The WAS z/OS team in ATS consists of John Hutchinson, Mike Kearney, Mike Loos, Louis Wilen, Lee-Win Tai and Don Bagwell. Glenn Fisher manages the team.

Mike Cox, Distinguished Engineer, serves as technical consultant to all our activities.
Table of Contents
Overview................................................................................................................................................4
WAS z/OS Thread Tuning Considerations...........................................................................................5
  1 Controller Region..........................................................................................................................5
  2 z/OS WLM Work Queue..............................................................................................................5
  3 Dispatch Threads in the Servant Region.......................................................................................5
  4 Number of Processors...................................................................................................................6
  5 Nature of the Application................................................................................................................6
  6 Maximum Open Connections.......................................................................................................7
  7 Multiple Servants..........................................................................................................................7
  8 Front-end HTTP Handling..............................................................................................................7
Overall Conclusion................................................................................................................................9
Document Change History....................................................................................................................10
Overview

We often are asked about the relationship between HTTP clients and thread tuning on WAS z/OS. This question usually comes up in the context of trying to address a performance issue. And the question is often phrased in the form of "How do I add more threads to a WAS z/OS server?"

The short answer is "Adding more threads may not be the answer. The thread model for WAS z/OS is different than on WAS for the distributed platforms."

The reason it is different is because WAS z/OS has a "multi-JVM" model (separate control region and servant region) with z/OS WLM between the Controller and the Servant. This makes the receipt of inbound HTTP asynchronous from the dispatch to the execution threads.

This short document will provide a technical overview of the WAS z/OS thread environment and how to approach tuning with respect to threads.

The following picture is used as a guide to the discussion that follows:
WAS z/OS Thread Tuning Considerations

1 Controller Region
This hosts the HTTP listener ports. The Controller Region (CR) has a thread pool related to initial handling of the inbound request, but it **unrelated** to the dispatch threads.

When a request is received the CR dispatches the request to a thread in the CR and it very quickly handles the request and places it on the z/OS WLM queue. The CR thread is then free to handle the next request.

**Note:** The CR uses the authorized ASYNC I/O interface to z/OS TCP/IP. This allows the CR threads to handle I/O requests very efficiently for HTTP and IIOP requests. This is what allows us to have a relatively small number of threads handle a large number of requests and a large number of connections.

We have seen WebSphere control regions with over 40,000 long lived (many hours long) SSL connections. One benchmark test validated the ability to have 125,000 concurrent sockets connected to one CR.

Work placed on the z/OS WLM work queue will get dispatched to a thread in the SR (see notes 2 and 3). If no thread is available the work remains queued in WLM.

The threads in the CR are unrelated to the dispatch threads in the Servant Region because of the z/OS WLM work queue. That makes request handling in the CR asynchronous from request handling in the SR.

**Key Point:** Do not confuse CR threads with SR dispatch threads because they are not the same thing.

We have very rarely seen cases where the CR thread pool needed increasing. The default value of 25 threads in the CR has proved more than enough in nearly every case we’ve seen.

2 z/OS WLM Work Queue
This provides the decoupling of the request received in the CR from the dispatch to the ORB thread in the servant region. This is what makes the received request asynchronous from dispatch to worker thread.

This is exclusive to WAS z/OS. This is what makes managing thread pools different than on distributed WAS. Requests that can not be immediately handled by a thread in the application JVM are "parked" here until a dispatch thread opens up to take the work. The depth of the WLM work queues dynamically expand and contract to handle temporary work spikes.

More than one WLM work queue exists between the CR and the SR, but that level of detail is outside the scope of this discussion. If you wish to know more, see WP101740 at ibm.com/support/techdocs.

3 Dispatch Threads in the Servant Region
These are the threads on which the request is dispatched and the work is processed.

The number of dispatch threads a Servant Region has a function of what's defined for the "z/OS Workload Profile," which is found under the server's "ORB service" and then "z/OS additional settings".

Four pre-defined profiles are provided, and starting with V7 a profile of "Custom" that allows you to specify the number of threads.

- **ISOLATE** = 1 thread for the servant region
- **IOBOUND** = 3 times the number of processors²

---
² There's a bit more math involved. Three times the number of processors is a good rule of thumb. In reality it is "3X CPU" or "5", whichever is the bigger number; and if that number is larger than 30, then 30 is the operative number of threads.
• CPUBOUND = the number of processors

• LONGWAIT = 40 threads

• CUSTOM = based on the value provided for the "servant_region_custom_thread_count" environment variable

Key Point: More threads is not necessarily better. It depends on the number of processors (note 4) and the nature of the application (note 5).

Number of Processors

The maximum number of threads that can be dispatched to CPU at the same time is equal to the number of processors.

For example, that's why the "CPUBOUND" profile limited threads to the number of processors. If the nature of the application called for intensive CPU-bound processing, then having more threads does no good -- if all the processors are occupied with CPU-bound work, additional threads will simply wait.

Key Point: With distributed WAS additional threads are used to "park" excess work, but on WAS z/OS that is not needed. WAS z/OS has the WLM work queues which is where excess work is "parked." When a thread opens in the servant region, WLM dispatches the work from the queue to the thread.

The "IOBOUND" profile was intended for applications that make calls elsewhere (to CICS, DB2, or a call to another JVM). Once the call is made the thread is freed from the CPU. But the work unit is not yet done, so the thread is still occupied. It's just not using the CPU. Therefore, more threads are needed than processors.

Is 3X the appropriate number? It depends on the nature of the application and how long the application on average spends waiting for the I/O to complete. The 3X number was determined to be a good number on average for the fixed profile "IOBOUND."

If the applications have longer wait times then the "LONGWAIT" profile may be more appropriate. That provides a fixed value of 40 threads.

Finally, the "CUSTOM" value allows you to set up to 500 threads if running in 64-bit mode.

Key Points:
• More is not necessarily better. It is a relationship between the number of processors and the proportion of time the request spends on the CPU vs. waiting for an I/O call to complete.
• Requests that arrive in excess of what the servant region can handle are queued in WLM. They are only dispatched to the servant when a thread in the servant frees to take work.

Nature of the Application

It is very important to understand what the application is doing, and in particular what portion of the time on the servant thread is spent on the CPU vs. waiting for some a request to some backend process to complete.

This understanding can be arrived at by understanding the application design, or by taking Java Core dumps at periodic intervals and using available tools to evaluate the state of the threads across the intervals.

Applications that spend the majority of their time in CPU without doing any other calls suggest more of a CPU-bound model. There is no good reason to have more threads than processors because when all processors are occupied the other threads will do unproductive waiting for CPU.

3 In reality it is “Number of CPUs minus 1” or “3”, whichever is the bigger number.
4 The default is 40 threads; the maximum you may specify is 100 if the server is in 31-bit mode, 500 if in 64-bit mode.
5 A very large number we do not expect many -- if any -- to actually use or need.
Applications that spend a minority of time in CPU relative to time waiting on a response to some I/O call suggest more of a IOBOUND or LONGWAIT model. Here is makes some sense to have more threads than processors because it allows more concurrent requests to begin the request processing and then wait for the I/O response to come back.

But configuring more threads does not necessarily mean more work accomplished -- depending on the overall architecture design is is very possible to have too many requests to backend systems going at once. If the backend data system is overwhelmed, then more waiting threads in the WAS servant region does nothing to improve throughput. The answer in that case is better tuning of the backend systems that are overwhelmed.

**Key Point:** Over-configuring dispatch threads in the WAS servant when the inhibitor is really the backend I/O systems may lead to dispatch or transaction timeouts caused by threads waiting too long for the response.

Timeouts that occur after dispatch to the servant threads incurs more overhead than timeouts that occur closer to the client.

It is far better to reject clients from entering an overloaded system than to accept them and time them out when dispatched and in the middle of executing the request transaction.

---

**6 Maximum Open Connections**

This is merely the number of open TCP connections permitted on this TCP channel for this server.

It is not directly related to the number of dispatch threads in the servant.

How many is needed is a function of the number of clients facing the server and the nature of average connection. It is also influenced by the nature of any front-end devices (note 8) the clients pass through.

Clients that make very short-lived connections intermittently may not require any change to the default value of 20,000.

Clients that make a connection and hold onto it for a long time may require a change to this.

If client TCP connections are terminated at some front-end device, and a fewer number of long-lived connections are created from front-end device to WAS z/OS server, then the default number if likely sufficient.

---

**7 Multiple Servants**

It is possible to configure WAS z/OS to host multiple servant regions behind the controller region.

Going from one servant to two servants doubles the configured dispatch threads available. But it does not double the number of processors on the LPAR, nor does it double the capacity of any backend systems the application will call. So increasing the number of servants does not necessarily increase the throughput. It can increase the throughput based on the capacity of other components of the system, but it does not by itself increase throughput.

Multiple servant regions provides two key advantages

1. Higher availability ... separate JVMs in separate address spaces with automatic WLM restart of failed regions
2. Increased total heap as each servant has its own configured JVM heap to hold objects created by the application.

**Key Point:** Increasing the number of servant regions will not necessarily increase throughput. It is better to have a good understanding of why you wish to increase servant regions before you do so. Make sure you have the system capacity to support the additional memory requirements, and make sure the backend systems being called are capable of handling the additional work that results.
Front-end HTTP Handling

The existence of front-end HTTP handling functions (such as HTTP Servers running the WAS Plugin, or the Proxy Server, or DataPower devices, or other solutions) can have a bearing on some of these tuning properties:

- If they are used to spray requests across a WAS z/OS cluster then the total clients end up being split across cluster members. The number of clients presenting themselves to any given cluster member is a fraction of the total.
- They can be used to terminate the TCP (and possibly even the SSL) connection from the client and present fewer and more long-lived connections back to the WAS z/OS server.
- They can be used to serve content from the edge, thus removing work from the WAS z/OS server that might otherwise occupy threads to serve up static content.
- They can be used to pre-process security function or pre-process XML validation, thus removing some portion of error handling work from the WAS z/OS servers.

But it's important to keep in mind that the rules we spelled out before still apply for that work which is presented to the CR from the front-end device. Or said another way, if the request flows from the front-end device and is received by the CR, then everything we've said to this point still applies.
Overall Conclusion
Here we offer a bullet summary of the points made earlier in the document:

- The threads in the CR are **not** related to the dispatch threads in the servant region.
- The CR very quickly handles requests and queues to the WLM work queue. The CR threads pool is almost never the bottleneck.
- The WLM work queue is what provides the asynchronous de-coupling between receipt of request by CR and the dispatch of work to the worker thread in the servant region.
- Work in the WLM work queue is never dispatched to a servant until a thread in the servant is free to take it. WAS z/OS works on a "pull" model rather than a "push."
- Temporary periods where requests exceed ability to process results in an expansion of the WLM work queue.
- As servant region(s) are able to pull the work and process the WLM work queue depth decreases.
- The WLM work queue is what provides a place to "park" work that has no place to be executed. configuring extra threads to "park" excess work is **not** needed with WAS z/OS.
- There is a **balance** to be achieved between the number of dispatch threads in the servant, the number of CPUs in the LPAR and the nature of the application. More threads is **not** necessarily better.
- There is a **balance** to be achieved between the number of requests handled by WAS servant threads and the ability of backend systems to handle the calls from those threads. Adding more threads to the WAS z/OS applications making calls to an overloaded DB2 or CICS environment does not increase throughput.
- If client requests exceed the ability of the overall system to handle the work, then it is better to reject the client request as close to the client as possible. Configuring more threads in WAS has the potential of (a) creating more overhead handling the request further into the architecture, and (b) incurring more overhead handling the errors that result from request and transaction timeouts that may occur if a request is started but can not be completed.
### Document Change History

Check the date in the footer of the document for the version of the document.

<table>
<thead>
<tr>
<th>Date</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>April 18, 2013</td>
<td>Original document</td>
</tr>
<tr>
<td>April 24, 2013</td>
<td>Updated threads possible with CUSTOM workload profile. Original document indicated 100, but the actual value when in 64-bit mode is 500.</td>
</tr>
<tr>
<td>May 30, 2013</td>
<td>Added a footnote to the IOBOUND and CPUBOUND workload profile descriptions. The number of threads for each described in this document is a bit of an oversimplification. A bit more insight into the actual math used was provided.</td>
</tr>
</tbody>
</table>