Understanding WAS z/OS Timeouts

David Follis, IBM

Gary Picher, IBM

Mike Stephen, IBM

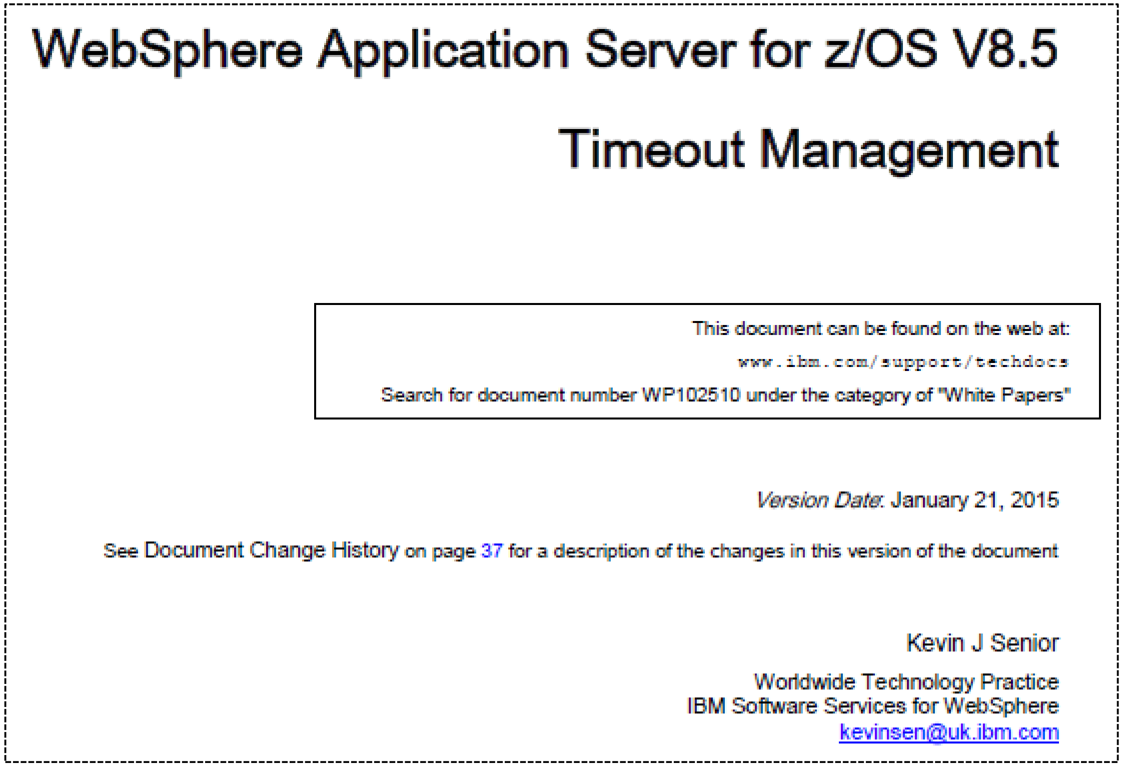
WebSphere Application Server

Session	Title	Time	Room
17363	Debug 101-Using ISA Tools for Apps in WebSphere Application Server z/OS	Monday 11:15	Europe 11
17367	WebSphere Liberty on Windows and z/OS (Among Other Things) Hands-On Lab	Tuesday 10:00	Asia 5
17361	ABCs of WAS	Tuesday 1:45	Oceanic 7
17368	z/OS Connect: Opening up z/OS Assets to the Cloud and Mobile Worlds	Tuesday 3:15	Oceanic 7
17362	Configuring Timeouts for WebSphere Application Server on z/OS	Wednesday 8:30	Oceanic 7
17366	WebSphere Liberty and WebSphere Application Server Classic - What's New?	Wednesday 11:15	Oceanic 7
17364	IBM Installation Manager for z/OS System Programmers: Web-based Installs, Fix Packs, and How iFixes Really Work	Thursday 4:30	Oceanic 7
17365	JSR 352 - The Future of Java Batch and WebSphere Compute Grid	Friday 10:00	Oceanic 6

Agenda

- Overview
- Timeout Scenarios
- Recommendations
- Gathering Documentation
- References

Advertisement



WebSphere Application Server for z/OS V8.5

Timeout Management

This document can be found on the web at:

www.ibm.com/support/techdocs

Search for document number WP102510 under the category of "White Papers"

Version Date: January 21, 2015

See Document Change History on page 37 for a description of the changes in this version of the document

Kevin J Senior

Worldwide Technology Practice
IBM Software Services for WebSphere
kevinsen@uk.ibm.com

This session is based on the work of Kevin Senior, who published his findings in a white paper

An objective of this session is to give you a sense for the framework of the timeout structure of WAS z/OS

The white paper is what you then use to drill deeper into the topic

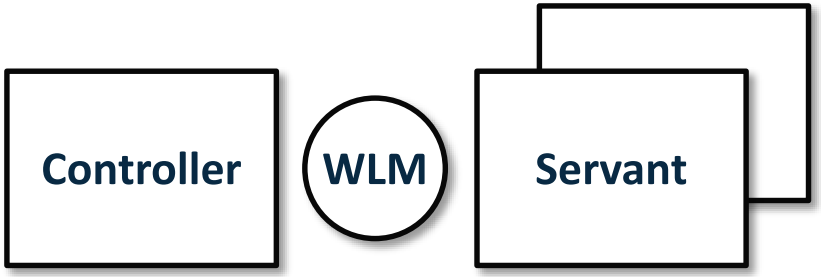
WP102510 at `ibm.com/support/techdocs`

Overview

Classic WAS z/OS, not Liberty

This topic relates to the Classic WAS z/OS product, not the "Liberty" topic:

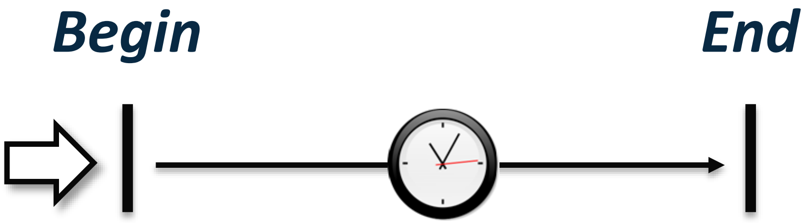
You are here



Liberty has a different set of timeout patterns

Similar in some ways, but different enough that it really is a separate topic

What are Timeouts? And Why Do We Care?



Time it takes to complete the processing between begin and end

Within a computing system, timeouts occur when a request process takes longer to complete than is expected

How many different timers are there? We'll get to that. Short answer: more than a few.

When something doesn't complete in the expected time, *something* needs to happen. Otherwise, users will be left waiting indefinitely, and server resources are used and never freed.

Actions That May Be Taken for a Timeout

Throw an exception

For cases where the timeout must be handled or the user alerted

Example: a `socket.read( )` IOException

If unhandled, then it is possible other timers will expire. Best practice: handle exceptions.

Reset the JVM (i.e., abend the servant)

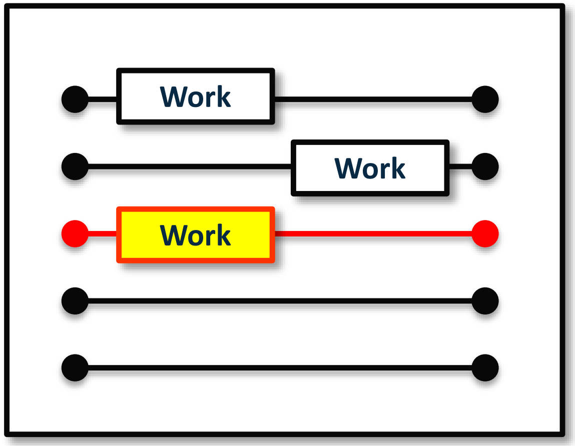
For cases where the only resolution is a reset of the environment

Example: When a 'dispatch timeout' occurs

WAS z/OS has a few things to help avoid or delay the EC3 abend ... more coming up.

Is it *Really* Necessary to EC3 the Servant?

Java Virtual Machine



This is true for any JVM ... z/OS or other operating system platform

For WAS z/OS the JVM is the servant address space. The ability to configure multiple servants provides availability during JVM reset.

A worker thread within the JVM is hung

Could it be left in a hung state?

Yes ... this is what WAS z/OS is capable of doing. More on "threshold" later.

But ... if there's something wrong with the application and *all* threads are hanging, then eventually all the threads will be exhausted. *Something* has to be done.

Could the thread be "reset" in some way?

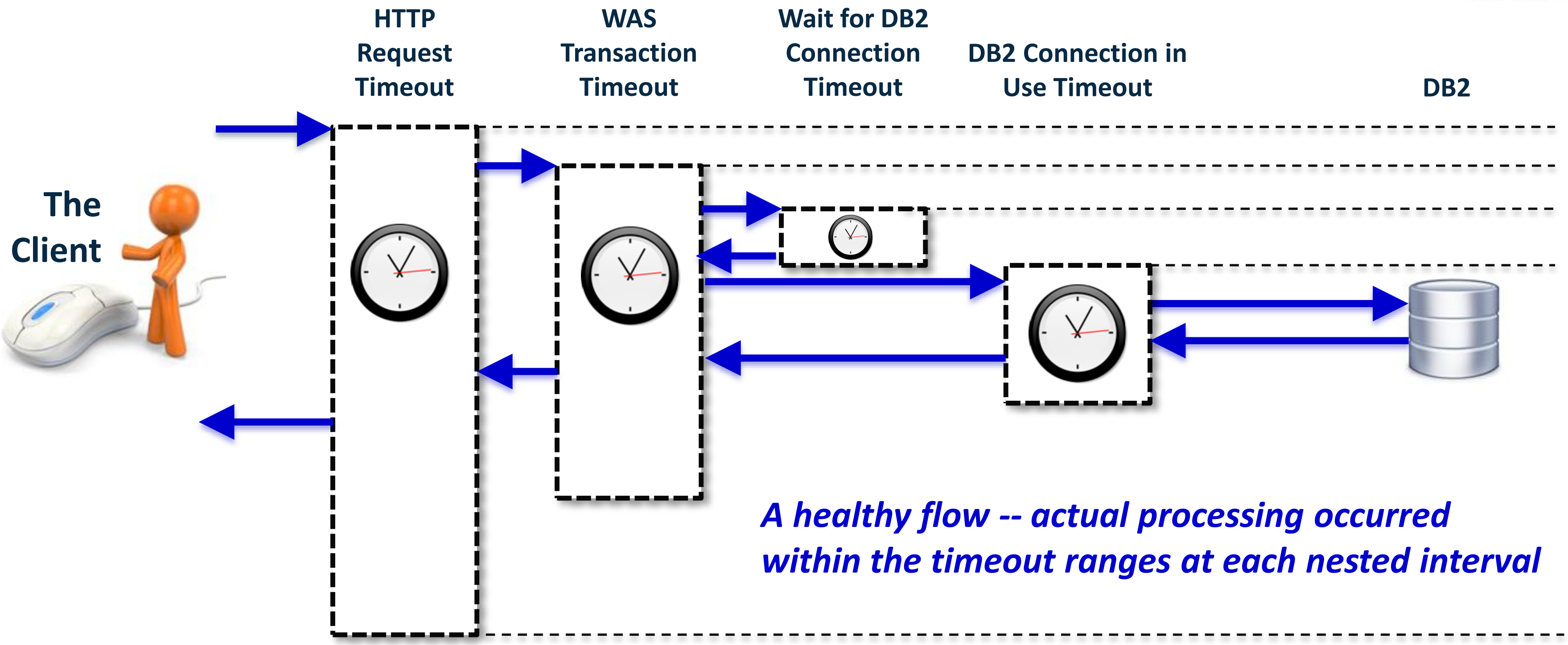
No ... the Java specification does not allow this. There is good reason for this:

- The dispatch thread probably has Java on the call stack
- Java has no MVS recovery (ESTAE or MVS Resource Manager)
- Java's signal handler can't clean up a thread's resources (synchronize locks remain held)
- Loss of a single thread would hang or corrupt the process

What's left is to reset the entire JVM

That implies stopping and restarting the JVM

"Nesting" of Timeouts

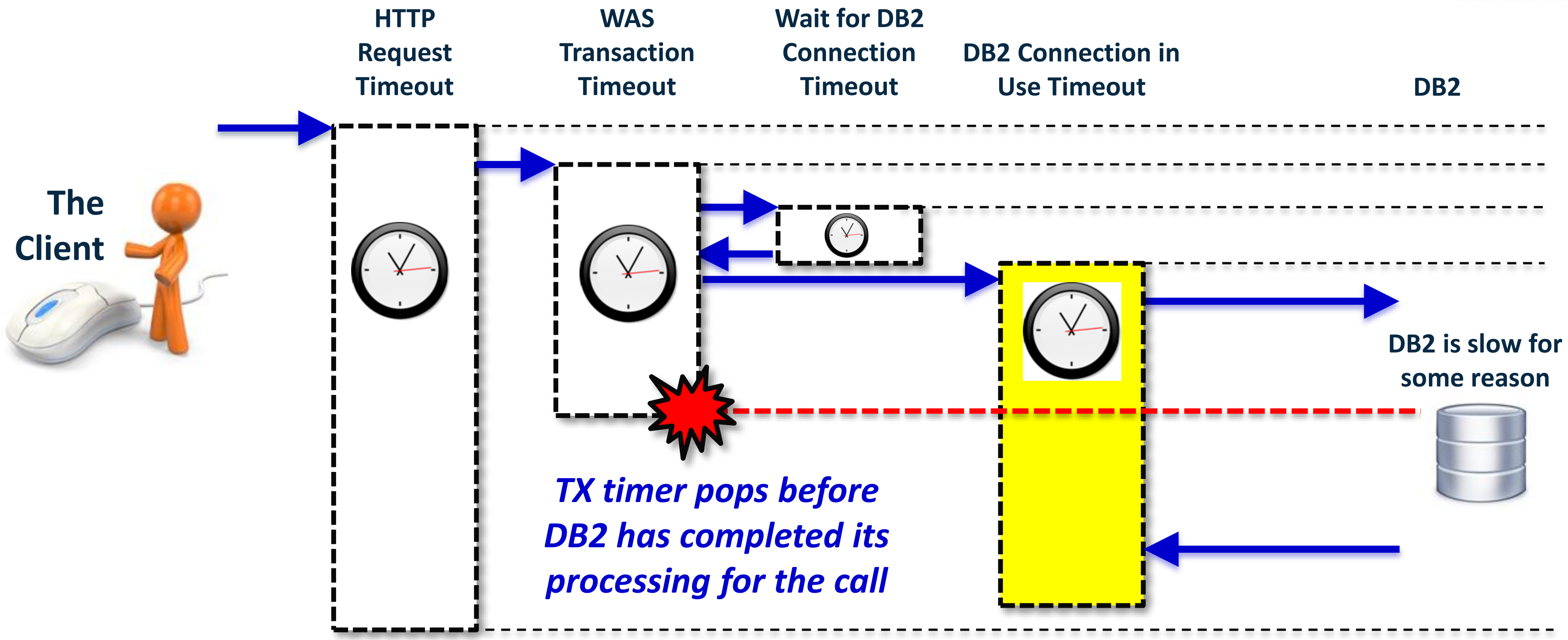

HTTP Request Timeout
WAS Transaction Timeout
Wait for DB2 Connection Timeout
DB2 Connection in Use Timeout
DB2
The Client

A healthy flow -- actual processing occurred within the timeout ranges at each nested interval

Two key points here:

1. Timeouts are often "nested" ... that is, they operate *within* earlier timeout ranges
2. In general, timeout values closer to the client are longer than timeout values further way from the client

Improperly Configured Nested Timeouts

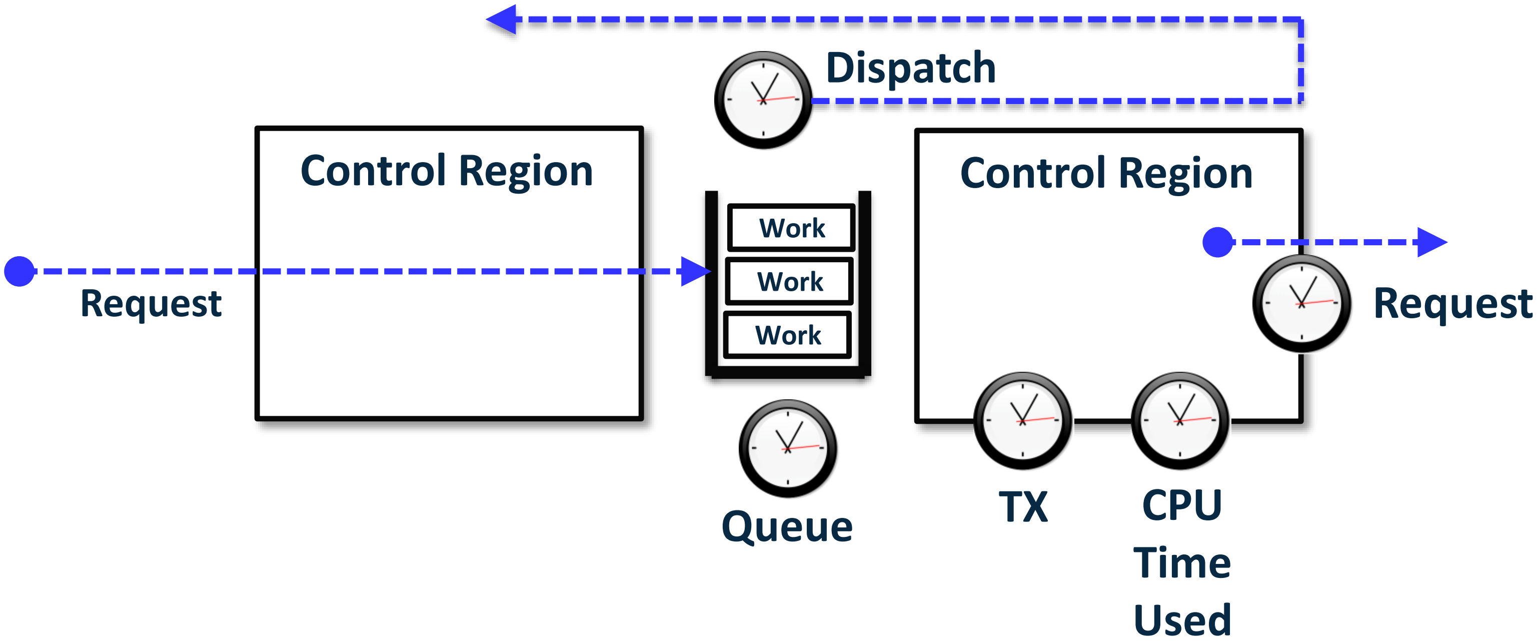

HTTP Request Timeout
WAS Transaction Timeout
Wait for DB2 Connection Timeout
DB2 Connection in Use Timeout
DB2
The Client
DB2 is slow for some reason

TX timer pops before DB2 has completed its processing for the call

Two points to be made here ...

1. Violates general rule of timeouts closer to client being longer than timeouts further away from the client
2. WAS z/OS times out the transaction and begins rollback ... but there is the potential DB2 is still processing request

Overview of the Some Key Timers



Dispatch -- time from placement in queue to work complete

Queue -- time in queue prior to dispatch into servant (expressed as % of Dispatch)

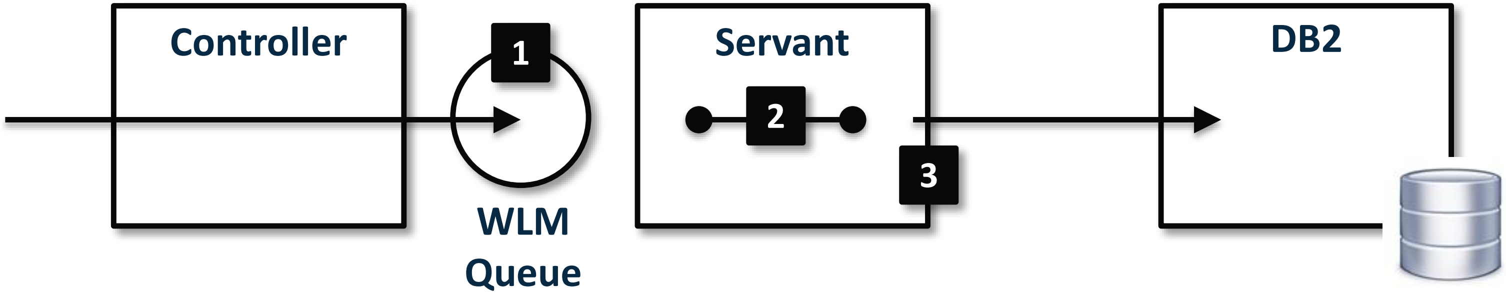
Transaction -- time from start to end of a transaction

CPU Time Used -- limit on the amount of CPU time a thread may consume before being quiesced

Request -- time for IIOP request out from application until return

Timeout Scenarios

HTTP Request and JDBC Call to DB2



1. Dispatch Timer = 300 seconds (default)
2. Transaction Timer = 120 seconds (default)
3. JDBC Connection Timer = 60 seconds (set value)
   The default value is 180 seconds

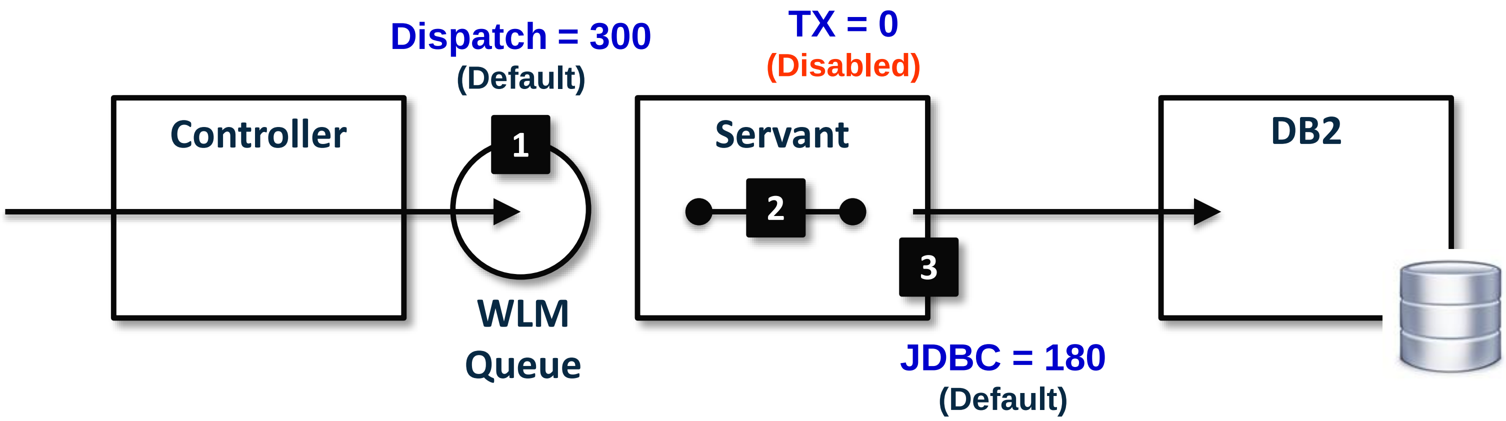
Note -- the *default* values do not naturally nest very well ... the inner value (JDBC connection) is greater than the middle (TX timer)

Assume a request to DB2 does not complete within the 60 second timeout value ...

- JDBC Connection timer pops
- Application catches and returns error message to caller
- Transaction canceled (so that timer no longer in effect)
- Dispatch completes (so that timer no longer in effect

This is a "well behaved" timeout scenario

Dispatch Timeout



Request to DB2 takes longer than 180 seconds and throws exception

Application is coded to try request again

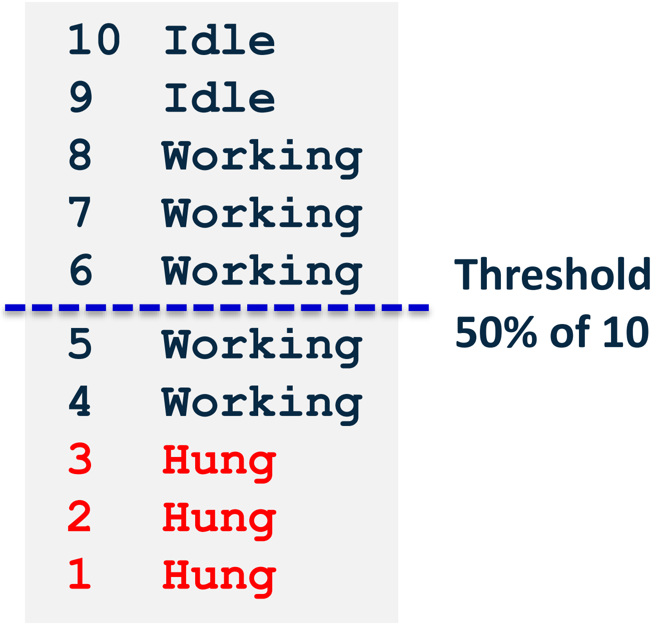
Same result ... DB2 takes longer than 180 seconds.

What happens?

- After first JDBC timeout the dispatch timer continues to tick away
- The transaction timer is out of the picture since it was disabled
- Second request implies up to 360 seconds (2 x 180), *which is more than dispatch 300*
- Dispatch timer pops and the thread is marked as hung
- WAS z/OS goes into attempted recovery
- If thread still hung, then servant EC3 abend

Threshold -- A Way to Delay EC3

Variable to set Thread count	`servant_region_custom_thread_count`
Variable to set Threshold	`server_region_stalled_thread_threshold_percent`
Message to verify threads	`BBOO0234I SERVANT PROCESS THREAD COUNT IS xx`



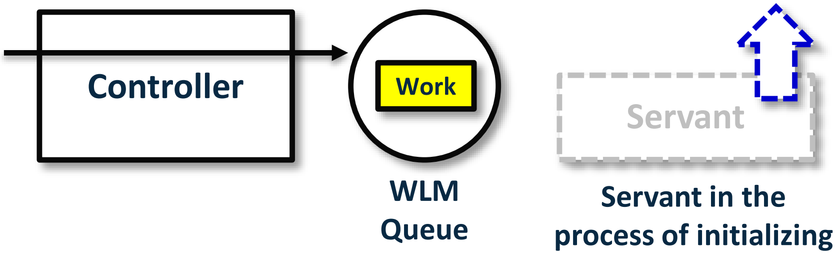
```
10  Idle
9   Idle
8   Working
7   Working
6   Working
-----------------
5   Working
4   Working
3   Hung
2   Hung
1   Hung
```

Without threshold set, the servant region would EC3 abend when the *first* thread is marked as hung

With threshold set, the servant delays action until hung threads meets or exceeds the threshold value

The “Bouncing Servant” Problem

This results when a new servant, start after an earlier EC3 timeout abend, receives work that’s been on the WLM queue a long time. By the time it gets into the servant, there’s no time to actually do the work ...



Controller

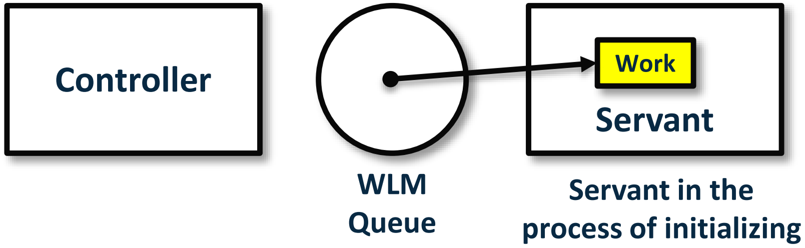
Work

WLM Queue

Servant

Servant in the process of initializing

- Servant unavailable to take work
- Work sits in WLM queue
- Dispatch timer continues to tick down



Controller

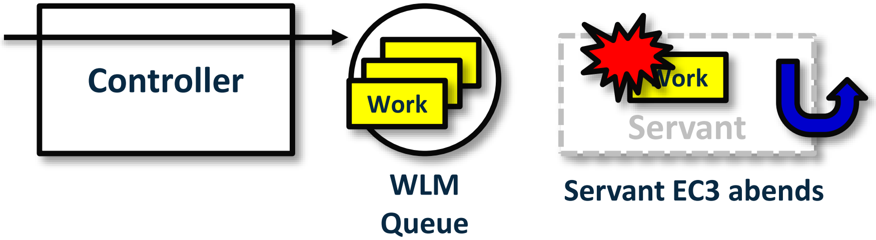
WLM Queue

Work

Servant

Servant in the process of initializing

- Servant initializes and signals ready for work
- WLM sends the work to the servant
- Dispatch timer has only a few ticks left ...



Controller

Work

WLM Queue

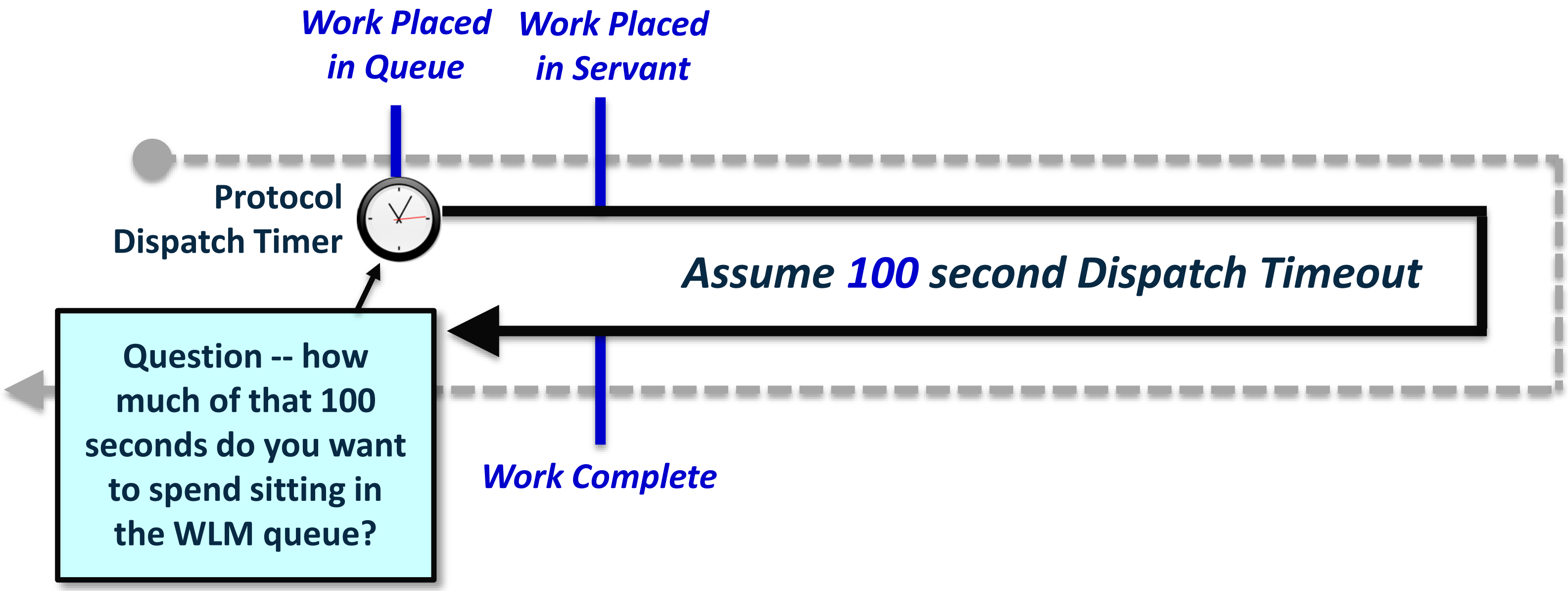
Work

Servant

Servant EC3 abends

- Work just gets going when ...
- Dispatch timer pops, causing EC3 abend
- Servant begins to re-initialize
- Meanwhile, more work in queue ...

Queue Timeout Percent


Work Placed in Queue
Work Placed in Servant
Protocol Dispatch Timer
Assume 100 second Dispatch Timeout
Question -- how much of that 100 seconds do you want to spend sitting in the WLM queue?
Work Complete

control_region_xxxx_queue_timeout_percent

The underlying protocol of the request -- http, https, mdb, etc.

✓ Set at 10% ... time in queue limited to 10 seconds; if dispatched then 90 seconds to work

✗ Set to 99% ... time in queue 99 seconds, leaving just one second for work

Recommendations

“Session” or “Servant” Option

```
protocol_http_timeout_output_recovery  = | SESSION
protocol_https_timeout_output_recovery = | SERVANT
```

SERVANT

If thread hung and no threshold, then EC3 abend of servant region

If thread hung and threshold not yet met, then thread remains hung

SESSION

If thread hung, then TCP socket and HTTP session ended and message sent to the client.

Thread is left to either complete or remain hung

Recommendation: use SERVANT with threshold because it provides a way to reset the JVM if hung threads stack up beyond defined threshold. SESSION has potential to exhaust thread pool with hung threads.

Other Recommendations

General recommendations

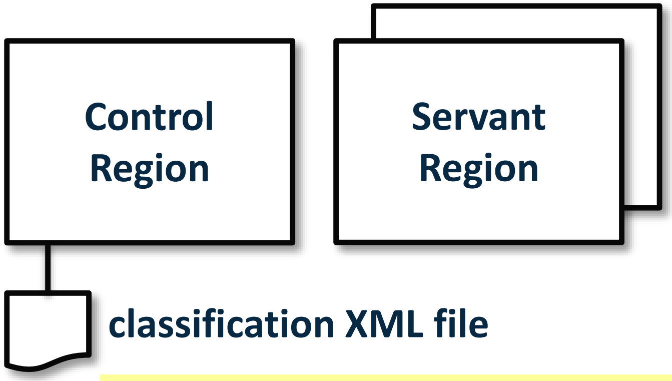
- Set the timeout delay to let the innocent get out of the way
- Use the classification XML to set granular timeout values
- Set queue timeout to avoid bouncing servants
- Defer IIOP work until after minSRs (protocol_accept_iiop_work_after_min_srs) HTTP defaults this way
- Avoid use of hung thread threshold unless you know what's going on
- Control_region_dreg_on_no_srs to stop the listeners when you can't run work (no SRs)
- Consider control_region_confirm_recovery_on_no_srs which gets you a WTOR before resuming the listeners

If bad things are happening....

- Adjust timeouts if that will help (slow vs. stuck)
- Turn off doc collection once you have enough doc...don't forget to turn it back on once the problem is resolved
- Consider hung thread threshold
- Be more aggressive - kill it faster
- Use DPM to help diagnose slow requests

Gathering Documentation

Granular RAS and Timeouts



Provides a way to drive timeout values to the request level, rather than at the server level

Values set at request level override server level

```
dispatch_timeout="_____"
queue_timeout_percent="_____"
request_timeout="_____"
stalled_thread_dump_action="_____"
cputimeused_limit="_____"
cputimeused_dump_action="_____"
dpm_interval="_____"
dpm_dump_action="_____"
SMF_request_activity_enabled="__"
SMF_request_activity_timestamps="__"
SMF_request_activity_security="__"
SMF_request_activity_CPU_detail="__"
classification_only_trace="__"
message_tag="_____"
timeout_recovery="_____"
```

Timeout Dump Actions

Some timeouts allow you to set what action should be taken if the timeout occurs. This can be useful in problem determination:

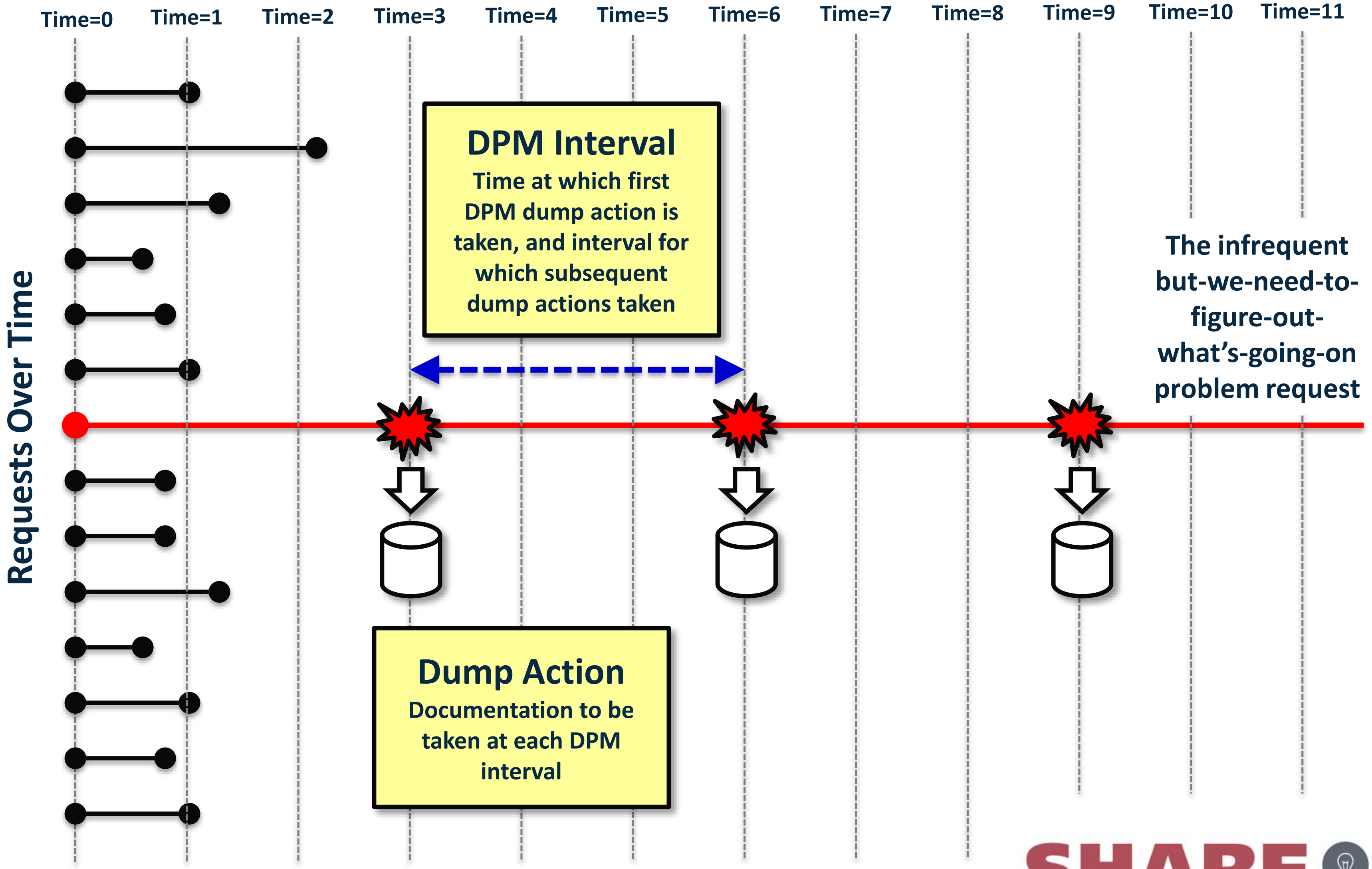
Examples:

Variable: `server_region_xxxx_stalled_thread_dump_action=`

XML file: `stalled_thread_dump_action=`

`none`	As name implies, does nothing.
`traceback`	Gives you information for ONE thread
`javacore`	Tells you what ALL the threads are doing and some environmental information
`heapdump`	Dumps the whole JVM heap
`javatdump` `svcdump`	These two are similar, except TDUMP only contains things visible to an unauthorized program and may not be enough.

Dispatch Progress Monitor (DPM)



DPM Interval
Time at which first DPM dump action is taken, and interval for which subsequent dump actions taken

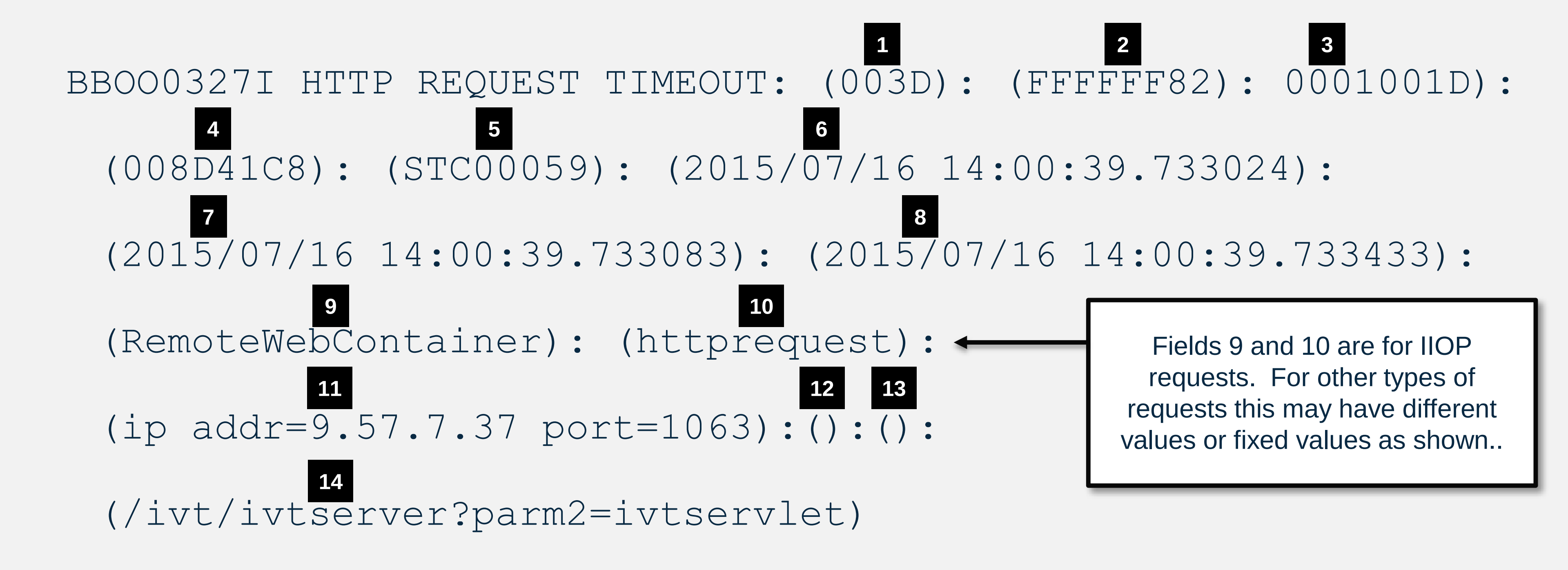
Dump Action
Documentation to be taken at each DPM interval

The infrequent but-we-need-to-figure-out-what's-going-on problem request

Complete your session evaluations online at www.SHARE.org/Orlando-Eval

The BBOO0327I Message

A long message that carries a wealth of information about a timeout:



```
BBOO0327I HTTP REQUEST TIMEOUT: (003D): (FFFFFF82): 0001001D):
  (008D41C8): (STC00059): (2015/07/16 14:00:39.733024):
  (2015/07/16 14:00:39.733083): (2015/07/16 14:00:39.733433):
  (RemoteWebContainer): (httprequest):
  (ip addr=9.57.7.37 port=1063):():():
  (/ivt/ivtserver?parm2=ivtservlet)
```

Fields 9 and 10 are for IIOP requests. For other types of requests this may have different values or fixed values as shown..

1 - (servant hexadecimal ASID)
2 - (request id)
3 - (internal information)
4 - (servant TCB address)
5 - (servant job id or job name)
6 - (time request received in controller)
7 - (time request queued to WLM)
8 - (time request dispatched in servant)
9 - (class name)
10 - (method name)
11 - (string indicating the origin of the request)
12 - (future data)
13 - (future data)
14 - (request URI)

The BBOO0327I with APAR PI40209*

This provides additional information on WLM classification:

```
BBOO0327I HTTP REQUEST TIMEOUT: (003D): (FFFFFF82): 0001001D):
  (008D41C8): (STC00059): (2015/07/16 14:00:39.733024):
  (2015/07/16 14:00:39.733083): (2015/07/16 14:00:39.733433):
  (RemoteWebContainer): (httprequest):
                                  A        B        C
  (ip addr=9.57.7.37 port=1063):(IVTTC   ,WASIVT  ,RPTIVT):():
  (/ivt/ivtserver?parm2=ivtservlet)
```

A - Transaction Class
B - Service Class
C - Report Class

The other fields are the same as shown on the previous chart. This APAR makes use of the “future data” field represented by block 12 on that chart

* http://www-01.ibm.com/support/docview.wss?uid=swg1PI40209

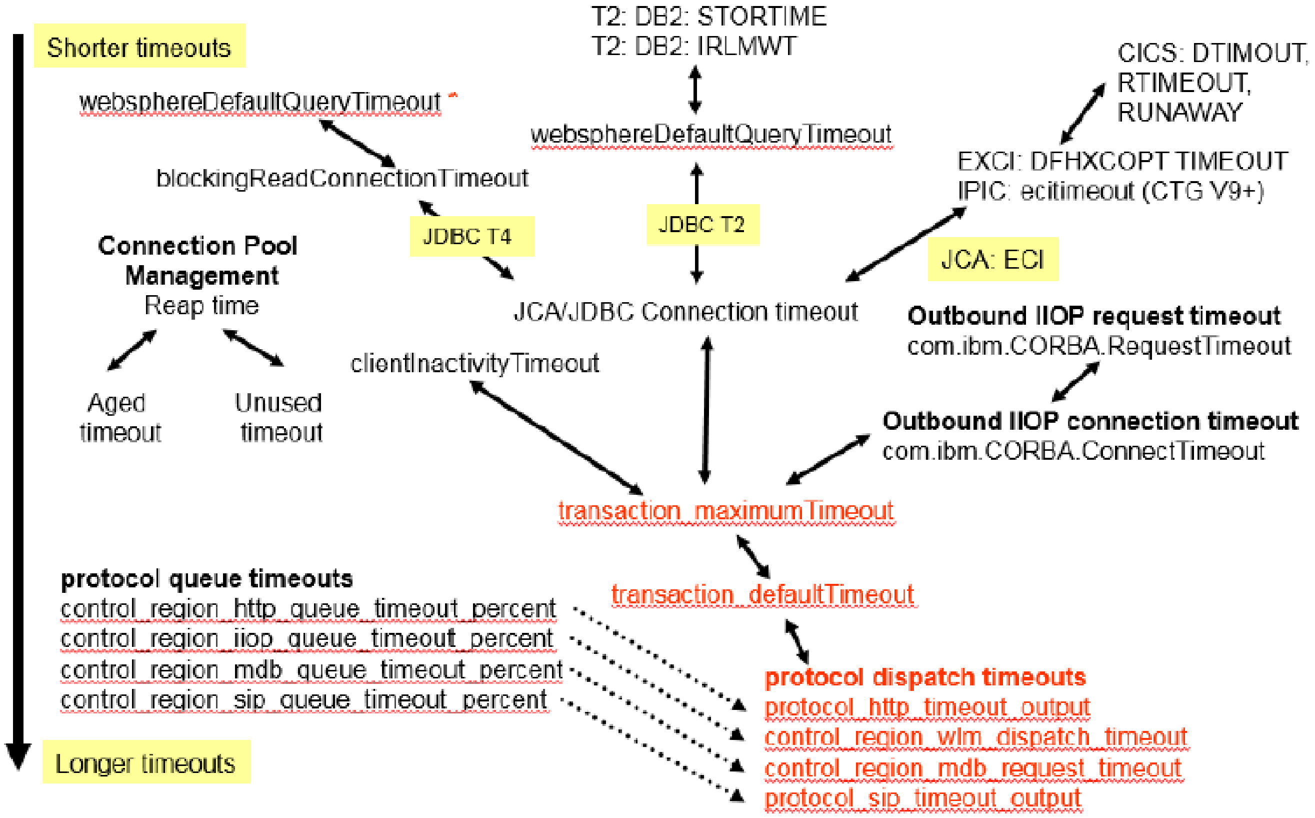
APAR PI40209 is currently targeted for inclusion in Fix Packs
8.0.0.11 and 8.5.5.7 of WebSphere Application Server.

References

Resources for Further Learning

WebSphere Application Server z/OS Timeout Management

http://www.ibm.com/support/techdocs/atsmastr.nsf/WebIndex/WP102510



Shorter timeouts

T2: DB2: STORTIME
T2: DB2: IRLMWT

CICS: DTIMOUT,
RTIMEOUT,
RUNAWAY

webSphereDefaultQueryTimeout *

webSphereDefaultQueryTimeout

EXCI: DFHXCOPT TIMEOUT
IPIC: ecitimeout (CTG V9+)

blockingReadConnectionTimeout

JDBC T4

JDBC T2

JCA: ECI

Connection Pool
Management
Reap time

Aged
timeout

Unused
timeout

JCA/JDBC Connection timeout

Outbound IIOP request timeout
com.ibm.CORBA.RequestTimeout

clientInactivityTimeout

Outbound IIOP connection timeout
com.ibm.CORBA.ConnectTimeout

transaction_maximumTimeout

transaction_defaultTimeout

protocol queue timeouts
control_region_http_queue_timeout_percent
control_region_iiop_queue_timeout_percent
control_region_mdb_queue_timeout_percent
control_region_sip_queue_timeout_percent

protocol dispatch timeouts
protocol_http_timeout_output
control_region_wlm_dispatch_timeout
control_region_mdb_request_timeout
protocol_sip_timeout_output

Longer timeouts

An incredibly rich source of detailed information on WAS z/OS timers and timeout patterns