

Technical Report

IBM® Hursley, Hursley Park, UK.

Java™ Virtual Machine Choices with CICS® Transaction Server for z/OS® Version 2 Release 3

**By Mike Brooks, Senior IT Specialist,
CICS Development, IBM Hursley, U.K.**

Introduction

Java™ has rapidly grown in popularity throughout the Information Technology (IT) industry, and for many organizations is now their programming language of choice. **CICS®** has extended the support for Java based workloads over a number of releases, in response to the uptake of the Java programming model. **CICS Transaction Server for z/OS® Version 2, Release 3** (CICS TS 2.3) adds further support for Java. This includes the provision of greater controls for the runtime environment used by Java application programs, Enterprise Java Beans™ (EJBs), and distributed **IIOP** applications.

This paper outlines the improvements to Java support in CICS TS 2.3, and explains how individual CICS regions can be configured to match the needs of a particular Java workload. The technical background to CICS Java support is extensively documented in the CICS Information Center. This paper emphasizes the information that need to be considered when choosing the correct configuration of CICS to support specific Java workloads.

Overview

CICS TS 2.3 makes use of the **IBM® Development Kit for z/OS, Java 2 Technology Edition** (SDK), to provide the necessary Java runtime environment. Version 1.4 of the SDK extends the support for long lived Java Virtual Machines (JVMs) by providing a new execution mode with a simplified storage heap structure. It also permits Java classes to be cached and shared by a number of JVMs. CICS TS 2.3 provides infrastructure to allow both of these features to be used. In addition, CICS provides improvements to Java resource management by extending its storage protection mechanism to Java programs, and by refining the selection process it uses to match Java requests to the JVMs that run under its control.

JVM Persistence and Reuse

JVM reuse is not a new concept with CICS TS 2.3. The SDK provides specific features which allow a JVM to be retained once an application has finished running in it, so that another application can then reuse the JVM at a later point in time. CICS includes the necessary control mechanisms to take advantage of this feature, and by doing so permits a number of separate CICS transactions to run consecutively using the same JVM.

The SDK provides a split storage heap structure and a heap reset mechanism which can be used to prevent the actions of one CICS transaction from interfering with any another that run later in the same JVM. This imposes some additional processing costs onto each transaction. It has been recognized that Java programs can include routines which tidy up the storage areas that they use,

making them suitable to run in a persistent JVM without the need of a split heap and the operations which reset it.

Task Control Blocks used by CICS workloads

CICS has a small number of Task Control Blocks (TCB) which it uses when running application programs. Of these, the Quasi Reentrant (QR) TCB is used to single thread the entire workload. At any point in time there may be a number of transactions running in a CICS region. Only one of these will have use of the QR TCB at any time. If that application needs to pause for some reason, for example, when writing to a data set or waiting for a resource lock, it becomes suspended by CICS. This allows another application to gain control of the QR TCB and carry out some work. This permits CICS to serialize the execution of all the applications it has concurrently running, preventing the actions of one from interfering with those of any others.

This mechanism was developed for applications written in fully compiled languages such as COBOL, PL/1, or C. However, the introduction of support for Java programs presented some additional challenges which CICS has had to address.

Pure Java requires an execution environment provided by a JVM written for the operating system where it runs. On z/OS, a JVM can sometimes issue blocking calls, such as MVS waits, which cause the TCB used by the JVM to go into a wait state. If a JVM was started on the QR TCB then blocking calls would cause the entire CICS workload to pause. To address this issue, a new type of TCB, known as the J8 TCB, was introduced. CICS uses one of these for each JVM it starts. As we will see later in this paper, with CICS TS 2.3 there is now a J9 TCB as well.

The JVM that is used by CICS supports multithreaded applications. However only one application can run in a JVM at any one time. More than one TCB can be used, when a single CICS region has to support multiple concurrent Java applications. In this way, a CICS region can have a single pool of J8 and J9 TCBs to support its Java workloads.

All new CICS transactions start on the QR TCB. If the transaction makes use of a Java program, then CICS switches to either a J8 or a J9 TCB and runs the program under the control of a JVM there. When the program terminates, or if it needs to access a CICS managed resource, then CICS switches control back to the QR TCB for that piece of processing. In this way the JVM can pause without interrupting the rest of the CICS workload, so serialization of access to CICS resources and the management of the start and end of all transactions is done using the QR TCB.

Addressing Integrity and Performance Issues

There are significant Central Processing Unit (CPU) costs in starting and stopping a JVM. Often these are greater than those incurred from the running of an application within a JVM. If a JVM can be serially reused by a number of CICS transactions then the overhead of starting and stopping the JVM is confined to the first and last transaction in the sequence, and those in between gain significant performance benefits from this.

However, if a JVM is reused then the potential exists for one Java application program to leave around objects whose state may interfere with the successful execution of transactions that later reuse the same JVM. EJBs are inherently self contained and as such cannot leave state around after their execution. Individual Java applications, on the other hand, tend to treat the JVM where they run as something as which they have exclusive use. They may assume that the JVM will be destroyed when the application finishes, and that all object instances, created during program execution, will be removed as a result of the termination of the JVM. This assumption is not valid when a JVM is serially reused and side effects, resulting from objects remaining in the JVM's storage heap following the termination of one application, may interfere with other applications which later reuse the same JVM.

To address this issue, the SDK provides a JVM with a modified heap structure, which can be driven in an attempt to reset its storage areas between separate program invocations. CICS TS version 2 can be configured to make use of this mechanism. Reset processing extends the operations of the JVM and does have some adverse effect on its performance characteristics.

Not all applications cause these problems and, for them, this reset processing is an unnecessary overhead. CICS TS 2.3 and the SDK 1.4.1 provide support for another JVM configuration, which offers the potential for reuse, but which places responsibility for reset processing onto the applications which run there. This provides significant performance benefits but does present opportunities for badly behaved transactions to interfere with other that reuse the same JVM.

Configuring JVMs for Reuse

CICS determines the JVM reuse option from a JVM Profile which it reads prior to starting a new JVM. The JVM Profile is a text file, stored in the UNIX Hierarchical File System (HFS), which is named in a Java program's CICS resource definition. CICS keeps track of reuse settings for all JVMs that it manages. The reuse setting is included as part of the Java Native Interface (JNI) call which CICS uses to start a new JVM. When a Java application has finished executing, CICS decide whether the JVM should be called for reset processing, and whether to allow the JVM to persist or to be terminated, based on its reuse property.

Three such settings are supported in CICS TS 2.3. These are defined with the REUSE option in a JVM Profile. This is the replacement for the Xresettable option, used with earlier releases of CICS, which is retained for migration purposes but which can only be used for two of the three possible settings. If both options are found in the same JVM Profile then REUSE take precedence over Xresettable. If neither are found, or if REUSE contains an unsupported value, then a default of REUSE=RESET is assumed.

If REUSE=NO is found, CICS starts a JVM and terminates it after it has been used once. This is referred to as a Single Use JVM in the CICS Documentation.

REUSE=RESET indicates that the JVM is long lived and that a controlled reset is required between program invocations. This is referred to as a Resettable JVM.

REUSE=YES indicates that a JVM is intended for reuse without the reset operation at the end of every program invocation. It is known as a Continuous JVM.

Single Use JVMs

Single use means that a JVM is created for each request, the Java application is run there, and then the JVM is thrown away. In this way, CICS effectively prevents the actions of one Java program from interfering with any other. This level of transaction isolation is expensive because every application incurs the cost of starting and stopping the JVM, and loading of all classes needed by it.

The JVM has a single heap structure, unlike the one used by the Resettable JVM, and does not undergo any reset processing prior to termination. This mode is the equivalent to Java on most other platforms.

The Resettable JVM

The Resettable JVM provides the potential to run consecutive Java application under its control. When an application finishes executing there, CICS calls the JVM to reset its storage areas, and waits for this operation to complete before terminating the current transaction. If the reset operation is successful, then the JVM becomes available for use by another CICS transaction. If the reset fails then CICS terminates the JVM.

The storage heap used by the resettable JVM is divided up into a number of areas, each with its own class loader. These areas are treated differently during reset processing, as shown in figure 1. In particular, there is a Transient Heap which contains objects created by a Java application as it executes. All these objects are deleted at reset time. There is also a Trusted Middleware Heap, used for middleware classes which are trusted to reset the object they create to some known state, as part of reset processing.

This heap structure is extensively documented in the SDK manual, shown in the reference section at the end of this paper.

Reset processing can fail for a number of reasons, such as a Java application changing one of the JVM's system property or by the loading of a Native Library. If the reset operation fails then CICS is informed that this has occurred and in response, it issues a JNI call to delete the JVM. This deletion prevents another application from encountering objects which erroneously remain in the storage heap after a reset failure.

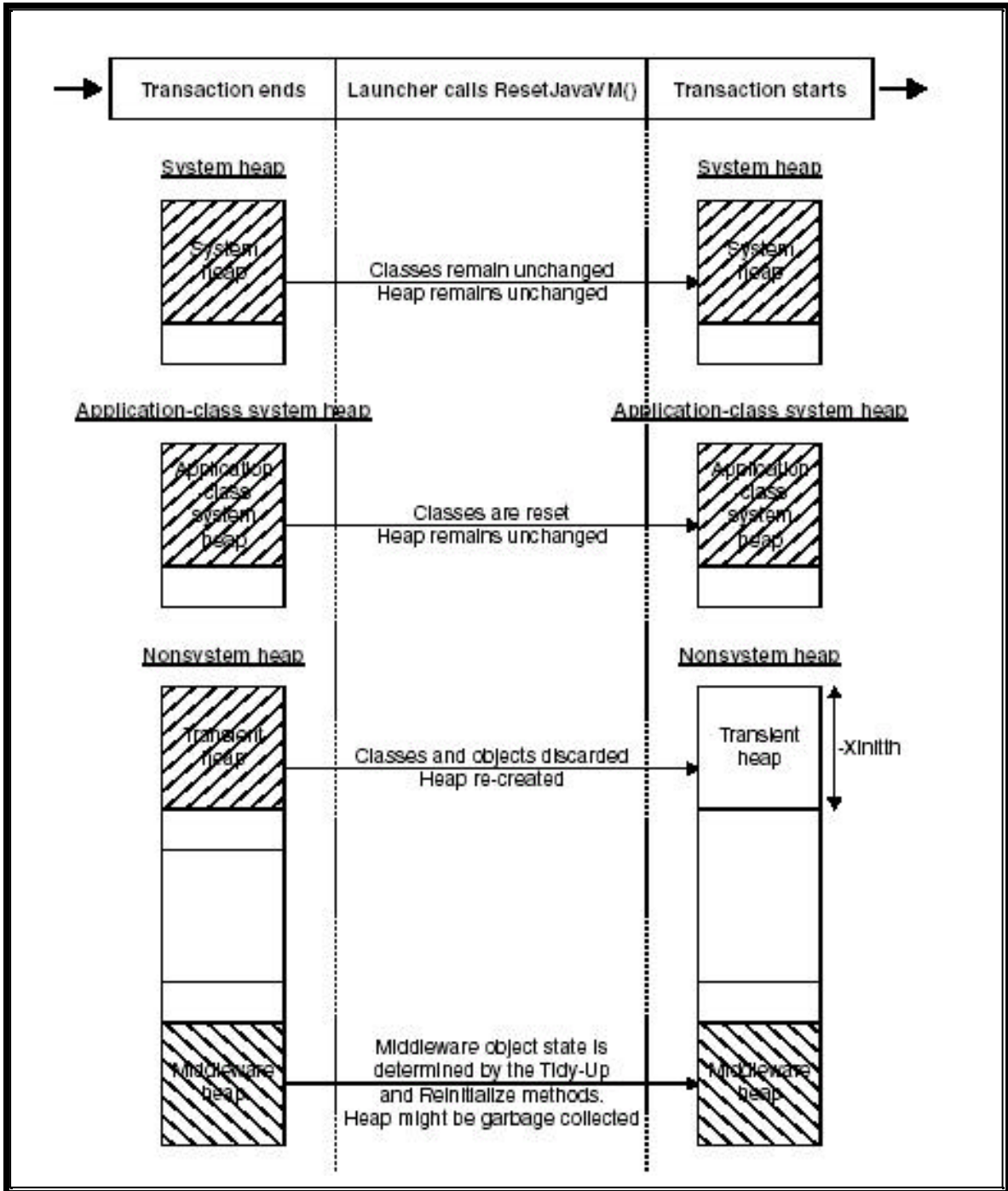


Figure 1. Reset operations against the Resetable JVM's Heap structure

The Continuous JVM

This is a potentially long lived JVM, which is started with a single storage heap. There is no mechanism for automatically resetting this storage when a Java program finishes execution. Each program is responsible for removing unwanted state data when it terminates, or restoring these

objects to their initial state prior to completion. Programs can intentionally leave data for subsequent applications to use, and by doing so they make use of a form of caching which is not available through the Resettable JVM.

The simplified storage management of the Continuous JVM offers large performance benefits over the Resettable JVM. However there are risks associated with running applications which do not conform to the reuse requirements of this mode.

Java programs running in this mode should not carry out actions which alter the JVM's state unless they restore the original values before ending. The most common cause of this is from the inclusion of static storage objects in Java programs. These are not automatically reinitialized on subsequent program invocations and should be used with care, when running in this mode. One way to prevent this is by defining all class fields as final.

Reuse choices

The selection of an appropriate reuse mode for a particular JVM is largely influenced by the nature of the applications which will run in it. However some consideration should also be given to those factors which are used to control all the JVMs that a CICS region manages. These are discussed in the next two sections of this document.

The Shared Class Cache Facility

Overview

The SDK 1.4 provides a mechanism which allows Java classes to be cached centrally and shared between different JVMs. CICS TS 2.3 introduces a Shared Class Cache Facility which extends this function to some, or all, of the JVMs that it controls.

The collection of JVMs which use this facility is referred to as a JVMset. This comprises of a single JVM, known as the Master, who's role is to manage the shared class cache, together with other JVMs, called Workers, which service individual Java requests. These requests can be to run a Java program, an EJB or an IIOP application. Figure 2 shows the relationships between Master and Worker JVMs within a CICS region's JVM Pool.

A CICS region can have a single Shared Class Cache which uses this feature. CICS controls the launching of the Master JVM and any Workers that are needed to service requests to run Java components. Individual Worker JVMs in a JVMset can have different characteristics, defined in their associated JVM Profiles and the JVM Properties files. These may be used to define the size of a JVM's storage heap, to set the trace options it uses, or to add some security controls.

Some characteristics of a JVMset are common to all of its JVMs. They are taken from the JVM Profile of the Master JVM. These include the REUSE option, outlined in the previous section of this paper. A Master JVM is potentially long lived and as such must be started in either resettable or continuous mode. The Master JVM cannot be configured with REUSE=NO and, as a result, Worker JVMs cannot run in this mode either.

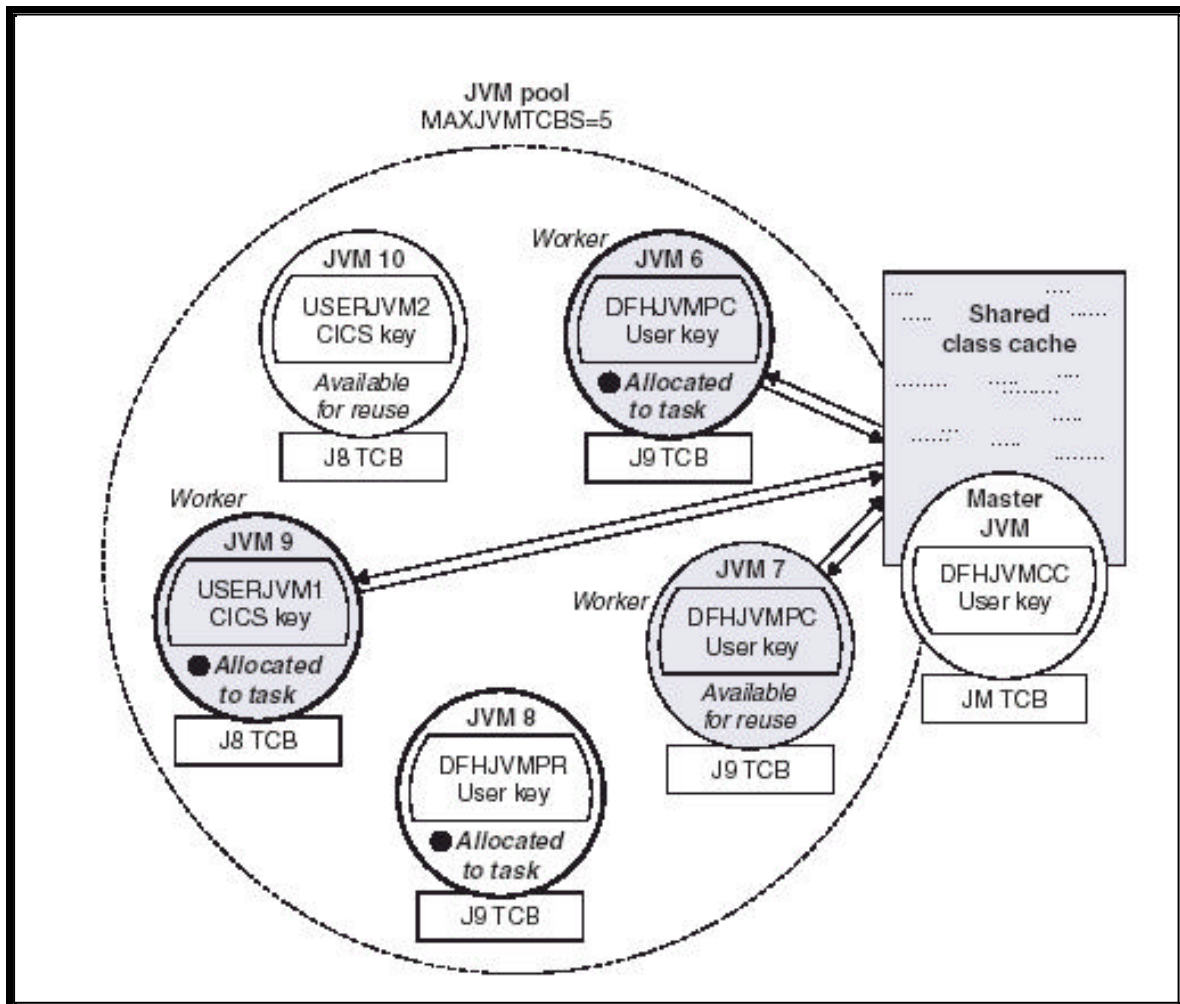


Figure2. The JVM components of a Shared Class Cache Facility

Benefits of using the Shared Class Cache

The Shared Class Cache facility offers a number of benefits to customer. Java classes are loaded once per CICS region rather than once per JVM, reducing the class loading overhead for all the Workers in a JVMset. It also reduces the overall storage requirement for the JVMset, by having one copy of each class in the cache area, instead of one in each Worker JVM's storage heap.

Because Worker JVMs run in either resettable or continuous mode they receive the performance benefits described earlier, when the JVM is reused for a number of transactions. In addition, the startup costs for a Worker JVM is significantly less than that of a Stand-alone JVM.

Managing Java Resources in CICS

A number of factors influence the way that a CICS region matches Java requests with the JVM resources it controls. Options in the CICS program resource definitions, the JVM Profiles, and the System Initialization Table (SIT) all affect the running of Java workloads. This section explains how these options can be used to configure the behavior of a CICS region.

Storage protection for Java applications

Prior to CICS TS 2.3, all Java programs ran in CICS key 8 storage areas. They made use of JVMs which were each started on a separate open TCB, known as a J8. As a result, the CICS storage protection mechanism could not be used with Java programs. Storage protection prevents a user application from inadvertently overwriting CICS control blocks or its own programs. In CICS TS 2.3, customers can choose to run Java programs in either CICS or User key. Those selected to run in User key do so by using JVMs which are started on a new open TCB: the J9.

The storage key information is derived from a Java program's resource definition. This defaults to User key if not explicitly defined there. However, if storage protection is turned off in the CICS region then all Java applications will run on JVMs which use J8 TCBs.

TCB and JVM management

Each CICS region has a limit for the maximum number of TCBs it has for its Java workloads. This value is defined using the MAXJVMTCBS option in the System Initialization Table (SIT). The default value of 5 TCBs can be overridden during CICS initialization or by using System Programming Interface (SPI) commands or the Master Terminal Transaction CEMT.

One TCB is used for each JVM. The set of JVMs that a CICS region controls is referred to as the JVM Pool. The maximum pool size is equal to the value of MAXJVMTCBS. The Master JVM is started using a special TCB, the JM TCB, and is not considered as part of the overall JVM Pool size.

With Storage Protection active, the JVM Pool can contain a mixture of JVMs running with J8 and J9 TCBs at the same time. The REUSE option, described earlier in this paper, allows JVMs to be started in one of three modes. Those that have REUSE=NO, run for one transaction and are then

removed from the JVM pool. Others can remain there for some time. When a pool contains a number of long lived JVMs, then at any time some of these may be servicing program requests while others are waiting for work.

The JVM pool can contain a number of JVMs which have been started with the same JVM Profile. JVMs started with different JVM Profiles are distinguished from each other by CICS, even if different JVM Profiles contain identical sets of options.

Request Matching

When a new request arrives, CICS attempts to match the storage key and the JVM Profile name, taken from the Java program's resource definitions, with that of a JVM which it has within the JVM Pool. If a match is found and that JVM is free, then CICS dispatches the new request to it. If no match can be found, or if all JVMs that are suited to the request are currently carrying out other work, then CICS looks to see if the JVM pool has reached its maximum size. If it has not then a new JVM is started to service this request. However, once the pool is full no more JVMs can be created, and new requests are placed on a work queue for processing when a suitable opportunity arises.

When the JVM Pool is full there may be some JVMs there which match the requirements of a new request, but which are currently active. They may be others which are not active, but which do not match the JVM Profile and/or the storage key requirements of the new request. CICS makes use of an elaborate mechanism to match requests in its work queue with its JVMs. CICS has the ability to destroy a JVM which is free, and then to start a new one using a different JVM profile on the same TCB. Equally CICS can recycle both TCB and JVM.

Whenever a JVM finishes its current work assignment, CICS looks at all outstanding requests on the work queue. Priority is not always given to the oldest item, and CICS may decide that some other request is a better use for this JVM. However once a request has been on the work queue for a prolonged period it will be given priority over all other requests, thus preventing it from being overlooked indefinitely.

The termination and startup of a JVM is expensive, both in terms of the CPU these processes consume, and in the slowing of response rates for individual requests. There is a larger overhead when a TCB has to be recycled as well as a JVM.

The request matching process will reduce the number of JVM and TCB recycle events. The delay this may have for individual requests which are made to wait for a JVM that matches their precise requirements is often less than the time taken to recycle a JVM and its TCB.

Conclusions and Recommendations

This paper shows that a JVM can be configured for use by a single application and then destroyed, or can be reused by more than one application. Two types of long lived JVMs are supported for use with CICS. Those which are created with a split storage heap can be driven for a reset of the storage areas under the control of the JVM between individual CICS transactions. Those which have a single storage heap rely on any necessary reset processing to be done as part of the termination of individual Java applications which run there. Classes can be cached centrally and shared between JVMs. JVM Profiles are used to tailor the properties of individual JVMs. Individual applications can use their own JVM Profiles so that individual JVMs support only a single application, or JVM profiles can be shared so that many applications may reuse a single JVM.

There can be large differences between the needs of individual workloads. As a result, there is no single configuration which matches the needs of all customers who use Java within CICS. Advice is offered in this paper to help customers select the best configuration for their specific needs.

Reuse settings

Selection of the REUSE setting for a particular JVM is largely influenced by the applications it is to service. The reuse characteristics of existing applications, whether they have been written in house or have been provided by another party, may not be well understood. Those which are not should be tested extensively to find out if they can be used with a persistent reusable JVM. When developing new CICS Java applications, consideration should be given to those issues relating to how they may eventually be run within a long lived JVM.

Other factors which may influence this decision relates to the way an application is used. In production a mixture of performance criteria and system stability apply. In a test system, the emphasis may be more on examining the behavior of a new application rather than getting it to run quickly. As a result an application may be developed in a Single-use JVM, and then moved to a Resettable or a Continuous JVM for system testing before transferring to the most suitable environment for the production system..

The Single-use JVM is best suited to a development environment. Each application executes there once before the JVM is destroyed. Any side effects from other applications are removed. Fresh copies of classes are loaded each time an application is run, so code changes can be picked up.

There is a role for Single-use JVMs in production systems. The overhead of starting and stopping them for each application invocation makes them unattractive for all but infrequently used ones. The majority of applications, run in a production system require the performance gains which persistent reusable JVMs offer.

The Resettable JVM provides the benefits of reuse while offering the greatest protection between program invocations. Applications whose reset characteristics are not well understood, or which do

not comply with those needed by the Continuous JVM, should be run in this mode. Some performance overhead will result from management of the split storage heap.

An application run in the Resettable JVM may cause problems which prevent the reset processing operation from completing successfully. This will result in the JVM being destroyed and later requests may have to start up a new JVM before they are processed. The impact of unresettable events occurring will adversely effect a Java workload as a whole. Trace options can be used with the Resettable JVM, to help in determining the cause of these errors, and remedial action should be taken so that the reset operation completes successfully.

The Continuous JVM offers significant performance benefits over a Resettable or Single-use JVM. Applications whose behavior is not well understood are better run in the Resettable JVM unless they can be extensively tested in continuous mode before they are placed in a production system. New applications should be written in such a way so that they can receive the performance benefits from the Continuous JVM without unduly effecting other CICS transactions which reuse the same environment.

Sharing cached classes

The use of the Shared Class Cache Facility relates more to the scale and scope of the Java workload that a CICS region supports, rather than the nature of individual transactions. If a system has more than one JVM running in it at any time supporting a common workload, then there are benefits to be gained from the use of shared classes, as outlined earlier in this paper. As the numbers of duplicate JVMs increase then the benefits increase as well.

A single CICS region can only have one active JVMset. All the JVMs that make up a JVMset must be running either in continuous mode or resettable mode. If a Java workload makes use of a mixture of Continuous and Resettable JVMs then multiple CICS regions will be needed if both are to make use of the Shared Class Cache Facility.

If there is a mixed workload, which use different JVMs for different applications, then the Shared Class Cache Facility will have to be suitably sized to hold the classes of the combined workload rather than that for an individual Worker. Classes are not removed from the shared cache in the event of Workers being recycled for different workloads. The starting of new Workers may result in different classes being loaded into the shared cache, and this will only be possible if it has space for them. If the Just In-Time (JIT) compiler is in use then space will also be needed in the shared cache for the JITed code.

JVM recycling events

The recycling of JVMs in the JVM Pool is controlled by CICS, but can be influenced in some ways. In a busy system, recycling events will have an impact on the overall performance of the system and should be kept to a minimum.

If an entire Java workload can be run in a particular storage key then TCBs will never have to be recycled. If all the JVMs also share a common JVM profile then the recycling mechanism is in effect disabled as all queued requests will match all JVMs in the pool.

The storage requirements for a single JVM supporting several applications will be less than that used by one JVM per application, as each JVM will require storage for the system classes that it uses. Individual applications may require specific options to be set for the JVMs where they run, and this may limit the way JVMs can be shared. The Shared Class Cache Facility can offer a suitable compromise for such configuration issues.

The use of a different JVM profile for every application in a workload will not usually be necessary. If two profiles contain identical options and differ only in name then this will unnecessarily increase the potential for recycling events taking place in the JVM pool. This should be avoided and a common JVM profile used for this purpose.

A small number of JVM profiles can be used to combine Java applications into groups each which runs within individual JVMs. If the number of JVM profiles is small compared to the size of the JVM pool and the number of requests against each of these JVMs is roughly the same then little recycling should take place. However if the number of JVM profiles is large compared with the size of the pool or if some JVMs are used less frequently than others, then recycling events are more likely to occur. As a result, it is recommended that the ratio between the number of JVM Profiles and the size of the JVM Pool is kept as low as possible.

Summary

CICS TS 2.3 has enhanced support for the Java programming model over earlier releases. In doing so, greater customization is now possible for the Java runtime environment.

Persistent reusable JVMs offer potential performance gains over Single-use JVMs. The Resettable JVM removes the risk of one application leaving state data in its storage heap which may adversely effect another application reusing the same environment. It should be used for those applications which are not well understood, or which have not been designed to run in long lived JVMs. The Continuous JVM provides the best performance characteristics. It should be used to run applications which reset themselves prior to termination. New Java applications should be designed to run in the Continuous JVM wherever possible.

The Shared Class Cache Facility reduces the memory requirements for a set of JVMs that run a common workload. Worker JVMs have improved startup times over stand-alone JVMs. The Class Cache can be used with either Continuous or Resettable JVMs, but not a mixture of both. The

sharing of classes is particularly relevant to production systems where large numbers of JVMs are used for the same application.

The JVM startup options are specified in a JVM Profile. JVM Profile names are used to match Java program invocation requests to specific JVMs. A balance should be sought between the number of different JVM profiles that are used and the maximum size of the JVM Pool. CICS manages the recycling of JVMs once the JVM Pool is full to provide best overall performance.

References

Manuals

CICS TS for z/OS version 2 release 3: Java Applications in CICS SC34-6238-00

CICS TS for z/OS version 2 release 3: Performance Tuning SC34-6247-00

IBM Development Kit for z/OS version 1 release 4 : New IBM Technology featuring Persistent Reusable Java Virtual Machines SC34-6201-01

Technical Reports

A Serially Reusable Java Virtual Machine Implementation for High Volume, High Reliability, Transaction Processing TR 29.3406

Trademarks

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

CICS IBM z/OS

Java and all Java based trademarks are trademarks of Sun Microsystems Inc. in the United States, other countries or both.

Contact information

Author - Mike Brooks

Email address - brooksm@uk.ibm.com