application Server studies, to investigate the comparative benefits of using either the type 2 (native) or type 4 (Java) JDBC driver with a CICS Liberty server.

The benefits studied are as follows:

- ▶ Total CPU cost. Both general processor and zIIP time.
- ▶ Non-zIIP-eligible time. CPU time that is not eligible to be offloaded to a zIIP.
- ▶ zIIP-eligible time. CPU time that could be offloaded to a zIIP.

Note: This zIIP-eligible work can run on a general processor if all the available zIIPs are busy running other work.

- ▶ Response time. As measured by the web client.

Considering the likely need in modern application design for global transactional scope, the most generally applicable comparison is between *type 2 optimized* and *type 4 Java Transaction API (JTA)*.

The type 2 JDBC driver uses the CICS DB2 attachment facility, and provides functions not available to the type 4 JDBC driver.

Figure 1 summarizes CPU resource use costs and response times for the key configurations investigated at a request rate of approximately 80,000 per minute.

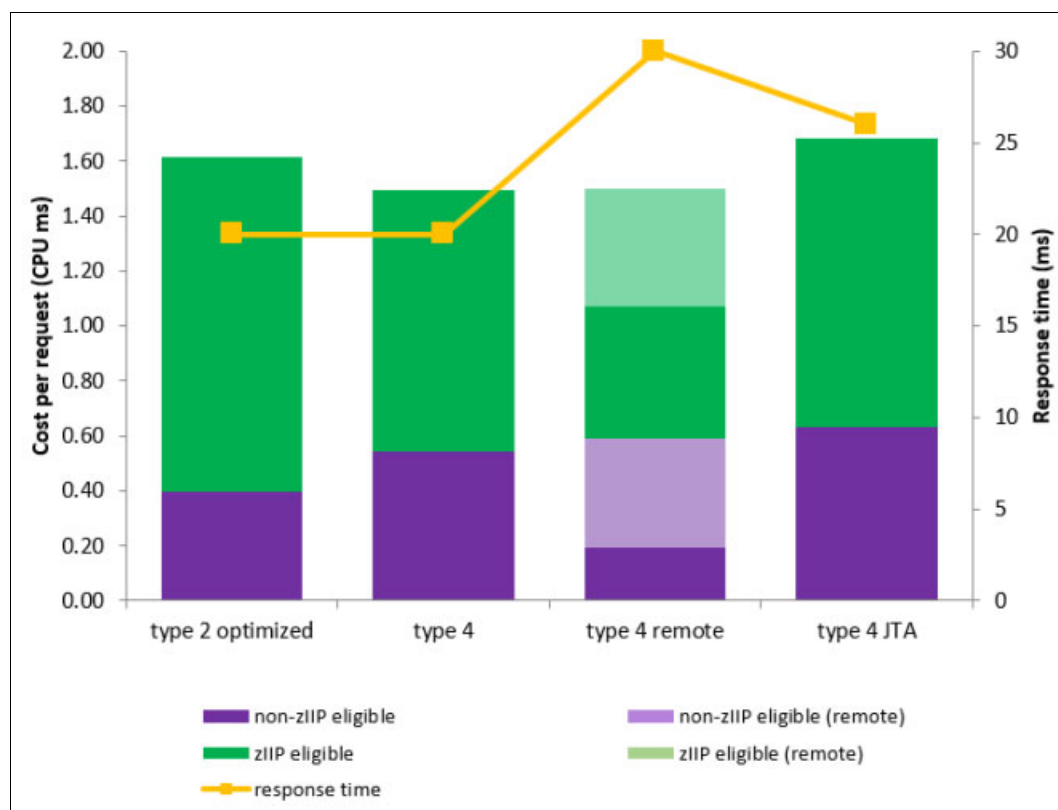


Figure 1 CPU and response time for key configurations

If minimizing total CPU use on the local logical partition (LPAR) is important, then the least expensive configuration is the type 4 JDBC driver using a remote database server. With this configuration, the cost of the database server running on the remote system is shown separately.

Response times might be longer using a type 4 remote connection, depending upon the speed and capacity of the intersystem network connection. Consider also that the remote system might not be capable of delivering the scalable service possible by running the database server on the local z/OS system.

If using a local database server and minimizing CPU use on the local LPAR is important, and zIIP processors are available, then the least expensive configuration is using a type 2 JDBC driver. This is the least expensive configuration because the type 2 JDBC driver has a higher zIIP offload capability than the type 4 JDBC driver. The type 2 JDBC driver also provides application optimizations that can reduce CPU usage. These optimizations are not available when using the type 4 JDBC driver.

If there are no zIIP processors available, then the least expensive configuration is using the type 4 JDBC driver. Note, however, that using a type 2 JDBC driver uses the CICS DB2 attachment facility. Therefore, it provides functions not available to the type 4 JDBC driver, such as monitoring, security integration, and debugging facilities.

If providing global transactional scope is important, then using the type 2 JDBC driver provides both the lowest non-zIIP use and lowest total CPU use compared to the alternative type 4 JDBC driver with JTA UserTransactions. The type 2 JDBC driver also delivers shorter response times.

Note, however, that the transaction scope using the type 2 JDBC driver only covers the resources managed by the CICS unit of work (UOW), such as file control, temporary storage (TS), multiregion operation (MRO), intersystem communication (ISC), DB2, or IBM MQ. It does not cover any resource managers coordinated using Java transactions through the facilities of the CICS Liberty transaction manager.

If response times are critical, then using the type 2 JDBC driver through the CICS DB2 attachment facility connected to a local database server is the key factor. The type 2 JDBC driver gives the shortest response time. The type 4 JDBC driver without JTA provides the same short response time, but without a global transactional scope.

The TradeLite application

The Java EE application chosen for this study is TradeLite. TradeLite is a version of the IBM DayTrader benchmark. TradeLite is a Java EE web application that uses servlets, JavaServer Pages (JSP), and JDBC. Unlike DayTrader, TradeLite does not use Enterprise JavaBeans (EJB) or the Java Message Service (JMS) application programming interface (API).

TradeLite has been used in a number of WebSphere Application Server performance studies. One study in particular is from the Impact 2012 conference presentation *WebSphere Application Server V8.5 Innovative Applications & Interactive Experience*¹. It is an ideal candidate for similar performance studies using the CICS Liberty JVM server.

A version of the TradeLite application (BlueTrader-E2E-IBM Bluemix™) is available for download from the following web page:

<https://github.com/acostry/BlueTrader-E2E-Bluemix/tree/master/src/com/ibm/icap/tradelite>

TradeLite does not use the Java class library for CICS (JCICS) API to access any CICS resources.

TradeLite uses dynamic Structured Query Language (SQL).

TradeLite is a simulated share trading application. The data used in this study consists of 500 trader accounts and 1000 company share quotations stored in a DB2 for z/OS database system as 6 tables and 11 indexes.

A web browser is used to access TradeLite functions.

The workload run is composed of a mix of TradeLite functions. On average, each web browser request makes 31 JDBC calls to the database system, resulting in between three and four database connection commit calls.

¹ WebSphere Application Server V8.5 Innovative Applications & Interactive Experience,
<http://www.slideshare.net/websphereindia/websphere-application-server-v85>

The test environment

This section describes the hardware and software configuration used to run the performance measurements. Most of the measurements are collected on a single IBM z13™ LPAR. A second LPAR is used in configurations where a remote database server is needed. This environment is shown schematically in Figure 2 on page 6.

Hardware configuration

The measurement environment is composed of LPARs of the same IBM z13 system. Each has dedicated processors and memory. The Transmission Control Protocol/Internet Protocol (TCP/IP) services on each LPAR are connected by a 10 gigabit (Gb) Open Systems Adapter (OSA) link.

Local system

The system on which the measured CICS Liberty JVM server and local database system runs consists of the following components:

- ▶ A z13 system 2964-7B3 LPAR (equivalent to a z13 system 2964-705)
- ▶ Three dedicated general-purpose central processors (CPs)
- ▶ Two dedicated zIIPs
- ▶ 16 gigabytes (GB) real storage

Remote database system

The system on which the remote database system runs consists of the following components:

- ▶ A z13 system 2964-7B3 LPAR (equivalent to a z13 model 2964-701)
- ▶ One dedicated CP
- ▶ 16 GB real storage

Software

Both systems in this study use the following software:

- ▶ z/OS V2R1
- ▶ CICS V5.2
- ▶ Java V7.1 SR2
- ▶ DB2 for z/OS 11.01.0

A DB2 for z/OS database server is used as the data source for the JDBC requests. Both local and remote database servers are used for this study. The local database server runs in the same LPAR as the CICS system. The remote database server runs on another LPAR in the same z/OS sysplex, connected with a 10 Gb OSA link.

System schematic

Figure 2 shows the relationship between the software components used for the three types of database connections measured:

- ▶ T2. Type 2 JDBC driver connection using a direct link between the CICS and local DB2 system by using the CICS DB2 attachment facility.
- ▶ T4. Local type 4 JDBC using an Internet Protocol network link between the CICS and DB2 systems. In this case, the network link is within the same TCP/IP host.
- ▶ T4 remote. Remote type 4 JDBC. In this case, the network link between the CICS and DB2 connects different TCP/IP hosts.

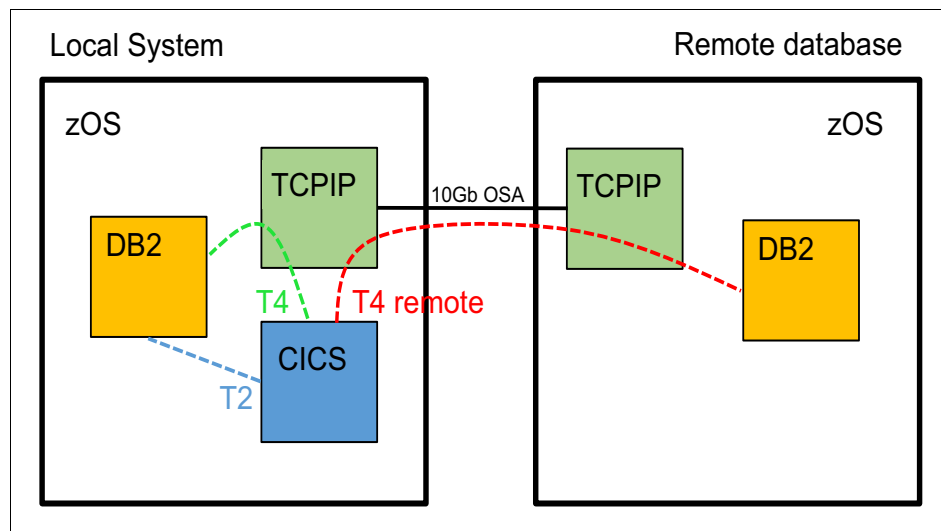


Figure 2 Test environment schematic

JDBC driver types

Java programs running in a CICS Liberty JVM server and using the JDBC API can choose one of two driver types to connect to a data source. These two types are as follows:

- ▶ The type 2 JDBC driver has a native (non-Java) component. When used in a CICS environment, the type 2 JDBC driver converts JDBC requests into their **EXEC SQL** equivalent. These converted requests use the CICS DB2 attachment facility to access the local DB2 system.
- ▶ The type 4 JDBC driver is pure Java, and connects directly to a database server using a network connection.

The major differences between type 2 and type 4 JDBC drivers are as follows:

- ▶ The type 2 driver connects to the database server using the CICS DB2 attachment facility. The type 4 connects using an Internet Protocol network connection.
- ▶ The type 2 driver includes some native code that is ineligible for offload to zIIP processors. The type 4 is written in pure Java, and is eligible for offload to zIIP processors.

- ▶ Although the type 4 driver is written in Java, only 60% of the DB2 for z/OS instructions running SQL requests are eligible for offload to a zIIP processor. For more details, review *SQL request workloads that use DRDA to access DB2 for z/OS over TCP/IP connections* in the IBM Knowledge Center at the following web address:

http://www.ibm.com/support/knowledgecenter/SSEPEK_11.0.0/com.ibm.db2z11.doc.pdf/src/tpc/db2z_ibmziip.dita

- ▶ The type 2 JDBC driver uses the CICS DB2 attachment facility. This facility provides a number of functions that are not available if using the type 4 JDBC driver, including the following functions:
 - CICS DB2 statistics data
 - CICS DB2 monitoring data
 - CICS execution diagnostic facility (EDF)
 - DB2 group attach support
 - Use of a CICS DB2ENTRY definition to set specific DB2 options, such as DB2 accounting options, primary and secondary authorization options, deadlock rollback support, thread limits, and setting task control block (TCB) priority

Configuring a JDBC driver type for a CICS Liberty JVM server

The CICS Liberty JVM server configuration is described in a series of elements in the `server.xml` configuration file. Type 2 and type 4 JDBC drivers require a different set of `featureManager`, `jdbcDriver`, and `dataSource` element definitions.

Type 2 JDBC driver `server.xml` definition

To enable type 2 JDBC driver connections, use the following elements:

- ▶ A `featureManager` element (Example 1):

Example 1 A `featureManager` element

```
<featureManager>
  <feature>cicsts:jdbc-1.0</feature>
</featureManager>
```

- ▶ A `jdbcDriver` element (Example 2):

Example 2 A `jdbcDriver` element

```
<cicsts_jdbcDriver id="defaultCICSJdbcDriver" libraryRef="defaultCICSDB2Library"/>
<library id="defaultCICSDB2Library">
  <fileset dir="/usr/lpp/db2v11/jdbc/classes" in-cludes="db2jcc4.jar
db2jcc_license_cisuz.jar"/>
  <fileset dir="/usr/lpp/db2v11/jdbc/lib" in-cludes="libdb2jcc2zos4_64.so"/>
</library>
```

- ▶ A `dataSource` element (Example 3):

Example 3 A `dataSource` element

```
<cicsts_dataSource id="defaultCICSDataSource" jndiName="jdbc/TradeDataSource"/>
```

Type 4 JDBC driver server.xml definition

To enable type 4 JDBC driver connections, use the following elements:

- ▶ A `featureManager` element (Example 4):

Example 4 A featureManager element

```
<featureManager>
  <feature>jdbc-4.0</feature>
</featureManager>
```

- ▶ A `dataSource` element (Example 5):

Example 5 A dataSource element

```
<dataSource jndiName="jdbc/TradeDataSource" type="javax.sql.XADataSource">
```

- ▶ A `jdbcDriver` element (Example 6):

Example 6 A jdbcDriver element

```
<jdbcDriver libraryRef="db2Lib"/>
<connectionManager agedTimeout="0" maxPoolSize="0"/>
<properties.db2.jcc databaseName="DSNV11P3" driverType="4"
password= "{xor}Lz4sLCgwLTs=" portNumber="41100" serverName=
"winmvs2a.hursley.ibm.com" user="WSADMIN"/>
</datasource>
```

- ▶ A `library` element (Example 7):

Example 7 A library element

```
<library id="db2Lib">
  <fileset dir="/usr/lpp/db2v11/jdbc/classes" in-cludes="db2jcc4.jar
db2jcc_license_cisuz.jar"/>
  <fileset dir="/usr/lpp/db2v11/jdbc/lib"
in-cludes="libdb2jcct2zos4_64.so"/>
</library>
```

Key elements are defined as follows:

- ▶ The `serverName` is the IP address for the system that hosts the data source. For this study, both local and remote hosts have been used.
- ▶ The `agedTimeout=0` setting prevents connection reuse.
- ▶ The `agedTimeout=-1` setting allows unlimited connection reuse.
- ▶ A `javax.sql.XADataSource` type of data source is enabled for connection pooling, and is also able to participate as a two-phase capable resource in transactions involving multiple resources.

JDBC driver connection reuse mechanisms

Creating and closing a connection between the JDBC application and a data source is an expensive operation. This cost can be reduced if the connection can be reused. Both types of JDBC drivers provide ways of reusing these connections without the need to change the application.

Protected threads with the type 2 JDBC driver

The type 2 JDBC driver converts JDBC requests into equivalent SQL calls. These SQL calls can use protected threads, which is a mechanism to reuse CICS-DB2 threads that reduces resource use when acquiring connections between CICS and DB2.

Protected threads are enabled with a CICS **DB2ENTRY** for the CICS Liberty transaction identifier, which is CJSA by default. The **PROTECTNUM** parameter specifies how many protected threads are to be made available to the transaction identifier. In addition, with the **REUSELIMIT** of 10000 for the **DB2CONNECTION** definition, a high level of DB2 thread reuse is achieved. This is shown in the extract of a CICS statistics report (Figure 3) that reports for 4256254 SQL calls. In this case, only 44 threads were created, and these were used 456425 times.

DB2ENTRY STATISTICS - REQUESTS										
DB2Entry Name	Call Count	Signon Count	Partial Signon	Commit Count	Abort Count	Single Phase	Thread Create	Thread Reuse	Thread Terms	Thread Waits/Overflows
TRADENT	4256254	0	0	0	0	456419	44	456425	44	0
TOTALS	4256254	0	0	0	0	456419	44	456425	44	0

Figure 3 CICS DB2ENTRY Statistics extract showing high thread reuse

Connection pooling with the type 4 JDBC driver

The type 4 JDBC driver provides a connection pooling mechanism, which is a framework for caching physical data source connections. With connection pooling, the data source connection can be serially reused by JDBC applications. Connection pooling is enabled by default for a CICS Liberty JVM server. It is enabled by setting the `agedTimeout` attribute of the `connectionManager` element of the `server.xml` file shown in “Type 4 JDBC driver server.xml definition” on page 8 to -1. Connection pooling can be disabled by setting the `connectionManager` element to 0.

Because connection pooling reuses connections, it also reduces the cost incurred in creating and discarding connection objects in Java. This connection pooling mechanism is not available when using the type 2 JDBC driver. Therefore, using the type 4 JDBC driver incurs lower Java resource use, most noticeably observed as lower garbage collection activity.

Performance measurement procedure

The same procedure was followed for each measurement, consisting of the following steps:

1. The CICS system was COLD started.
2. The DB2 tables and indexes were re-created.
3. The DB2 tables and indexes were populated with 500 user accounts and 1000 stock quotation records using the TradeLite application.
4. A simulator tool was used to simulate 200 web clients.
5. The simulator script was run at an initial rate of about 700 web requests per second for 10 minutes to stabilize the system.
6. Measurements were taken after this stabilization period by starting at a low request rate, and increasing the request rate at 5-minute intervals. The data shown in this study is for the final minute of each 5-minute period.
7. IBM z/OS Resource Measurement Facility™ (RMF™) and CICS statistics data was collected at 1-minute intervals.

Performance metrics

The Results section reports the following metrics:

- ▶ Requests per minute
- ▶ CPU utilization percent
- ▶ Response time

Requests per minute and CPU utilization percent (%) are extracted from RMF workload activity data. For example, the extract of an RMF report shown in Figure 4 is for a single CICS system. The transaction count shows that 88584 transactions completed in the 1-minute measurement interval. The APPL% section shows that 31.66% of general processor time was used (represented by the CP data). Of that time spent on a general processor, 6.76% was eligible to use a zIIP processor (represented by the IIPCP data).

The zIIP-eligible work runs on a general processor if all of the available zIIP processors are busy. 75.53% of zIIP time was used (represented by the IIP data).

REPORT BY: POLICY=POLICY				REPORT CLASS=CICS2A20			
				DESCRIPTION =			
-TRANSACTIONS-	TRANS-TIME	HHH.MM.SS.TTT	--DASD I/O--	---SERVICE---	SERVICE TIME	---APPL %---	
AVG 1.00	ACTUAL	19	SSCHRT 0.0	IOC 66510	CPU 64.307	CP 31.66	
MPL 1.00	EXECUTION	0	RESP 0.0	CPU 4876K	SRB 0.007	AAPCP 0.00	
ENDED 88584	QUEUED	0	CONN 0.0	MS0 0	RCT 0.000	IIPCP 6.76	
END/S 1476.42	R/S AFFIN	0	DISC 0.0	SRB 528	IIT 0.000		
HSWAPS 0	INELIGIBLE	0	Q+PEND 0.0	TOT 4943K	HST 0.000	AAP N/A	
EXCTD 0	CONVERSION	0	IOSQ 0.0	/SEC 82391	AAP N/A	IIP 75.53	
AVG ENC 0.00	STD DEV	14			IIP 45.320		
REM ENC 0.00				ABSRPTN 82K			
MS ENC 0.00				TRX SERV 82K			

Figure 4 RMF Workload Activity extract

CPU utilization percent (%) is calculated as $CP + IIP$. Non-zIIP eligible time represents the time that must be run on a general processor, and is calculated as $CP - (IIP + IIPCP)$. Note that $IIP + IIPCP$ is the zIIP-eligible time, and is the sum of actual time spent on a zIIP and the time spent on a general processor that could have been run on a zIIP, but none were available.

Minimizing general processor use is a performance objective for many CICS customers, because this approach can also reduce software costs.

RMF reports performance data for all of the subsystems contributing to this workload. These subsystems are as follows:

- ▶ CICS
- ▶ TCP/IP, for network traffic
- ▶ DB2
- ▶ DDF, the distributed data facility component of DB2 that runs requests using the type 4 JDBC driver

Response times are extracted from the simulator tool reports, as illustrated in Figure 5.

```
IWL0058I Time limit of 300 seconds has been reached. Shutting down.
IWL0038I Run time = 00:05:04
IWL0007I Clients completed = 0/200
IWL0059I Page elements = 397189
IWL0060I Page element throughput = 1303.760 /s
IWL0059I Transactions = 328969
IWL0060I Transaction throughput = 1079.830 /s
IWL0059I Network I/O errors = 0
IWL0059I Web server errors = 0
IWL0059I Num of pages retrieved = 397189
IWL0060I Page throughput = 1303.760 /s
IWL0060I HTTP data read = 3305.795 MB
IWL0060I HTTP data written = 168.631 MB
IWL0060I HTTP avg. page element response time = 0.022
```

Figure 5 Simulation tool report extract

For more detail about performance measurement procedures, see the IBM Redpaper publication *IBM CICS Performance Series: Effective Monitoring for CICS Performance Benchmarks*, REDP-5170.

Considering subcapacity general-purpose processors

The IBM z13 system 2964-7B3 used for this study consists of both general purpose and offload processors running at the same uncapped capacity. It is possible to reduce the licensing charges associated with such a system by restricting the capacity of the general-purpose processors. This restriction does not apply to the offload processors.

There are also models of the IBM z13 system available with subcapacity settings for the general processors (models 2964-4xx, 5xx and 6xx). These systems consist of general processors that run slower than those in 7xx models. These systems have a reduced Processor Capacity Index (PCI) compared to the 7xx models. The zIIPs in such systems run at full speed.

Subcapacity configurations are generally beneficial to workloads with high zIIP-eligible content, such as the Java EE application investigated in this paper. Capacity planning for such environments is discussed in the IBM developerWorks® article *New zIIP Capacity Planning Presentation* by Martin Packer, which is available at the following web address:

https://www.ibm.com/developerworks/community/blogs/MartinPacker/entry/new_ziip_capacity_planning_presentation?lang=en

Configurations measured

This section covers the configurations used for the results reported in this study. In each of these configurations, the TradeLite application is defined in the CICS Liberty server.xml configuration file (Example 8).

Example 8 CICS Liberty server.xml configuration file

```
<application id="tradeLite" location= "/u/rawson/bundles/tradeLite.war"
    name="tradeLite" type="war">
  <application-bnd>
    <security-role name="cicsAllAuthenticated">
      <special-subject type="ALL_AUTHENTICATED_USERS"/>
    </security-role>
  </application-bnd>
</application>
```

T2: Type 2 JDBC driver with no optimization

In this configuration, a CICS **DB2ENTRY** resource definition was used that specified a **TRANSID** of CJSA with no protected threads by setting **PROTECTNUM(0)**. In addition, a CICS **DB2CONN** resource definition specified **THREADWAIT(YES)**.

In this configuration, every connection to the data source issued by the TradeLite application is required to establish and release a connection after use, if there were no queued thread requests.

T4: Type 4 JDBC driver with no optimization

In this configuration, there are no CICS DB2 definitions required. The connection between the data source and TradeLite application is defined in the server.xml configuration file as described in “Type 4 JDBC driver server.xml definition” on page 8, setting **agedTimeout=0**.

In this configuration, every connection to the data source issued by the TradeLite application is required to establish and release a connection after use, if there were no queued thread requests.

T2 reuse: Type 2 JDBC driver with protected threads

In this configuration, connection reuse is enabled as described in “Protected threads with the type 2 JDBC driver” on page 9.

T4 reuse: Type 4 JDBC driver with connection pooling

In this configuration, connection reuse is enabled as described in “Connection pooling with the type 4 JDBC driver” on page 9.

T4 reuse remote: Type 4 JDBC driver connected to remote data source

In this configuration, the data source definition described in “Type 4 JDBC driver server.xml definition” on page 8 is modified by changing the **serverName** to a data source on another LPAR.

T2 OC: Type 2 JDBC using optimized commit calls

Analysis of the CICS statistics data showed that when using the type 2 JDBC driver, a large number of single-phase commit requests were driven per web client request, as shown in Figure 6.

DB2ENTRY STATISTICS - REQUESTS										
DB2Entry Name	Call Count	Signon Count	Partial Signon	Commit Count	Abort Count	Single Phase	Thread Create	Thread Reuse	Thread Terms	Thread Waits/Overflows
TRADENT	2455733	0	0	0	6	262978	26	262982	26	0
TOTALS	2455733	0	0	0	6	262978	26	262982	26	0

Figure 6 CICS DB2 statistics extract showing high Single Phase count

On examination of the TradeLite application source code, it was found that after every call to the data source a connection commit was issued. This is common practice for Java EE applications that do not use a global XA transaction with the type 4 JDBC driver. In this case, the requests to the JDBC data source are local transactions. The changes made within the scope of the local transaction must be committed after every JDBC call with a **connection.commit()** call.

A benefit to using the type 2 JDBC driver is that commit processing with DB2 is run automatically when a CICS sync point is run. A CICS sync point is always run at the completion of a CICS transaction. The type 4 JDBC driver does not cause commit processing in DB2 when a CICS sync point is run.

Using the type 2 JDBC driver allows an optimization of the TradeLite application, where the **connection.commit()** calls can be removed, and the necessary commit processing is triggered by the CICS sync point at completion of the CICS transaction running the servlet request. This optimized version of the TradeLite application will not run successfully using the type 4 JDBC driver, because the database updates will not be committed.

T4 JTA: Type 4 JDBC using Java Transaction API calls

It is possible to remove the need for the multiple **connection.commit()** calls in the TradeLite application by creating a global transaction. This is done by using a JTA transaction to include all of the JDBC calls within the web client request.

A JTA UserTransaction instance can be used to provide a global XA two-phase commit transaction, extending the scope of a transaction across multiple calls to multiple resource managers.

In this configuration, a modification is made to the TradeLite application to use JTA to coordinate transactional updates. This modification is made by creating a UserTransaction object to encapsulate and coordinate updates made with JDBC calls by the TradeLite application.

Documentation for the code changes necessary is available in the CICS V5.2 IBM Knowledge Center topic *Java Transaction API* at the following web address:

http://www.ibm.com/support/knowledgecenter/api/content/SSGMCP_5.2.0/com.ibm.cics.ts.java.doc/topics/dfhpj2_jta.html?locale=en

When using JTA with the type 4 JDBC driver, the CICS Liberty transaction manager is the transaction coordinator.

T2 JTA: Type 2 JDBC using Java Transaction API calls

In this configuration, the same version of the TradeLite application is used as for the *T4 JTA* configuration. There is no functional need to use a JTA `UserTransaction` when using the type 2 JDBC driver with the TradeLite application because, unlike the type 4 JDBC driver, all of the DB2 updates are committed by a CICS sync point.

When using JTA with the type 2 JDBC driver, the CICS Liberty transaction manager is the transaction coordinator.

Transaction scope in the TradeLite application

In its original form, the TradeLite application uses local transactions to control the integrity of the JDBC updates that it makes. The TradeLite application uses explicitly coded `connection.commit()` calls after each update to instruct the database server to commit the update.

A global transaction is needed if the scope of the updates of an application extends to multiple database updates as a single consistent change, or if resources controlled by another resource manager are updated.

JTA `UserTransactions` can provide global transactional scope when using the type 4 JDBC driver. The TradeLite application needs modification to use `UserTransactions`. Using the type 2 JDBC driver does not need such changes, because it benefits from global transactional scope provided through use of the CICS DB2 interface.

Although using local transactions with the type 4 JDBC driver is relatively inexpensive, local transactions can only be used with resources owned by a single resource manager. Modern applications, for example those using the Java EE Connector Architecture (JCA), are much more likely to require coordination of updates between multiple resource managers as provided by global transactions.

Considering the likely need for global transactional scope, the most generally applicable comparison with TradeLite is to compare the *T2 OC* configuration with the *T4 JTA* configuration.

Results

This section contains a number of comparisons of the performance characteristics of JDBC drivers. CPU utilization in these comparisons is the total of general processor and offload processor time for all of the subsystems contributing to the workload, as described in “Performance metrics” on page 10. Note that the local system provided a maximum CPU utilization of 500%, provided by three CPs and two zIIPs.

Reusing connections

Figure 7 shows a comparison of the CPU cost of running the TradeLite application at different requests rates, with and without reusing connections.

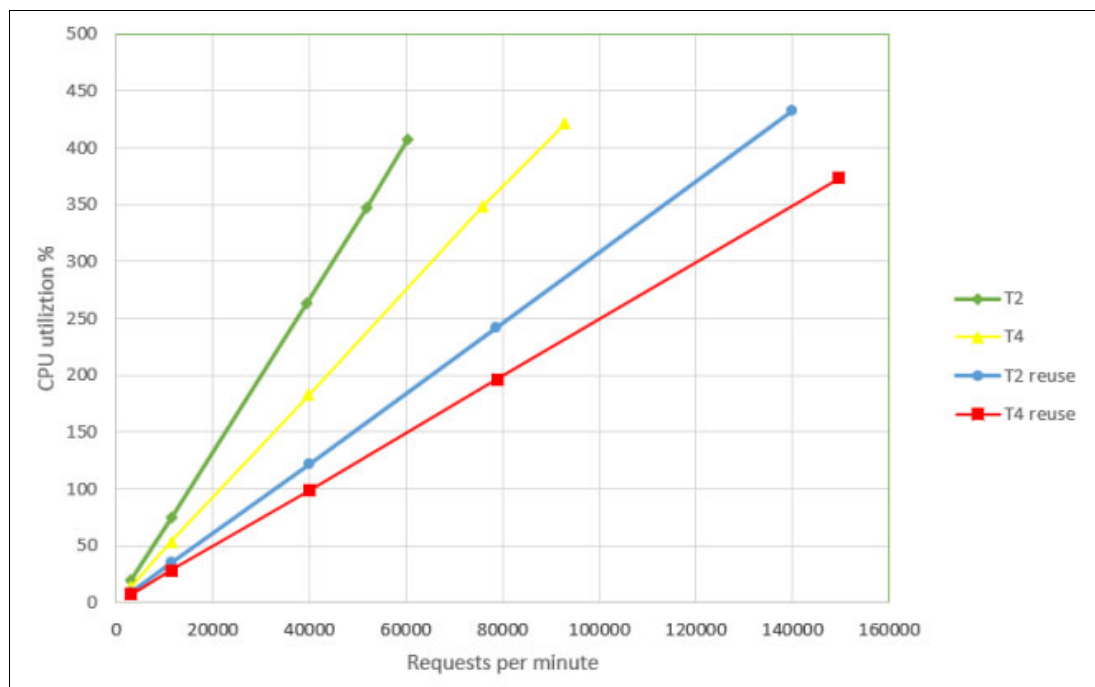


Figure 7 CPU utilization for type 2 and type 4 JDBC drivers, with and without reusable connections

Observations

Comparing the T2 and T4 configurations that do not reuse connections, using the type 4 JDBC driver provided both higher maximum throughput and lower cost per request compared to the type 2 JDBC driver. Code sampling profiles of these workloads revealed that the significant difference is due to time spent by the type 2 JDBC driver creating `com.ibm.db2.jcc.t2zos.T2zosConnection` objects. Fewer resources are required to create comparable `com.ibm.db2.jcc.am.Connection` objects using the type 4 JDBC driver.

Comparing both reusable configurations, the type 4 JDBC driver provides lower CPU cost per request. The lower CPU cost per request also enabled the type 4 JDBC driver to deliver a higher maximum throughput rate. Both type 2 and type 4 scale well, delivering the same cost per request with increasing request rates. The type 2 JDBC driver does not cache `com.ibm.db2.jcc.t2zos.T2zosConnection` objects. The type 4 JDBC driver does provide a caching mechanism for `com.ibm.db2.jcc.am.Connection` objects, and it is caching that provides a significant advantage to using a type 4 JDBC driver over the type 2.

Using a remote data source

Figure 8 shows the cost of using the type 4 JDBC driver to access a remote data source, and compares it to using a data source on the local system. Both the costs incurred by the *T4 remote* configuration only on the local system, and the total costs on both local and remote database systems, are shown.

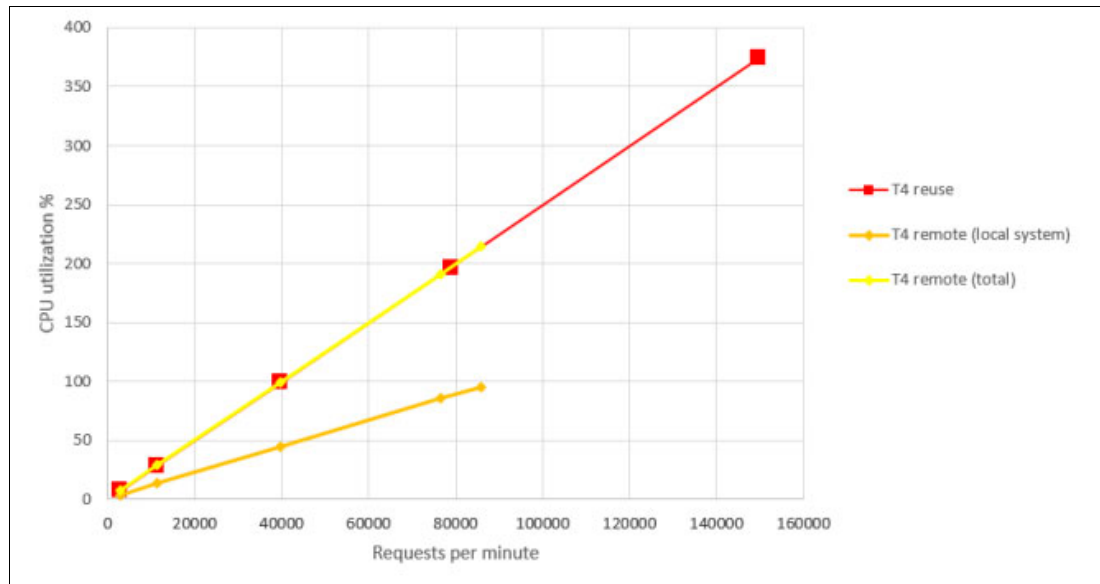


Figure 8 CPU utilization for type 4 JDBC drivers comparing local and remote connection

Observations

Using a connection to either data source on either the local system or a remote system scaled well to use the available z/OS system resources. The *T4 remote* configuration uses less CPU on the local system, but the total CPU cost, including that on the remote system, is comparable to that incurred if the data source is on the local system.

The maximum throughput achievable for the remote configuration is lower, because the remote data source runs on a less powerful system. In this study, the remote system is a single-processor LPAR, comparable to a z13 model 2964-701. However, the data source could be hosted on a completely different hardware and software platform (for example, DB2 running on an IBM AIX® server running on IBM Power Systems™).

Using the optimized commits version of TradeLite

Figure 9 shows the CPU cost of running an optimized version of the TradeLite application at different requests rates, compared to the original version of the TradeLite application that includes `connection.commit()` calls.

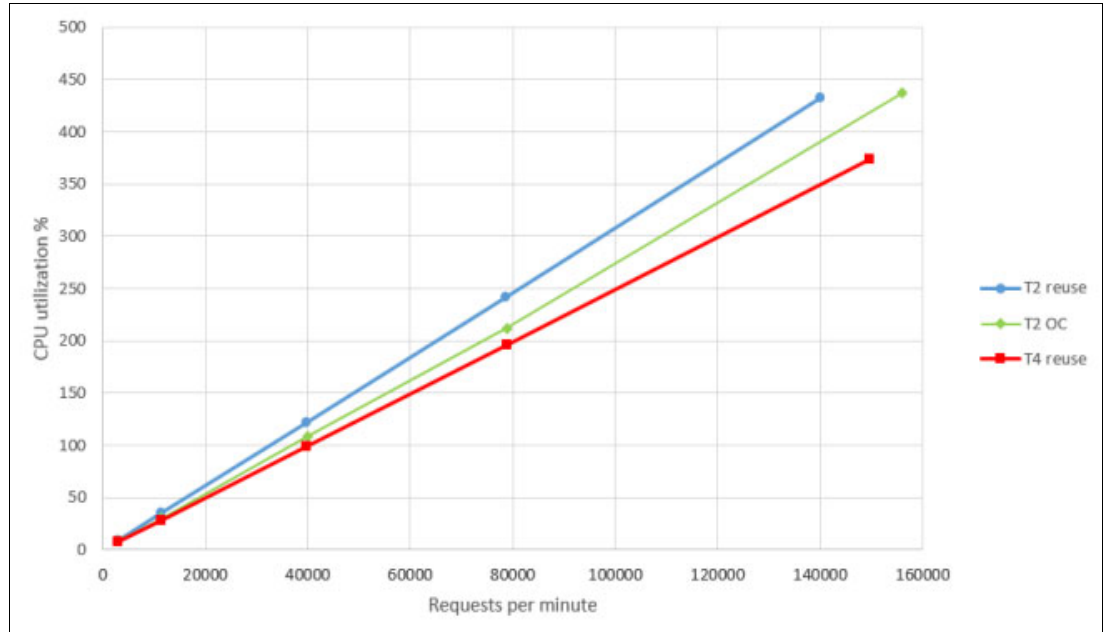


Figure 9 CPU utilization for optimized commits and original versions of TradeLite

Observations

When the TradeLite application is modified to remove the connection commit calls, the total CPU cost per request is reduced, and the maximum web request rate that can be achieved is increased. These results are as compared to using the unmodified version of the application with the type 2 JDBC driver.

However, the type 4 JDBC driver still achieves lower total CPU cost per web request, but it is only providing local transaction scope for the multiple JDBC calls contained in the web requests. The type 2 JDBC driver provides global transaction scope.

Using Java Transaction API

Figure 10 compares the CPU cost of running configurations that provide global transaction scope.

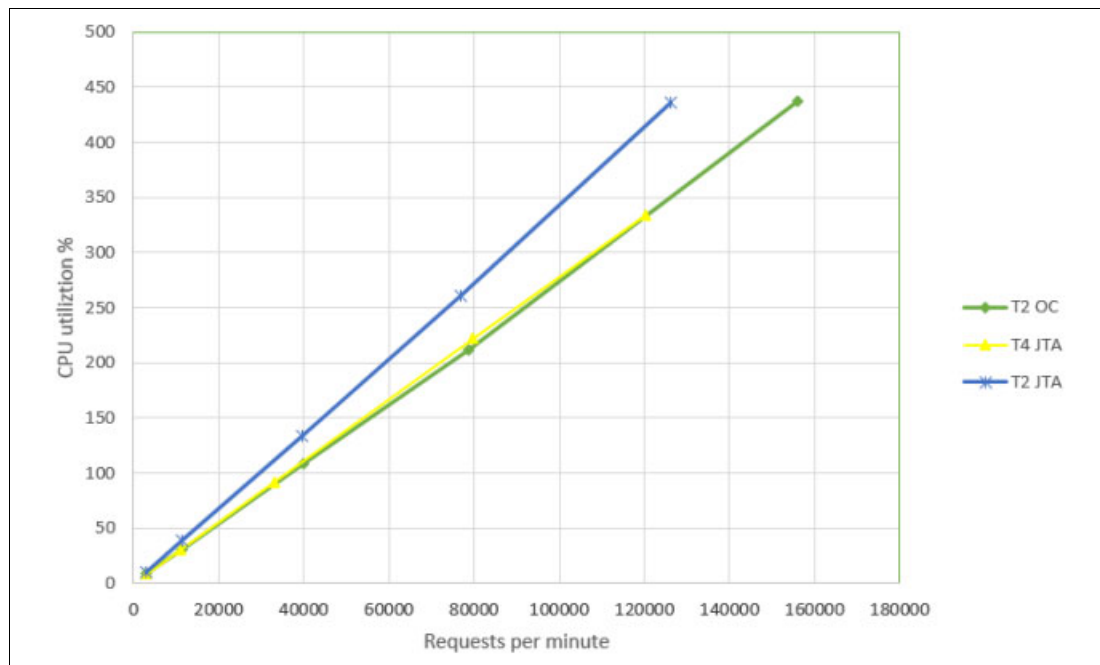


Figure 10 CPU utilization plotted against request rate for global transaction scope configurations

Observations

All three configurations provide global transactional scope, and so have altered the management of database updates compared to those made by the original version of the TradeLite application.

For the *T2 OC* case, the TradeLite application has been modified to remove the immediate connection commit calls, and commit processing with DB2 is run automatically when a CICS sync point is taken.

For the *T2 JTA* and *T4 JTA* cases, the TradeLite application has been modified to remove the immediate commit calls, and the database updates are contained in a JTA UserTransaction.

When using the type 2 JDBC driver with JTA UserTransactions, the coordinator role is performed by the Liberty component of CICS. The type 2 JDBC driver is optimized to use single-phase commit protocol when a DB2 database server is the only resource manager used by the UserTransaction.

When using the type 4 JDBC driver with JTA UserTransactions, the coordinator role is performed by the Liberty component of CICS, using a two-phase commit protocol involving both the CICS Recovery Manager and the DB2 database server. This approach requires that the CICS Recovery Manager writes link information to the CICS system log for every JTA UserTransaction. This additional logging activity reduces maximum throughput achievable for the *T4 JTA* case. This logging activity also affects response times, as shown in Figure 11 on page 19.

Additional CICS sync points caused by JTA

When a JTA UserTransaction is started, CICS issues a **SYNCPOINT** command to commit any existing recoverable updates, and begins a new Unit of Work (UOW). The scope of this UOW continues until the JTA UserTransaction is committed. Committing a JTA UserTransaction requires CICS to issue another sync point, and another UOW is created.

In this manner, the modified version of the TradeLite application that uses a JTA UserTransaction causes five sync points to be taken, compared to one sync point taken by the unmodified version of TradeLite.

Response times

Figure 11 shows response times for the variety of configurations measured.

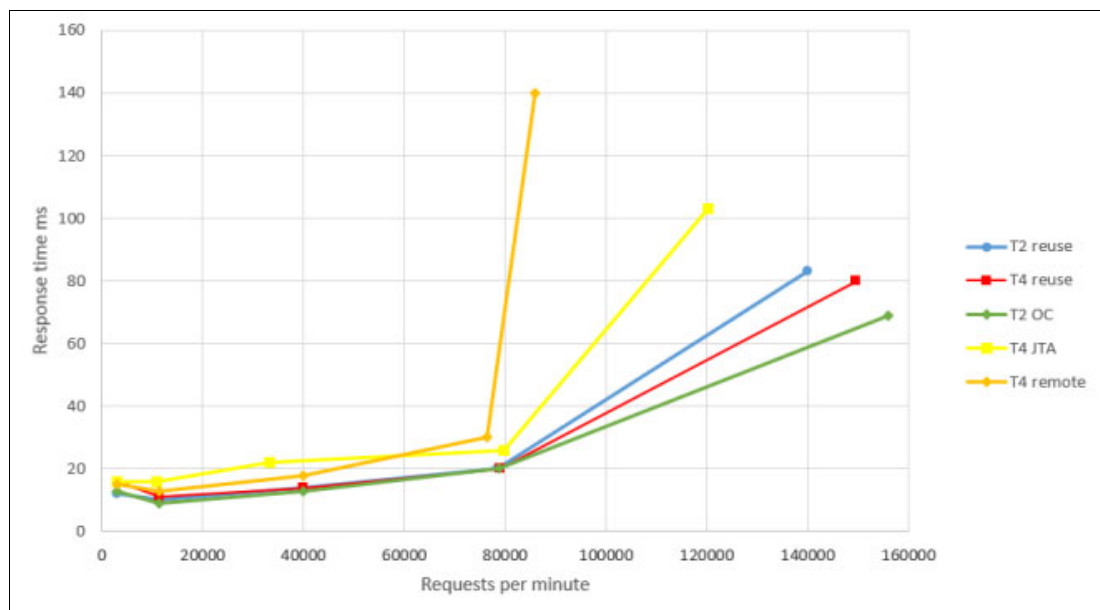


Figure 11 Response times with increasing request rates

Observations

In most configurations, response times are consistently low. Response time increases when a point of constraint is reached.

At the highest request rates, the number and type of commit requests become the constraining factors. *T2 OC* performs the smallest number of single-phase commits per web request, and achieves the shortest response times. *T2 reuse* and *T4 reuse* perform many more single-phase commit calls per web request than the *T2 OC*, with correspondingly longer response times.

The *T4 JTA* configuration shows longer response times because a two-phase commit process is being used, which causes extra CICS system logging. Writing this additional logging data adds a minor delay.

The *T4 remote* configuration is constrained by the responses from the remote DB2 system and the capacity of the network link used. For the environment used in this study, the capacity of this link was around 80,000 requests per minute.

Web request cost by subsystem and zIIP usage

The figures in this section show CPU cost per web request at a request rate of approximately 80,000 per minute for the *T2 reuse*, *T2 OC*, *T2 JTA*, *T4 reuse*, *T4 JTA*, and *T4 remote* configurations. Each figure shows the same set of results, but the results are categorized and displayed differently to show use by zIIP eligibility and subsystem.

Figure 12 shows CPU cost per web request categorized by zIIP and non-zIIP eligibility. It shows that the *T4 reuse* configuration has both the lowest total CPU and lowest non-zIIP-eligible usage.

When using global transaction scope, Figure 12 shows that *T2 OC* has a lower total CPU and lower non-zIIP-eligible usage than the other configurations providing global transaction scope, which are *T2 JTA* and *T4 JTA*.

Using the remote JDBC configuration, *T4 remote*, has the lowest zIIP and total CPU on the local system. However, when including the cost incurred on the remote system, the overall cost is equivalent to using a local database server, as shown by *T4 reuse*.

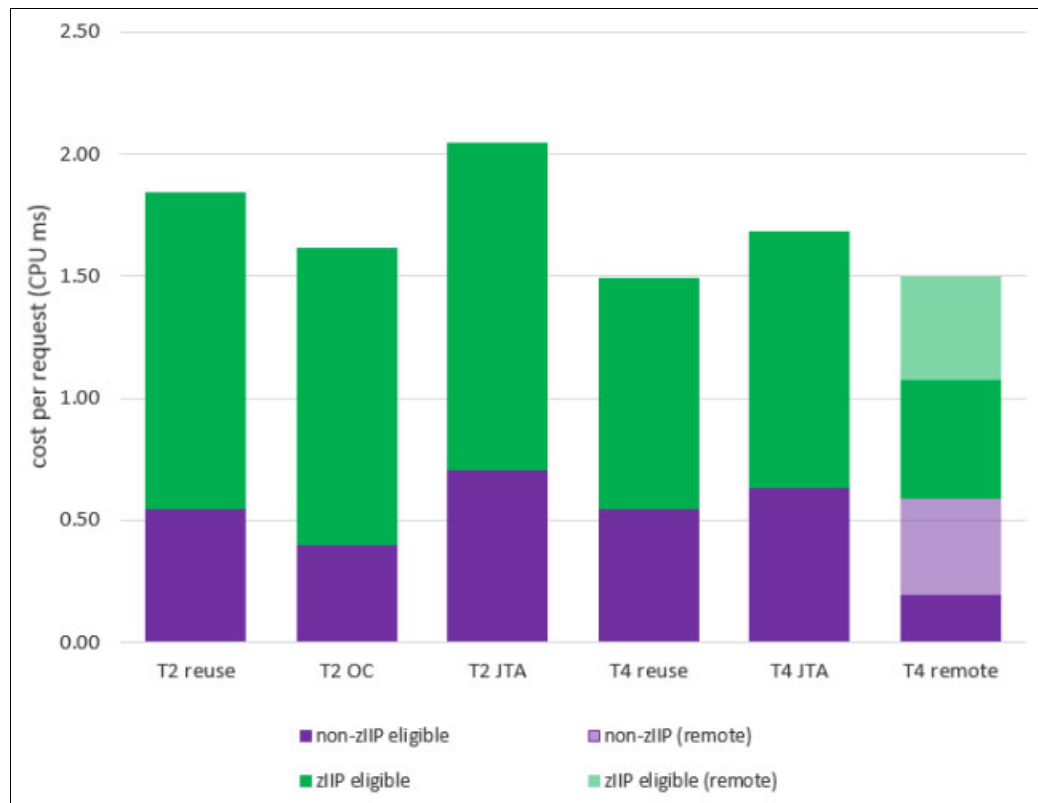


Figure 12 CPU cost per web request categorized by zIIP eligibility

Figure 13 shows the same data as Figure 12 on page 20, but has additional categorization by subsystem. Note that T4 configurations comprise substantial use of DB2 and its associated DDF subsystem. The contribution to CPU cost of the network component (TCP/IP) is negligible in all cases.



Figure 13 CPU cost per web request categorized by subsystem

Figure 14 shows the same data as Figure 12 on page 20 and Figure 13 on page 21, but subsystem contributions are further categorized by zIIP non-eligibility. It shows how the CPU used by the DDF subsystem for the T4 configurations is only partly (approximately 60%) zIIP-eligible. As described in the DB2 for z/OS 11.0.0 IBM Knowledge Center topic *IBM System z Integrated Information Processors*², up to 60% of the DB2 for z/OS instructions running SQL requests to access DB2 over a TCP/IP connection are eligible to run on a zIIP.

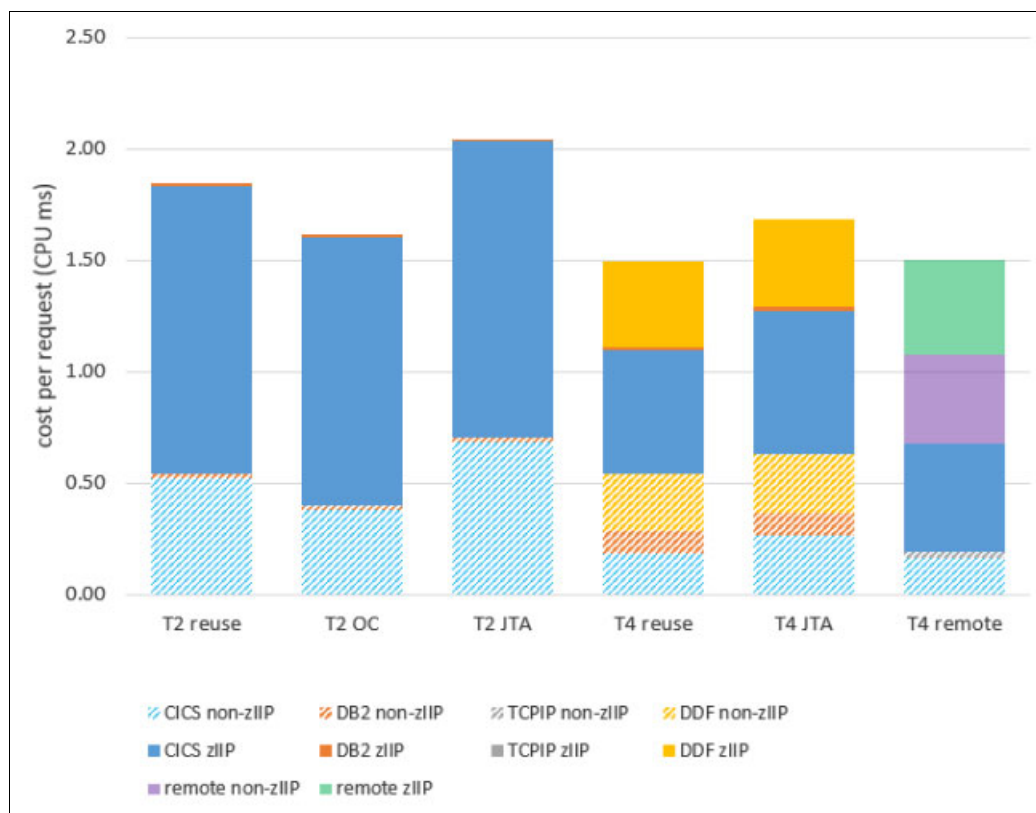


Figure 14 CPU cost per web request categorized by subsystem and zIIP eligibility

Observations

For global transaction scope, using the type 2 JDBC driver provides a lower total CPU and lower non-zIIP eligible usage compared to a type 4 JDBC driver using JTA UserTransactions.

For local transaction scope, each JDBC update must be explicitly committed. In this case, using the type 4 JDBC driver provides a lower total CPU and lower non-zIIP eligible usage compared to a type 2 JDBC driver.

Although the DDF subsystem is supporting SQL requests made over a TCP/IP connection by Java code, only 60% of the z/OS instructions running such SQL calls are eligible for zIIP usage.

² http://www-01.ibm.com/support/knowledgecenter/SSEPEK_11.0.0/com.ibm.db2z11.doc.perf/src/tpc/db2z_ibm_ziip.dita

Conclusion

The TradeLite application only makes changes to recoverable resources using JDBC calls. No other resource manager is used, so single-phase commit is all that is required in most configurations. A more complex application involving multiple resource managers needs a global transaction scope to coordinate updates distributed between these multiple resource managers. The addition of JTA UserTransactions provides a global transaction scope with the type 4 JDBC driver.

For the global transaction scope configurations used in this study, it was found that the type 2 JDBC driver delivered the best performance characteristics, in terms of total CPU cost, zIIP usage, and web request response times.

If a local transaction scope is all that is required, so that each update is committed immediately, it was found that the type 4 JDBC driver delivered the best performance characteristics.

Both the type 2 and type 4 JDBC drivers scale well, maintaining the same cost per web request with increasing request rates.

Response times for both types of JDBC driver remain similarly low for request rates up to about 1300 per second (about 80,000 per minute). The *T4 JTA* configuration shows a slightly higher response time compared to the other configurations. This result is because using the type 4 JDBC driver with JTA UserTransactions uses a two-phase commit protocol that writes more recovery data to the CICS systems log than the single-phase commit process used with the type 2 JDBC driver. This additional logging adds to the response time.

The type 4 JDBC driver is written in Java, but only 60% of the SQL processing is eligible for zIIP offload. As shown in Figure 12 on page 20, a greater proportion of the total CPU use of the type 4 JDBC driver configurations are zIIP-ineligible compared to using the type 2 JDBC driver, even though the type 4 is written in Java and the type 2 is written in native code.

There is little resource use incurred connecting to a remote database server compared to connecting to a database server running locally in the same LPAR. This result is shown in Figure 8 on page 16, where total costs of *T4 remote* closely match the *T4 reuse* CPU costs.

Using the *T4 remote* configuration reduces the CPU costs on the local system by running a database server on a remote system. *T4 remote* produces longer response times compared to using a local database server, as expected from delays introduced by using a cross-system network link.

The type 2 JDBC driver provides some functional advantages. Because the type 2 JDBC driver uses the CICS DB2 attachment facility, an application using the type 2 JDBC driver can use the tuning and diagnostic facilities provided by the facility, such as CICS DB2 statistics and the CICS EDF for CICS DB2. Using the type 2 JDBC driver also provides the ability for specific transactions to reserve threads or set specific DB2 configurations through use of a CICS **DB2ENTRY**. Such facilities are not available when using the type 4 JDBC driver.

Authors

This paper was produced by a CICS performance specialist working with the International Technical Support Organization (ITSO), Poughkeepsie Center.

Graham Rawson is a CICS Performance Analyst in the CICS Development group based at the IBM Laboratory in Hursley, England. Graham has worked for IBM for 30 years in various roles supporting CICS Transaction Server and IBM Java technology. Graham holds a Bachelor of Science (Hons) degree in Chemistry from the University of East Anglia (Norwich, England), and a Certificate in Software Engineering from the University of Oxford (England).

The project that produced this publication was managed by **Marcela Adan**, IBM Redbooks® Project Leader - ITSO, Raleigh Center.

Thanks to the following people for their contributions to this project:

Ivan Hargreaves
IBM United Kingdom

David Roberts
IBM United Kingdom

John Tilling
IBM United Kingdom

Phil Wakelin
IBM United Kingdom

Lindamay Patterson
IBM Sales & Distribution, Digital Sales

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 two - six weeks in length, and you can participate either in person or as a remote resident working from your home base.

Learn more about the residency program, browse the residency index, and apply online:

ibm.com/redbooks/residencies.html

Stay connected to IBM Redbooks

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

Notices

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

IBM may not offer the products, services, or features described in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

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

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

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law: INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

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

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

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

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

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

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

COPYRIGHT LICENSE:

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

© Copyright International Business Machines Corporation 2015. All rights reserved.

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

Send us your comments in one of the following ways:


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



Trademarks

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

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

AIX®	IBM z13™	RMF™
Bluemix™	Power Systems™	System z®
CICS®	Redbooks®	WebSphere®
DB2®	Redpaper™	z/OS®
developerWorks®	Redbooks (logo)  ®	z13™
IBM®	Resource Measurement Facility™	

The following terms are trademarks of other companies:

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

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



REDP-5208-00

ISBN 0738454389

Printed in U.S.A.

Get connected

