

Troubleshooting WebLogic 10.3.x Issues

06/07/2011

ORACLE®

Troubleshooting WebLogic 10.3.x Issues

Contents

Introduction	3
WebLogic Overview	4
Types of Memory used by WebLogic	4
WebLogic Garbage Collection	6
WebLogic Threads.....	7
Where to Start with Troubleshooting WebLogic Issues?	8
Most Common WebLogic Issues.....	10
Issue#1: WebLogic is Out of Memory	11
Issue#2: Database/App Server Hold-Ups causing Requests to Queue up on Web Server	14
Issue#3: Issues with Session Stickiness.....	16
WebLogic Tools for Monitoring and Troubleshooting	19
Monitoring WebLogic Memory Usage	19
Monitoring WebLogic Sessions.....	23
Monitoring WebLogic Threads	24
Enabling HTTP Access Logs.....	25
Monitoring the Tuxedo Application Server.....	26
Creating/Analyzing Thread Dumps	28

Introduction

This document provides information on troubleshooting WebLogic 10.3.x issues, with emphasis on hangs, crashes and performance issues. This document is intended for PeopleTools environments using WebLogic 10.3.x. But much of this information applies to other WebLogic versions as well.

This information is presented in the following sections:

- Section “WebLogic Overview”: provides an overview of WebLogic 10.3 memory usage/management and thread usage
- Section “Where to Start with Troubleshooting WebLogic issues?”: provide suggestions on first steps to take to troubleshoot and identify issues in your WebLogic environment.
- Section “Most Common WebLogic Issues” provides a list of the most common WebLogic issues. For each issue, there is a detailed description of the issue, symptoms and recommended solution.
- Section “WebLogic Tools for Monitoring and Troubleshooting” contains a list of tools that can be used to assist with monitoring WebLogic and troubleshooting WebLogic issues.

WebLogic Overview

This section gives you some background information about types of memory used by WebLogic, how memory is managed, and thread usage. This may be helpful to understand, when you are researching memory and/or thread related issues in your WebLogic environment

Types of Memory used by WebLogic

There are two types of memory used by WebLogic:

1. **Java Heap:** the memory that the JVM (Java Virtual machine) uses to store objects used by Java programs
2. **Native Heap:** this is another heap, allocated outside of the java heap, which the JVM uses for its own internal operations. Thread stacks are also stored in native heap.

Below are more details on “Java Heap Memory” memory and “Native Heap Memory”

Java Heap:

The Java heap size is configurable by using these java parameters:

- 1) -Xms = minimum heap size (initial heap size)
- 2) -Xmx = maximum heap size

Setting the JVM Xms/Xmx parameters to the same value is recommended to avoid the performance hit incurred by dynamically growing the JVM. This also improves predictability and lessens the frequency of JVM garbage collection.

Note that the memory for the Java Heap, is pre-allocated, when you start the WebLogic PIA. So for example, if your Xms/Xmx java heap setting is set to 1 GB, then 1 GB of heap will be set aside when you start the WebLogic PIA.

Refer to the following document for instructions on how to adjust the java heap setting:

[Doc ID 638298.1: How To Increase/Decrease JVM Heap Size for WebLogic](#)

Native Heap:

The maximum amount of native heap is limited by the virtual process size limitation on any given OS and the amount of memory that is already committed for the java heap size with the -Xmx flag. For example, if the application can allocate a maximum of 2 GB for the java process and the max java heap is 1.5 GB, then the max possible native memory is approximately 500MG. There is very little tuning that can be performed for native memory.

Types of Memory Used by WebLogic, cont'd

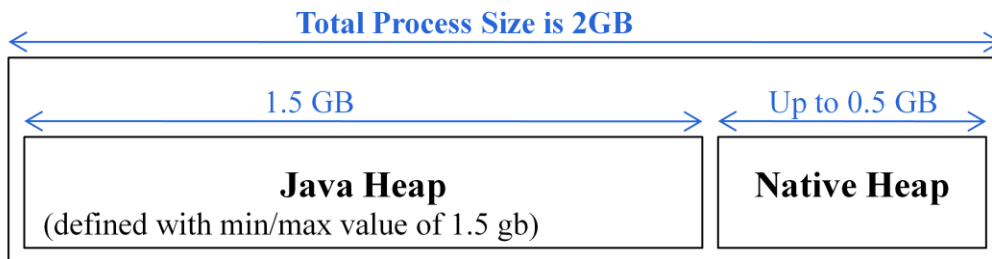
Process size – Process size will be the sum of the java heap, native memory and the memory occupied by the loaded executables and libraries. On 32-bit operating systems, the virtual address space of a process can go up to 4 GB. Out of this 4 GB, the OS kernel reserves some part for itself (typically 1 – 2 GB). The remaining is available for the application. For example, in Windows – by default, 2 GB is available for the application and 2 GB is reserved for Kernel's use (with the exception of some variants of Windows). For other operating systems, please refer to the OS documentation for your configuration.

Out of memory in the native heap usually happens when the process reaches the process size limitation on that OS, or the machine runs out of RAM and swap space. In this case, the JVM will exit and may generate a core file when it gets a sigabrt signal. If the machine doesn't have enough RAM and swap space, then the OS will not be able to give more memory to this process that could also result in out of memory. Make sure that the sum of RAM and swap space in the disk is sufficient to cater to all the running processes in that machine.

The example below shows us how memory might be allocated in a situation where you can have a maximum process size of 2GB. In this example, the java heap was defined to be 1.5GB, which leaves 500 MB left over for native heap. If more than 1.5GB of java heap is needed or if more than ½ GB of native heap is needed, an “out of memory” error would be issued.

Note that starting with PeopleTools 8.51, WebLogic uses 64-bit java (for PeopleTools 8.50 and lower versions, 32-bit java is used). So starting with PeopleTools 8.51, WebLogic does not have the address space limitations that you have with 32-bit java, thus will have the ability to access/use larger segments of memory for java heap and native heap (provided that the server has plenty of memory available)

Example of WebLogic Java Process

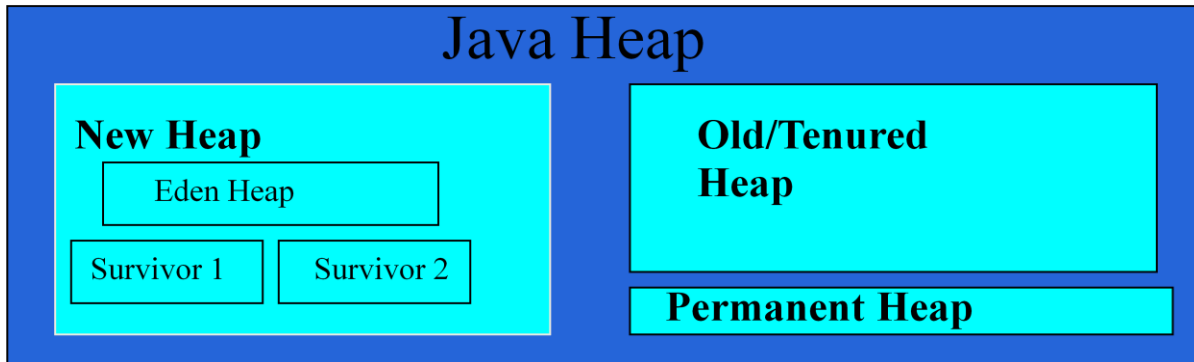


In situations where your WebLogic environment is reporting “Out of Memory” errors, it means that WebLogic is having issues with either the “Java heap” memory or “Native heap”. So you will need to tune the environment accordingly. (more details in section “Most Common WebLogic Issues: Issue#1: WebLogic is Out of Memory”)

WebLogic Garbage Collection

Periodically, a java process called the "garbage collector" (GC) runs and it moves or frees any objects in the java heap that are no longer being used so that memory is reclaimed.

There are different types of memory within the java heap as illustrated below



There are two types of garbage collection (GC) that take place:

1. A **full GC** runs over the entire heap to clean up memory. It typically takes no more than 3-5 seconds. The JVM cannot service user requests during full GC.
2. A **minor GC** moves/cleans objects within the 'new heap' and takes less than 2 milliseconds (can happen every 3-10 seconds)

If the heap is totally consumed, and the garbage collector cannot reclaim any memory, then a WebLogic out-of-memory error is recorded to the WebLogic log, and the server usually hangs.

There are tools that can be used to monitor heap utilization and how often garbage collection occurs. These tools may be helpful in situations where you are having issues with running out of java heap memory (more details in section "Monitoring WebLogic Memory Usage")

WebLogic Threads

When a request is submitted to the web server, a thread is created. The thread is not released until the request has been completed.

In older WebLogic releases (eg WebLogic 8.1), it was necessary to define the maximum thread count in the WebLogic configuration and all threads were allocated when the web server was started.

However, starting with WebLogic 9.2, there is a 'self tuned thread pool' and WebLogic 9.2 and 10.3 automatically increases/decreases available threads depending on demand. Thus there is no need to adjust the thread count for your WebLogic 9.2 and WebLogic 10.3 environment.

Refer to the following knowledge document for more details:

[Doc# 660080.1: Managing Threads in WebLogic Server 9.2 and 10.3](#)

There are different tools that can be used to monitor thread usage in your WebLogic environment. These tools may be helpful in situations where you are having performance issues or issues with stuck threads. (more details in section “Monitoring WebLogic Threads”)

Where to Start with Troubleshooting WebLogic Issues?

Below are some tips on how to begin troubleshooting issues in your environment

Check for Warning/Error Messages in the WebLogic log:

Typically when you are having WebLogic-related issues, there are warning/error messages in the WebLogic logs. So you should check the WebLogic log as these messages will give you clues as to cause of problem (the logs are usually the first thing that Oracle Support Engineers check, in order to learn more about the problem)

Location of WebLogic Logs

The web server logs are located in directory:

<PS_HOME>/webserv/<DOMAIN_NAME>/servers/PIA/logs

For Unix/Linux, log information is stored in these log files:

- PIA_weblogic.log
- PIA_stderr.log
- PIA_stdout.log

For Windows, log information is stored in these log files:

- PIA_weblogic.log
- NTservice-<domain_name>-PIA.log

File “PIA_weblogic.log” is of most interest, but you should check for errors in other log files as well.

How to analyze WebLogic Log Files

The log messages are prefixed with a date/time stamp. So check the logs to see if any errors or warning messages are being logged when issue occurs (you may want to also review the messages just prior to the start of issue, to see if there are any messages issued leading up to the time of the problem).

For file PIA_weblogic.log, you may want to search for the following strings, as these will help you identify the most common issues encountered in PeopleSoft environments:

- “OutOfMemory”
Any type of memory-related error will typically contain string “OutOfMemory”. Insufficient memory is a common WebLogic issue, and can cause hangs/crashes. So if you are experiencing WebLogic hangs or crashes, it is important to check your log to see if this problem is occurring in your environment. If you notice any OutOfMemory errors, refer to the ‘Out of Memory’ section in “Most Common WebLogic Issues” for details on how to address this problem
- “[STUCK]”
When a transaction is running more than 5 minutes, a message (example below) is logged to the PIA_weblogic.log.
<Apr 18, 2011 12:47:04 PM PDT> <Error> <WebLogicServer> <BEA-000337> <[STUCK]
ExecuteThread: '4' for queue: 'weblogic.kernel.Default (self-tuning)' has been busy for "675" seconds working on the request

Note that the message shows that the thread is 'STUCK'. But in fact, the thread may not be stuck, but is just taking a long time to complete. These threads often successfully complete, if given enough time.

If you see a lot of long-running threads, at the time users are experiencing problem, then this indicates that the web server is having issues processing threads, which may cause the web server to hang.

Long running threads can be caused by different issues. The problem often occurs due to issues on the app server or database subsequently causing the threads to queue up and wait on the web server.

So if you see a lot of stuck threads, you may want to troubleshoot further by doing the following:

1. Get a thread dump as described in section "Creating/Analyzing Thread Dumps". The thread dump may help you determine whether the threads are getting hung up on the app server or database.
2. Have your DBA check for long running SQL's and/or DB locks
3. You can also look at the 'Stuck' thread messages in the PIA_weblogic.log to see what user(s) are running the transactions and the specific component they are running. This may help you determine if there is a specific user and/or transaction that is causing the problems.

- 'javacore' or 'dump'

If the web server crashes (ie the WebLogic java process is terminated), the log directory may also contain a java dump file with more details. The file will contain reference to 'javacore' or 'dump' (depending on type of OS)

Other Troubleshooting Tips:

Usually the logs will provide sufficient clues to start your investigation into the issue. However, if there is no relevant information in the logs, you may need to use other tools to collect more information.

For example, if the web server is hanging or performance is poor, you can collect a thread dump at time of problem, to find out what the threads are doing (see section "Creating/Analyzing Thread Dumps" in the "Monitoring Tools" chapter)

You may also want to view "Most Common WebLogic Issues" to see if any of these problems match to the symptoms you are experiencing in your PeopleSoft environment.

Most Common WebLogic Issues

Summary of Issues

The most common causes for WebLogic PIA issues are:

- The WebLogic PIA is out of memory
- Issues on the Database or App Server are causing requests to queue up and wait on Web Server
- Session Stickiness Issues

The user experience, when there are WebLogic-related issues, may include any of the following:

- Internal Server Error 500
- Session Hangs
- Slow Performance
- Getting “kicked out” of session and brought to signon page

The following sections cover each of the most common causes for WebLogic-related issues. Each section includes:

- Description of problem
- Symptoms from Logs and Monitoring Tools
- End-User Symptoms
- How to fix the issue

Issue#1: WebLogic is Out of Memory

Description of Problem:

WebLogic uses two types of memory:

- 1) Java heap memory
- 2) Native heap memory

(Refer to section "Types of Memory used by WebLogic" in the "WebLogic Overview" chapter for more details)

If not enough memory is available for java heap or native heap, WebLogic will momentarily hang as it does not have memory needed to process incoming requests..

Out of Memory errors are usually easy to identify because WebLogic logs an "OutOfMemory" error to PIA_weblogic.log

Most (but not all) "out of memory" errors occur when there is a high load on the PeopleSoft environment. The WebLogic PIA often recovers from the out-of-memory issue on its own (when load is reduced and more memory is freed up). Although in some cases, the WebLogic java process crashes (ie the WebLogic java process terminates) and the WebLogic PIA must be restarted

Symptoms from Logs and Monitoring Tools:

Logs

WebLogic file "PIA_weblogic.log" shows "OutOfMemory" error. The exact message may vary, but it will always contain string "OutOfMemory". Below are a few examples of messages in PIA_weblogic.log:

Example#1

```
####<May 13, 2011 4:30:28 PM EST> <Error> <HTTP> <mywebsvr> <PIA> <[ACTIVE] ExecuteThread: '16' for  
queue: 'weblogic.kernel.Default (self-tuning)'> <<WLS Kernel>> <> <> <1292275828134> <BEA-101017>  
<[ServletContext@921319146[app:peoplesoft module:/ path: spec-version:2.5]] Root cause of ServletException.  
java.lang.OutOfMemoryError
```

Example#2

```
####<Feb 25, 2011 2:16:39 PM EST> <Error> <HTTP> <mywebsvr> <PIA4> <[ACTIVE] ExecuteThread: '4' for  
queue: 'weblogic.kernel.Default (self-tuning)'> <<WLS Kernel>> <> <> <1298661399989> <BEA-101017>  
<[ServletContext@4382585[app:peoplesoft module:/ path: spec-version:2.5]] Root cause of ServletException.  
java.lang.OutOfMemoryError: unable to create new native thread  
    at java.lang.Thread.start0(Native Method)  
    at java.lang.Thread.start(Thread.java:640)
```

Example#3

```
May 23, 2011 8:59:23 AM CDT> <Error> <HTTP> <BEA-101017> <[ServletContext@16918822[app:peoplesoft  
module:/ path: spec-version:2.5]] Root cause of ServletException.  
java.lang.OutOfMemoryError: allocLargeObjectOrArray: [B, size 38848  
    at java.util.Arrays.copyOf(Arrays.java:2786)  
    at java.io.ByteArrayOutputStream.write(ByteArrayOutputStream.java:94)  
    at java.io.DataOutputStream.write(DataOutputStream.java:90)  
    at bea.jolt.SOutputBuffer.add(SOutputBuffer.java:376)  
    at bea.jolt.JoltRemoteService.encodeCALL(JoltRemoteService.java:413)  
Truncated. see log file for complete stacktrace
```

If you are monitoring memory (see section "Monitoring WebLogic Memory Usage"), you may see that JVM is frequently garbage collecting, yet unable to free up any heap space.

Issue#1: WebLogic is Out of Memory (cont'd)

End-User Symptoms:

- Users will often get an “Error 500—Internal Server Error” when Java is out of memory
- Also, sessions will often hang.
- The web server often recovers on its own, so the end users may experience the issues for just a brief period of time. However, in some cases, the out of memory error will cause the WebLogic process to terminate. In that case, users will get “Page Cannot Be Displayed Errors” until the WebLogic PIA is restarted.

How to Fix the Issue:

Memory related issues can be caused by many problems including:

- Java Heap is undersized for the environment
- There is not enough native memory available for the java process
- Web server is overloaded

Below are some general guidelines on how to address memory-related issues. In addition, you should search the “My Oracle Support” knowledge base for the specific “OutOfMemory” error message that you see in your PIA_weblogic.log.

Also, you can collect further details on WebLogic memory usage by using monitoring tools referenced in section “Monitoring WebLogic Memory Usage”

General Guidelines on Fixing OutOfMemory Issues

1. First, you need to determine if WebLogic is running out of “native heap” memory or “java heap” memory. Typically you are able to tell this by checking the “OutOfMemory” error message in the PIA_weblogic.log:
 - a. If the error message refers to “native” or to a “thread” related error, it is an issue with “native memory”. Examples of errors due to running out of native memory are:
 - “Unable to create new native thread”
 - “Error starting thread: Not enough storage is available”Native memory errors are more likely to occur on PeopleTools 8.50 and lower versions where you are running a 32-bit java process (which has address space limitations)
 - b. Any other error messages, are usually due to running out of java heap memory
2. If you are running out of java heap, you may want to start by increasing the java heap settings (if you are unable to increase java heap setting, then the other option is to add more WebLogic PIA’s to the environment).

Increase the java heap setting as follows:

- a. First check the current heap setting by searching string “-Xmx” in your WebLogic log (for Unix, search PIA_stdout.log. For Windows, search NTservice-<DOMAIN_NAME>-PIA.log). The “-Xmx” value shows you the current heap setting. For example, these settings show that the minimum heap (-Xms) and maximum heap (-Xmx) are set to 512mg:
Java command line=java -server -Xms512m -Xmx512m
- b. For PeopleTools 8.50, try increasing the heap (at 256mg increments) up to 1.5gb. For PeopleTools 8.51, you can increase the heap even higher, provided the server has enough memory available.

- For Unix, you can change the heap setting in file setEnv.sh. Example:
`JAVA_OPTIONS_LINUX="-jrockit -XnoOpt -XXnoJITInline -Xms768m -Xmx768m`
- For Windows, you will either need to change the setting in the Windows registry, or else change setting in setEnv.cmd and then rebuild the Windows Service

Refer to the following document for more details on changing the java heap setting:

[Doc#638298.1: How To Increase/Decrease JVM Heap Size for WebLogic](#)

3. If you are running out of native memory heap, then you may want to consider doing the following:
 - a. Lower the java heap setting (ie Xmx/Xms settings) in order to allow more of the java process' memory to be used for "Native Memory". (see step 2b above, for instructions on changing java heap setting)
If the java heap is already being fully utilized, and you are unable to lower it, then you may want to consider adding additional PIA's to your environment
 - b. Lower the thread stack size. Note that the threads use native memory, so if you lower the thread stack size, then the threads will not consume as much memory. The thread stack size is specified using parameter "-Xss". Refer to the following document for details
[651285.1: WebLogic Error: "java.lang.OutOfMemoryError: unable to create new native thread"](#)

Issue#2: Database/App Server Hold-Ups causing Requests to Queue up on Web Server

Description of Problem:

If the app server is performing slowly or if the database is busy due to long running sql's and/or DB locks, then the web server threads begin to queue up. This may cause web server performance issues and may eventually bring the web server to a halt. And in some cases, the web server may run out of memory due to the extra memory required to store thread information and session information for all the threads that are queued up

Symptoms from Logs and Monitoring Tools:

WebLogic Logs

You will often see "stuck thread" messages in PIA_weblogic.log. Example:

```
<Apr 18, 2011 12:47:04 PM PDT> <Error> <WebLogicServer> <BEA-000337> <[STUCK] ExecuteThread: '4' for  
queue: 'weblogic.kernel.Default (self-tuning)' has been busy for "675" seconds working on the request .....
```

These messages are issued if a thread has been processing more than 10 minutes. And one of the reasons that threads may take longer to process is when they are queued up waiting on the app server/database to finish processing the request.

Thread Dump

If you get a thread dump (see section "Creating/Analyzing Thread Dumps"), you will notice that many threads show this call stack (which indicates the thread is waiting on the app server to finish the request:

```
"[ACTIVE] ExecuteThread: '2' for queue: 'weblogic.kernel.Default (self-tuning)'" id=35 idx=0x80 tid=2304 prio=5 alive,  
waiting, native_blocked, daemon
```

```
-- Waiting for notification on: bea/jolt/IOBuf@0x11730730[fat lock]  
at jrockit/vm/Threads.waitForNotifySignal(JLjava/lang/Object;)Z(Native Method)  
at java/lang/Object.wait(J)V(Native Method)  
at bea/jolt/IOBuf.waitOnBuf(IOBuf.java:119)  
^-- Lock released while waiting: bea/jolt/IOBuf@0x11730730[fat lock]  
at bea/jolt/NwHdlr.recv(NwHdlr.java:1564)  
at bea/jolt/CMgr.recv(CMgr.java:185)
```

psadmin

If you monitor the application server using "psadmin" (see section "Monitoring the Tuxedo Application Server"), you may see many busy requests and you may also see requests queuing up

Tuxedo Logs

The TUXLOG may show a large number of SVCTIMEOUTs if there are problems on the database.

End-User Symptoms:

End users will notice slow performance and or session hangs

Issue#2: Database/App Server Hold-Ups causing Requests to Queue up on Web Server **(cont'd)**

How to Fix the Issue:

How you fix this issue is specific to the problem occurring in your environment:

- If the database is the source of the issue (long running queries or DB locks), you will need to get your DBA involved in researching the issue.
- If the App Server is the source of the issue, you may need to add more app server domains and/or psappsrv processes. It is beneficial to set up multiple domains for application server load balancing and failover. This will distribute the load amongst separate Application Server domains (all pointing to the same database.) If one domain goes down or needs to be brought down, all users will failover to the other domain(s).

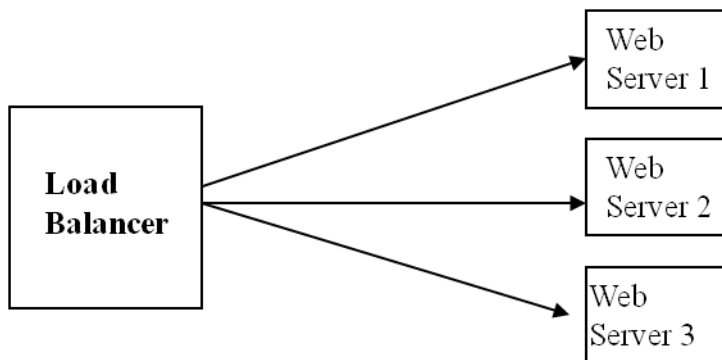
If you suspect that the issue is related to specific transactions that users are running, you can gather more details as follows:

1. Set “logfence=4” in psappsrv.cfg in order to log each user request (if you do not have "Allow Dynamic Changes" set to “Y”, you will need to go into PSADMIN and reload the configuration and restart the app server, in order to pick up the change)
2. After making the above change, file APPSRV_MMDD.log will show the transaction “elapsed time”. So you can look for transactions taking longer amounts of time.

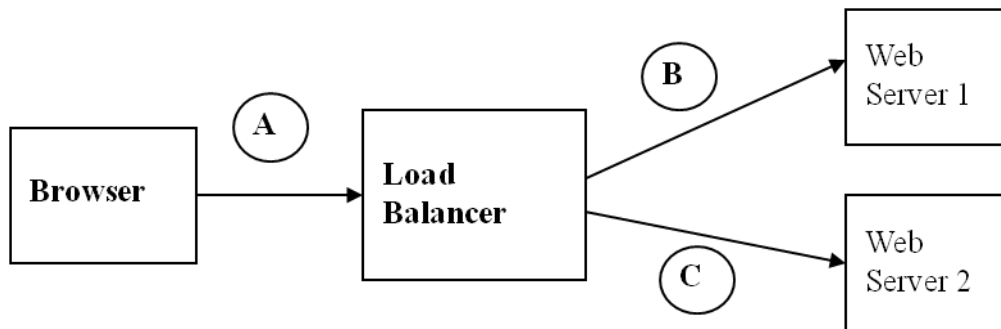
Issue#3: Issues with Session Stickiness

Description of Problem:

Session Stickiness, also referred to as session persistence, states that once a session is created by a user, all activity for that session, must go to the same web server. You need to be concerned about this only if you have two or more web servers. For example, in the diagram below, there are three web servers being used, so it is important that the environment is configured so that all traffic, from a single session, goes to the same web server



Below is an example where session stickiness does not exist.



In this example:

- User logs in from browser. The url references load balancer (LB), so the request is sent to the LB (see path A)
- The LB can then direct the request to Web Server 1 or 2. In this example, the request is directed to web server 1 (see path B).
- Web Server 1 then receives the request and stores the user's session information and processes the request
- User then clicks on a menu item
- The LB receives the request and this time it routes the request to Web Server 2 (see path C)
- Web Server 2 then receives the request. It does not have session information, so it stores the session information and attempts to process the request

Issue#3: Issues with Session Stickiness (cont'd)

Symptoms from Logs and Monitoring Tools:

For this particular issue, the WebLogic logs usually do not show any symptoms of the problem. However, there are many tools available to validate whether there are issues with session stickiness. Start by reviewing the following document which contains tips on identifying Session Stickiness Issues:

[Doc# 1307344.1: How To Identify Session Stickiness Problem On Load Balancer?](#)

In addition, refer to the following

WebLogic Console

If you are monitoring sessions from the WebLogic console (see section “Monitoring WebLogic Sessions”), you will see that many users have a session on multiple WebLogic PIA’s

HTTP Access Logging

If you have http access logging enabled (see section “Enabling HTTP Access Logs”), you will see http requests, for the same user session, being logged to multiple WebLogic PIA’s. For example, below are excerpts from two PIA_access.logs that shows issues with session stickiness as there are http requests from same user being logged to both servers at the exact same time and some of the requests are returning error '403' which is often observed when there are session stickiness issues

Excerpt from WebServer 1

DATE	TIME	C-IP	CS-METHOD	TIME-TAKEN	SC-STATUS	BYTES	CS-URI
05/21/2011	14:23:14	10.138.250.92	GET	0.281	200	731	
/ps/ps/EMPLOYEE/PT_LOCAL/h/?tab=DEFAULT&							
05/21/2011	14:23:14	10.138.250.92	GET	0	200	2501	/cs/ps/cache/PT_SAVEWARNINGSRIPT_1.js

Excerpt from WebServer 2

DATE	TIME	C-IP	CS-METHOD	TIME-TAKEN	SC-STATUS	BYTES	CS-URI
05/21/2011	14:23:14	10.138.250.92	GET	0	403	0	/cs/ps/cache/PSSTYLEDEF_1.css
05/21/2011	14:23:14	10.138.250.92	GET	0	403	0	/cs/ps/cache/PT_ISCROSSDOMAIN_1.js
05/21/2011	14:23:14	10.138.250.92	GET	0	403	0	/cs/ps/cache/PORTAL_REFRESHPAGE_1.js

Check Application Server Logs for Multiple Jolt Connections per User

Check the Application Server log (APPSRV_MMDD.LOG) for any indication of multiple authentication requests for the same user. There should be only one GetCertificate Request per login from the same client IP (unless the client IP/name is from a proxy or load balancer, in which case, the actual end-user's client IP/name cannot be determined). A scenario where a user is first authenticated for the PeopleSoft user/password and later followed by one or more “PeopleSoft Token authentication succeeded” messages would indicate a session stickiness problem. Subsequent “PeopleSoft Token authentications” should not occur because the first Get Certificate Request has already authenticated the user/password. “PeopleSoft Token authentications” are expected only in scenarios where content is pulled in by Portal.

For example, first GetCertificate request for PSJOBS from machine:

machine50.com:

PSAPPSRV.807128 (38) [08/21/08 10:31:04 GetCertificate](3) PeopleSoft ID and Password authentication succeeded for user PSJOB@machine50.com.

Notice that this GetCertificate request authenticates the PeopleSoft ID and Password PSJOB.

Subsequent GetCertificate for PSJOBS from same client machine:

PSAPPSRV.807128 (54) [08/21/08 10:31:54 GetCertificate](3) PeopleSoft Token authentication succeeded: PSJOB@machine50.com.

This indicates that that ID PSJOB has been authenticated more than once, which is a symptom of a stickiness issue.

End-User Symptoms:

End users may notice the following:

- Randomly kicked out of the PeopleSoft application and returned to search page or signon page
- Randomly get “Page is no longer available” message
- HTTP 403 errors

How to Fix the Issue:

Correcting session stickiness issues is dependent on your environment and what kind of Load Balancer (LB) or Reverse Proxy Server (RPS) you are using. Below are some things to keep in mind:

- If using a LB, you should configure the LB to use cookies
- The cookie name should be defined the same on the RPS/LB and the web servers
- Refer to the following document for more details:

[Doc# 653998.1 PeopleSoft and Load Balancers](#)

WebLogic Tools for Monitoring and Troubleshooting

Monitoring WebLogic Memory Usage

There are many different tools that can be used to monitor the WebLogic memory usage. In this section, we will cover the following tools:

1. Enabling logging of Verbose Garbage Collection (verboseGC)
2. Monitoring Memory usage from WebLogic console
3. Java tools and 3rd Party Tools

If you are not familiar with Java “Garbage Collection” and types of memory that are used by WebLogic, you may want to first review these two sections, in the “WebLogic Overview” chapter, before using these memory monitoring tools:

- Section “Types of Memory used by WebLogic”
- Section “WebLogic Garbage Collection”

Note that most of the monitoring tools covered in this section, are for monitoring the “java heap” (which is where we tend to see more issues)

Monitoring WebLogic Memory Usage (cont'd)

Monitoring WebLogic Memory Usage using Verbose Garbage Collection

One of the easiest ways to monitor memory usage over a period of time, is to enable “verbosegc”. When this is done, WebLogic logs java heap details every time the JVM cleans up the java heap (ie “Garbage Collects”).

How to Enable Verbose Garbage Collection:

Verbose Garbage Collection is enabled via a Java parameter when starting the WebLogic PIA. The Java parameter varies depending on the OS platform that WebLogic is on. Refer to the following document for details on how to enable Verbose Garbage Collection in your environment:

[Doc#759137.1: What is Verbose Garbage Collection and How do I Enable it on WebLogic?](#)

After you enable Verbose Garbage Collection (GC), information will be logged to one of the WebLogic log files (refer to above document to find out where the information will be logged for your OS).

How to Interpret Verbose GC Information:

When viewing the verbose garbage collection information, the main points of interest are:

- 1) How much heap is being freed up when the JVM runs a full GC?
- 2) How long is the GC taking?

The GC method and the way that verbose GC information is logged, will vary depending on the JVM you use. Below is an example showing GC information for JRockit JVM (which is used on Linux and Windows platforms):

By default, JRockit has a “Generational Garbage Collection” and each GC is either:

- 1) YC (Young Collection): This collect cleans up the “nursery” (ie heap reserved for new objects)
- 2) OC (Old Collection): This cleans up both old and new heap. It is a “full GC”

The format of the VerboseGC information, for JRockit is as follows:

[memory][Thu Jun 02 14:37:53 2011][07432] <start>-<end>: <type> <before>KB-><after>KB (<heap>KB), <time> ms, sum of pauses <pause> ms.

In the example below, an “Old Collection” (which includes a “Full GC”) was performed on June 2 at 3:42pm and reduced the heap size from 360mg to 97mg. This garbage collection took 0.8 seconds

[memory][Thu Jun 02 15:42:06 2011][07432] [OC#2] 3854.339-3854.422: OC 360945KB->97053KB (524288KB), 0.083 s, sum of pauses 80.192 ms, longest pause 80.192 ms.

In the example below, a “Young Collection” was performed on June 2 at 3:52pm and reduced the heap size from 335mg to 162mg. This garbage collection took 0.15 seconds

[memory][Thu Jun 02 15:52:36 2011][07432] [YC#8] 3944.309-3944.325: YC 335414KB->162704KB (524288KB), 0.016 s, sum of pauses 15.784 ms, longest pause 15.784 ms.

If you see frequent GC’s with very little heap getting reclaimed, this indicates that the JVM is in danger of running out of memory.

Also, if the GC is taking a long time (more than 10 seconds), this could also have impact on the user as no requests are processed when GC is occurring (ie “the world stops” momentarily).

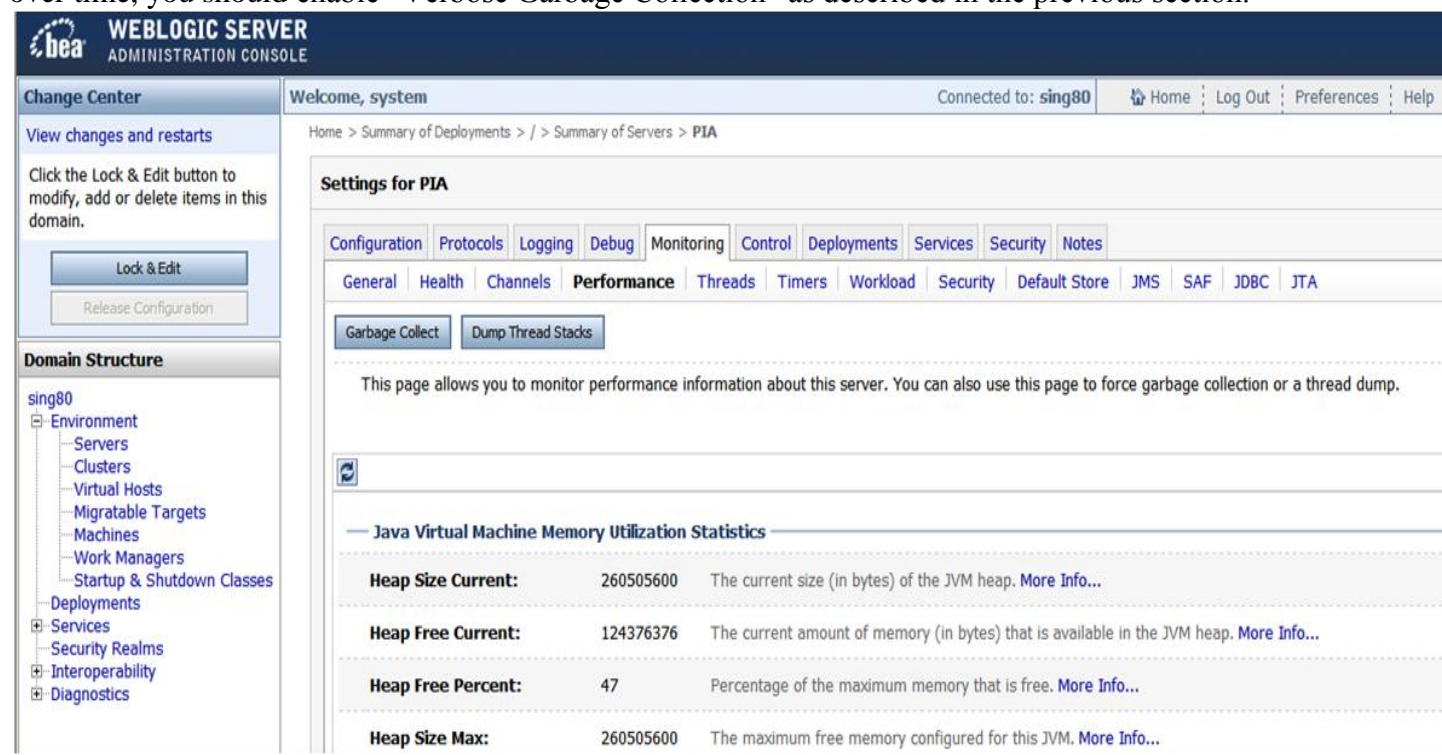
Monitoring WebLogic Memory Usage (cont'd)

Monitoring WebLogic Memory Usage from WebLogic Console

You can monitor the WebLogic memory from the WebLogic console as follows:

1. Log into WebLogic console and Go to Environment->Servers
2. Click 'PIA' hyperlink
3. Click Monitoring tab and 'Performance' sub tab
4. This will show you the current amount of heap being used and percent free

Below is an example of the console. In this example, we see that 124MG (of the 260 MG heap) is free. The WebLogic console will only give you a "snapshot" of the current heap. So if you want to track heap usage over time, you should enable "Verbose Garbage Collection" as described in the previous section.



The screenshot shows the WebLogic Server Administration Console interface. The top navigation bar includes the 'bea' logo and 'WEBLOGIC SERVER ADMINISTRATION CONSOLE'. The left sidebar shows the 'Domain Structure' with a tree view containing 'sing80', 'Environment', 'Servers', 'Clusters', 'Virtual Hosts', 'Migratable Targets', 'Machines', 'Work Managers', 'Startup & Shutdown Classes', 'Deployments', 'Services', 'Security Realms', 'Interoperability', and 'Diagnostics'. The main content area displays the 'Settings for PIA' page, with the 'Monitoring' tab selected. Under the 'Monitoring' tab, the 'Performance' sub-tab is active. The 'Performance' page shows 'Java Virtual Machine Memory Utilization Statistics' with the following data:

Java Virtual Machine Memory Utilization Statistics		
Heap Size Current:	260505600	The current size (in bytes) of the JVM heap. More Info...
Heap Free Current:	124376376	The current amount of memory (in bytes) that is available in the JVM heap. More Info...
Heap Free Percent:	47	Percentage of the maximum memory that is free. More Info...
Heap Size Max:	260505600	The maximum free memory configured for this JVM. More Info...

Monitoring WebLogic Memory Usage (cont'd)

Java Tools and 3rd Party Tools

In addition to the tools mentioned above, there are also many third party tools that will monitor memory usage.

Also, if you are using JRockit, you can use the “jrcmd” command to collect memory details for both java heap and native memory. Refer to the following knowledge document for more details:

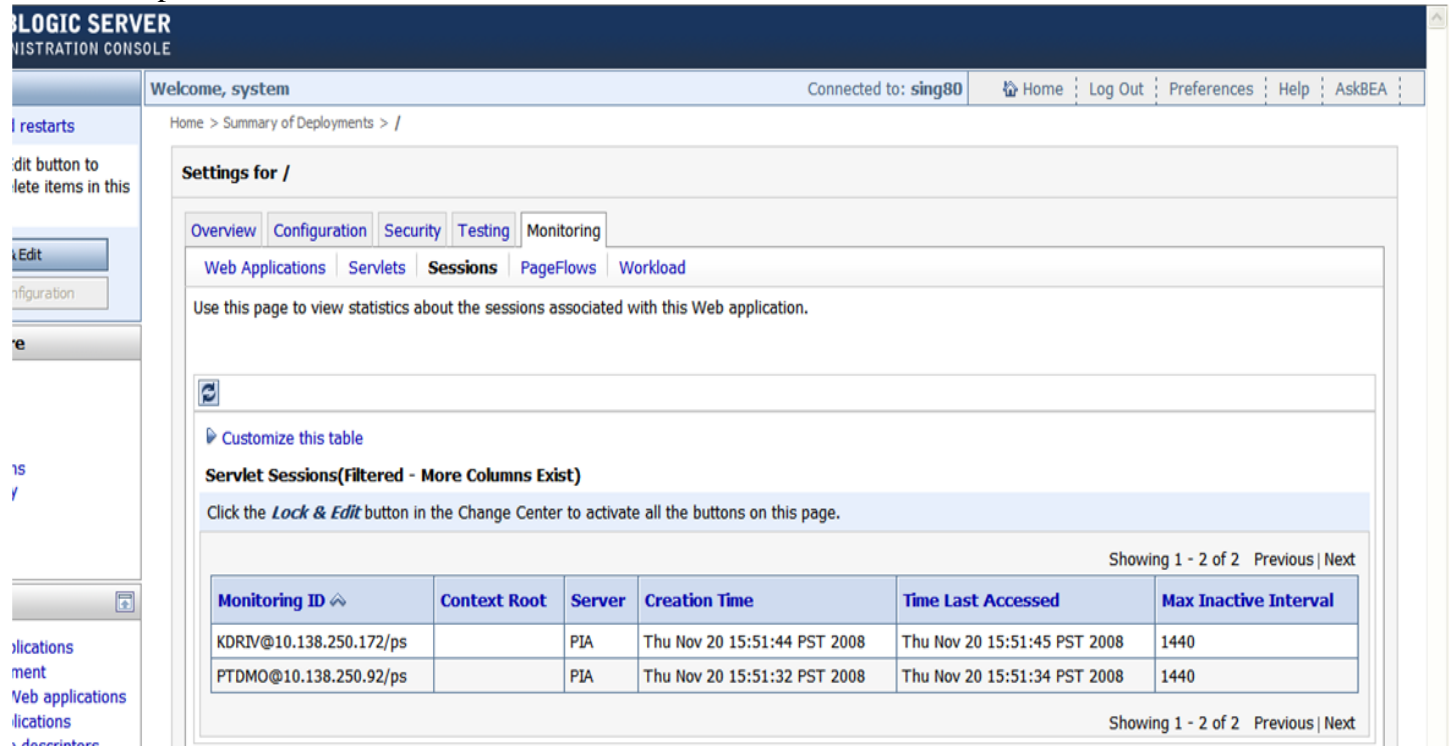
[Doc ID 1118810.1: Why is Process "beasvc.exe" Larger Than Max Heap Size \(-Xmx\)?](#)

Monitoring WebLogic Sessions

In order to assess how busy the web servers are, you can check the number of active sessions on each WebLogic PIA via the WebLogic Console as follows:

1. Log into the WebLogic Console and click 'Deployments' on left pane of the console.
2. Click the 'Control' tab
3. Click the '+' sign next to deployment 'Peoplesoft' in order to expand it
4. Click the '/' in the first entry under 'Modules' (no name is specified in this entry)
5. Click the 'Monitoring' tab, then click the 'Sessions' tab
6. The first time you go here, click 'customize this table' and add 'Monitoring id' to the 'chosen columns'

In the example below, there are two sessions for users KDRIV and PTDMO



The screenshot shows the WebLogic Server Administration Console interface. The top navigation bar includes 'Welcome, system', 'Connected to: sing80', and links for 'Home', 'Log Out', 'Preferences', 'Help', and 'AskBEA'. The left sidebar contains navigation links for 'restarts', 'edit button to delete items in this', 'Edit', 'configuration', 'e', '15', 'f', 'lications', 'ment', 'Web applications', 'lications', and 'discriminator'. The main content area is titled 'Settings for /' and includes tabs for 'Overview', 'Configuration', 'Security', 'Testing', and 'Monitoring'. Under the 'Monitoring' tab, there are sub-tabs for 'Web Applications', 'Servlets', 'Sessions', 'PageFlows', and 'Workload'. The 'Sessions' sub-tab is selected, displaying a message: 'Use this page to view statistics about the sessions associated with this Web application.' Below this, there is a 'Customize this table' link and a section titled 'Servlet Sessions (Filtered - More Columns Exist)'. A message states: 'Click the **Lock & Edit** button in the Change Center to activate all the buttons on this page.' A table displays session data with columns: 'Monitoring ID', 'Context Root', 'Server', 'Creation Time', 'Time Last Accessed', and 'Max Inactive Interval'. The table shows two sessions: one for 'KDRIV@10.138.250.172/ps' and another for 'PTDMO@10.138.250.92/ps', both on the 'PIA' server. The table is paginated, showing 'Showing 1 - 2 of 2' with 'Previous' and 'Next' links.

Monitoring ID	Context Root	Server	Creation Time	Time Last Accessed	Max Inactive Interval
KDRIV@10.138.250.172/ps		PIA	Thu Nov 20 15:51:44 PST 2008	Thu Nov 20 15:51:45 PST 2008	1440
PTDMO@10.138.250.92/ps		PIA	Thu Nov 20 15:51:32 PST 2008	Thu Nov 20 15:51:34 PST 2008	1440

Monitoring WebLogic Threads

You can monitor thread usage via WebLogic Console. By monitoring the threads, you can find out how many transactions are currently running on your web server (to get additional thread information, see section “Creating/Analyzing Thread Dumps”). To view threads via WebLogic Console, do the following:

1. Log into WebLogic console and navigate to Environment->Servers
2. Click ‘PIA’
3. Click Monitoring tab and ‘Threads’ sub tab

In the example below, there are two active threads. One thread is running component “KDCOMP.GBL” and the other thread is running the WebLogic console.

If you see a lot of active threads processing requests, then it is possible that there are issues on the app server and/or database which is causing the transactions to run slowly. To further troubleshoot, you may want to get a thread dump and also monitor the app server and database.

Self-Tuning Thread Pool (Filtered - More Columns Exist)

Active Execute Threads	Execute Thread Total Count	Execute Thread Idle Count	Queue Length	Pending User Request Count	Completed Request Count	H
2	5	0	0	0	4245	1

[Customize this table](#)

Self-Tuning Thread Pool Threads (Filtered - More Columns Exist)

Name	Current Request
[ACTIVE] ExecuteThread: '0' for queue: 'weblogic.kernel.Default (self-tuning)'	weblogic.servlet.internal.ServletRequestImpl@226895e[POST /psc/ps/EMPLOYEE/PT_LOCAL/c/WEB_PROFILE.KDCOMP.GBL HTTP/1.1 Accept: image/gif, image/x-bitmap,application/vnd.ms-xpsdocument,application/xhtml+xml,application/vnd.ms-excel,application/vnd.ms-powerpoint,application/msword,*/* Referer: http://kdriver3-la-us. PORTALPARAM_PTCNAV=KDCOMP_GBL&EOPP.SCNode=PT_LOCAL&EOPP.SCPortal=EMPLOYEE&EOPP.SCName=PT_WEB_PROFILE&EOPP.SCLabel=Web% 20Profile&EOPP.SCPTname=PT_WEB_PROFILE&FolderPath=PORTAL_ROOT_OBJECT.PT_PEOPLETOOLS.PT_WEB_PROFILE.KDCOMP_GBL&IsFolder=false&PortalActualU 2fWEB_PROFILE.KDCOMP.GBL&PortalContentURL=http%3a%2f%2fkdriver3-la-us.us.oracle.com%3a8000%2fpsc%2fpsc%2fEMPLOYEE%2fPT_LOCAL%2fc% 2fWEB_PROFILE.KDCOMP.GBL&PortalContentProvider=PT_LOCAL&PortalCRefLabel=KDCOMP_GBL&PortalRegistryName=EMPLOYEE&PortalServiceURI=http%3a%2f%2f 3a8000%2fpsc%2fpsc%2f8PortalHostNode=PT_LOCAL&NoCrumbs=yes&PortalKeyStruct=yes Accept-Language: en-us Content-Type: application/x-www-form-urlencoded CLR 2.0.50727;.NET CLR 1.1.4322;.NET CLR 3.0.4506.2152;.NET CLR 3.5.30729;.NET 4.0C;.NET 4.0E; MS-RTC LM 8) Content-Length: 349 Connection: Keep-Alive Ca 2Cbpgtcp%3Emp_ajc%2CamkMP%2F8g%5Dnaj%60neran%3Ckn%5D_ha%_jMP%2F8%2F26%2C65%2C7%2C22MP%2F8-04%43%5%00; ORA_UCM_SVC=3*OPN~1 3A~*EMP~1~0~/34/~null~*GMO~1~0~/34/~null; ORA_UCM_INFO=3~2B21968CD81A03A8E040018A24BF1000~Driver~Karen~karen.driver@oracle.com~USA~en~N 253Dhttp%25253A//my.oracle.com/index.htm%2526oid%253Dhttp%25253A//files.oraclecorp.com/app/HomePage%2526ot%253DA%2526oi%253D1223; ORA_ML3_SI PORTAL_PSSESSIONID=Mb4pNpYJ21BPL5d1GkXxpqqK9Tww1LQI-55365621; ExpirePage=http://kdriver3-la-us.us.oracle.com:8000/psc/ps/; PS_LOGINLIST=http://kdr PS_TOKEN=ogAAAAQDgEBAAAAYIAAAAAAAsAAAAABABTaGRyAk4AbQg4AC4AMQAwABRhg2J9QjFwVugAKKTEI1OnYNI1hHWIAAAAFANkYXRhVnclYg7DkBAAAVNeYX ADMINCONSOLESESSION=VDnVnpXmVR4shhcJbRYBVkm1yPJ06qspGZypZy77ZcppB7vV01I-55365621; HPTabName=DEFAULT]
[ACTIVE] ExecuteThread: '1' for queue: 'weblogic.kernel.Default (self-tuning)'	weblogic.servlet.internal.ServletRequestImpl@2268d13[GET /console/console.portal?_nfpb=true&_pageLabel=ServerMonitoringThreadsPage&handle=com.bea.console. image/x-bitmap, image/jpeg, image/pjpeg, application/x-shockwave-flash, application/x-ms-application, application/x-ms-xbap, application/vnd.ms-xpsdocument, applica http://kdriver3-la-us.us.oracle.com:8000/console/console.portal?_nfpb=true&_pageLabel=ServerMonitoringPerformancePage&handle=com.bea.console.handles.JMXHar Encoding: gzip, deflate User-Agent: Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 5.1; .NET CLR 2.0.50727;.NET CLR 1.1.4322;.NET CLR 3.0.4506.2152;.NET CLR 3. ORASSO_AUTH_HINT=v1.0~20110603233825; ORA_UCM_VER=%2FMP%2F8l_pd%2Cbpgtcp%3Emp_ajc%2CamkMP%2F8g%5Dnaj%60neran%3Ckn%5D_ha%_jMP 3ASE1%3ASE1%3ASE1%3ASE1%3ASE1%3A~*EMP~1~0~/34/~null~*GMO~1~0~/34/~null; ORA_UCM_INFO=3~2B21968CD81A03A8E040018A24BF1000~Driver~Ka s_cc=true; s_sq=oracleintranet%3D%2526pid%253Dhttp%25253A//my.oracle.com/index.htm%2526oid%253Dhttp%25253A//files.oraclecorp.com/app/HomePage%25 SignOnDefault=PTDMO; KDRIVER3-LA-US-8000-PORTAL_PSSESSIONID=Mb4pNpYJ21BPL5d1GkXxpqqK9Tww1LQI-55365621; ExpirePage=http://kdriver3-la-us.us.orad PS_TOKENEXPIRE=3_Jun_2011_17:34:28_GMT; PS_TOKEN=ogAAAAQDgEBAAAAYIAAAAAAAsAAAAABABTaGRyAk4AbQg4AC4AMQAwABRhg2J9QjFwVugAKKTEI1OnYNI1hHWIAAAAFANkYXRhVnclYg7DkBAAAVNeYX ADMINCONSOLESESSION=VDnVnpXmVR4shhcJbRYBVkm1yPJ06qspGZypZy77ZcppB7vV01I-55365621; HPTabName=DEFAULT]
[STANDBY] ExecuteThread: '2' for queue: 'weblogic.kernel.Default (self-tuning)'	
[STANDBY] ExecuteThread: '3' for queue: 'weblogic.kernel.Default (self-tuning)'	

Enabling HTTP Access Logs

When HTTP access logs are enabled, each http request, received by the Web Server, is recorded to a log file. For each http request, the log includes details about the exact request, time taken to process the request, http status code, and cookie information.

To enable the HTTP access logs, do the following on each web server:

1. Log into the WebLogic Admin Console and choose Environment -> Servers from left menu bar
2. If you have not already done so, in the Change Center of the Administration Console, click Lock & Edit
3. In the Servers table, click the name of the server instance whose HTTP logging you want to configure (PIA)
4. Select tab 'Logging' and sub tab 'HTTP'
5. Select the HTTP Access Log File Enabled check box, if it is not already selected.
6. Go to 'advanced' section on bottom of page
7. Choose format 'extended' and for 'Extended Logging Format Fields', enter:
date time c-ip cs-method time-taken sc-status bytes cs-uri cs(Cookie)
8. Save and click 'Activate Changes'
9. Restart the WebLogic PIA to pick up the changes

After the https access logs are enabled, all http requests will be logged to the PIA_access.log. Below is an excerpt from the access log:

date	time	c-ip	method	time-taken	sc-status	bytes	cs-uri	cs(Cookie)
11/21/2008	14:36:21	10.138.250.92	POST	0.047	200	3130	/psc/ps/EMPLOYEE/PT_LOCAL/c/WEB_PROFILE.WEB_PROFILE.GBL	PSJSESSI....
11/21/2008	14:36:21	10.138.250.92	GET	0	200	873	/cs/ps/cache/PT_PREVIOUSROW_D_1.gif	PSJSESSI

One key piece of information shown in the access logs is the HTTP status code for each HTTP request:

The following is a partial list of HTTP response status codes and standard associated phrases, intended to give a short textual description of the status. These status codes are specified by RFC 2616, along with additional, non-standard status codes sometimes used on the Web.

- 200 – OK: The request has succeeded.
- 304 – Not Modified: GET request - Document has not been modified
- 404 – Not Found: The server has not found anything matching the Request-URI.
- 500 – Internal Server Error: The server encountered an unexpected condition, which prevented it from fulfilling the request.

For a complete list of HTTP status codes you can refer to the following Website:

<http://www.w3.org/Protocols/rfc2616/rfc2616-sec10.html>

Refer to the following document for additional details on enabling HTTP access logs:

[Doc# 662319.1: Enabling and Configuring an HTTP Log for WebLogic](#)

Monitoring the Tuxedo Application Server

Web server issues can often be caused by performance issues on the Application server or Database. So it is important to monitor your app server.

Use “psadmin” command to monitor app server activity as follows:

1. cd <PS_HOME>/appserv
2. psadmin
3. Go to ‘domain status menu’
4. From here, you can view server status, queue status and client status.

Each PeopleSoft process has its own queue. The Queue Status menu will tell you how many requests are sitting in the queue (waiting for one of the servers to free up). It is normal to see some queuing during peak usage. But if you are seeing it often and seeing more than about 5 requests sitting in the queue, this is usually an indication that you need to start more of that particular server.

However, if you are seeing queuing on the app server and you also notice SVCTIMEOUTs in the tuxedo log, this could be pointing to a problem on the database.

In the example below, showing ‘Queue Status’, we see that there are two PSAPPSRV processes and one request is queued up and waiting.

```
-----
PeopleSoft Domain Status Menu
-----
      Domain Name: DOMAIN2
1) Server status
2) Client status
3) Queue status
q) Quit

Command to execute (1-3, q) [q]: 3

Prog Name      Queue Name  # Serve Wk Queued  # Queued  Ave. Len  Machine
-----
PSAPPSRV.exe   APPQ        2         -           1          -    mymachine
BBL.exe        51984       1         -           0          -    mymachine
PSMONITORSRV.e MONITOR     1         -           0          -    mymachine
JSL.exe        00095.00200 1         -           0          -    mymachine
PSWATCHSRV.exe WATCH       1         -           0          -    mymachine
```

Monitoring the Tuxedo Application Server (cont'd)

Below is an example of psadmin output for Client Status. In this example, there are three requests being processed (see status = BUSY/W)

LMID	User Name	Client Name	Time	Status	Bgn/Cmmt/Abrt
mymach	NT	JSH	0:39:17	IDLE	0/0/0
mymach	JPOOL_2	dhcp-psft-g-4t+	0:05:40	BUSY/W	0/0/0
mymach	JPOOL_1	kdriver-us	0:15:58	BUSY/W	0/0/0
mymach	JPOOL_3	kdriver-us.us.+	0:00:36	BUSY/W	0/0/0
mymach	NT	tadmin	0:00:00	IDLE	0/0/0

Creating/Analyzing Thread Dumps

Thread dumps can be very helpful when troubleshooting issues with WebLogic hangs or performance issues, as the thread dump will show you what each thread is doing at time of problem. This often provides clues as to cause of problem.

When you are experiencing WebLogic performance issues or a WebLogic hang, we recommend that you get a thread dump, prior to restarting the WebLogic PIA (ideally you should get 3 thread dumps at 2 minute intervals).

How to Create a Thread Dump:

For Unix platforms, you can create a thread dump by running this command:

kill -3 <PID> (where <PID> is the java process for the WebLogic PIA)

For Windows platforms, you can create a thread dump as follows:

- a. cd <WEBLOGIC_HOME>\server\bin
- b. beasvc.exe -dump -svcname:peoplesoft-PIA
(replace "peoplesoft-PIA" with your service name as it appears in Windows Control Panel/Services)
- c. You'll see message 'Sending threaddump request to service peoplesoft-PIA'

Refer to the following document if you are having issues getting the thread dump:

[Document#659452.1: How to Create a Thread Dump with WebLogic Server 9.2 and 10.3](#)

Locating the Thread Dump:

The thread dump is placed in the WebLogic log file. The log file location varies depending on the OS platform:

- a. For UNIX, the output is sent to:
<PS_HOME>/webserv/<DOMAIN_NAME>/servers/logs/PIA_stdout.log
- b. For Linux, the output is sent to:
<PS_HOME>/webserv/<DOMAIN_NAME>/servers/logs/PIA_stderr.log
- c. For Windows, the output is sent to:
<PS_HOME>\webserv\<DOMAIN_NAME>\servers\PIA\logs\NTservice-<DOMAIN_NAME>-PIA.log

Thread Dumps (cont'd)

Analyzing a Thread Dump:

The thread dump can be a bit challenging to analyze, and you may need assistance from an Oracle Support Engineer. Below are some tips on how to analyze the thread dump. This information is broken out into the following sections:

1. General Information about the thread dump
2. Overview of types of threads commonly seen in thread dump
3. Examples of different issues you may observe in the thread dump

General Information about the Thread Dump:

Note that the thread dump always begins with this line:

===== FULL THREAD DUMP =====

And ends with this line:

===== END OF THREAD DUMP =====

The first line of the thread dump shows when the thread dump was created, followed by the exact java version you are using. Example:

Mon Apr 18 12:46:56 2011

Oracle JRockit(R) R28.0.0-679-130297-1.6.0_17-20100312-2123-windows-ia32

The above example shows we are using 32 bit JRockit for 1.6.0_17 (32-bit java must be used for PeopleTools 8.50 and below. 64-bit java must be used for PeopleTools 8.51 and above).

Within the thread dump, you will see details for each thread which include the state of the thread along with its call stack. Below is an example of a thread in the thread dump. The call stack should be read like any other Java or C++ call stack, from bottom up. You will be able to tell from the call stack, whose code is being executed (eg WebLogic's, Java, PeopleSoft's or some other application's).

[ACTIVE] ExecuteThread: '0' for queue: 'weblogic.kernel.Default (self-tuning)'" id=16 idx=0x48 tid=3836 prio=5 alive, waiting, native_blocked, daemon

```
-- Waiting for notification on: weblogic/work/ExecuteThread@0x21614890[fat lock]
at jrockit/vm/Threads.waitForNotifySignal(JLjava/lang/Object;)Z(Native Method)
at java/lang/Object.wait(J)V(Native Method)
at java/lang/Object.wait(Object.java:485)
at weblogic/work/ExecuteThread.waitForRequest(ExecuteThread.java:157)
^-- Lock released while waiting: weblogic/work/ExecuteThread@0x21614890[fat lock]
at weblogic/work/ExecuteThread.run(ExecuteThread.java:178)
at jrockit/vm/RNI.c2java(IIII)V(Native Method)
-- end of trace
```

The reason that we sometimes recommend getting multiple thread dumps, is to determine if threads are “stuck” or if they are progressing. For example, if you run multiple thread dumps that show the same thread (eg. Execute Thread#60) executing different pieces of code on each dump, that is considered normal.

Thread Dumps (cont'd)

Overview of Types of Threads commonly seen in Thread Dump:

Threads waiting for Requests

You will always see some threads that are just waiting for work, as WebLogic always allocates some threads to be available and ready to process any incoming requests. These threads can easily be identified because you'll see "ExecuteThread.waitForRequest in the call stack (see example below). These threads will be in 'ACTIVE' or 'STANDBY' mode. These threads do not have much significance when troubleshooting. However, if you see a lot of these threads waiting for requests (20 or more), it most likely indicates that the environment is just recovering from a very heavy load, when the thread dump was taken (and as the load diminishes, WebLogic will remove many of these extra threads that are waiting for requests)

[ACTIVE] ExecuteThread: '0' for queue: 'weblogic.kernel.Default (self-tuning)' id=16 idx=0x48 tid=3836 prio=5 alive, waiting, native_blocked, daemon

```
-- Waiting for notification on: weblogic/work/ExecuteThread@0x21614890[fat lock]
at jrockit/vm/Threads.waitForNotifySignal(JLjava/lang/Object;)Z(Native Method)
at java/lang/Object.wait(J)V(Native Method)
at java/lang/Object.wait(Object.java:485)
at weblogic/work/ExecuteThread.waitForRequest(ExecuteThread.java:157)
^-- Lock released while waiting: weblogic/work/ExecuteThread@0x21614890[fat lock]
at weblogic/work/ExecuteThread.run(ExecuteThread.java:178)
at jrockit/vm/RNI.c2java(I)I(Native Method)
-- end of trace
```

Socket Muxer Threads

You will also see approximately two to five socket muxer threads. These threads' main responsibility is to read the request off the socket and pass the work to the appropriate thread. WebLogic allocates a percentage of execute threads from the self-tuning thread pool to be Muxer threads. Usually you will see three or four of these threads:

"ExecuteThread: '0' for queue: weblogic.socket.Muxer" id=25 idx=0x60 tid=2068 prio=5 alive, in native, daemon
at
weblogic/socket/NTSocketMuxer.getIoCompletionResult(Lweblogic/socket/NTSocketMuxer\$IoCompletionData;)Z(Native Method)
at weblogic/socket/NTSocketMuxer.processSockets(NTSocketMuxer.java:81)
at weblogic/socket/SocketReaderRequest.run(SocketReaderRequest.java:29)
at weblogic/socket/SocketReaderRequest.execute(SocketReaderRequest.java:42)
at weblogic/kernel/ExecuteThread.execute(ExecuteThread.java:145)
at weblogic/kernel/ExecuteThread.run(ExecuteThread.java:117)
at jrockit/vm/RNI.c2java(I)I(Native Method)
-- end of trace

ListenThreads

You will also see approximately six "listen threads", usually three for SSL and three for non-SSL. The purpose of these threads is to wait for connections to arrive. All browser requests enter the WebLogic server through these threads.

"DynamicListenThread[Default]" id=39 idx=0x90 tid=2812 prio=9 alive, in native, daemon
at java/net/PlainSocketImpl.socketAccept(Ljava/net/SocketImpl;)V(Native Method)
at java/net/PlainSocketImpl.accept(PlainSocketImpl.java:390)
^-- Holding lock: java/net/SocksSocketImpl@0x10297E98[biased lock]
at java/net/ServerSocketImpl.accept(ServerSocket.java:453)
at java/net/ServerSocket.accept(ServerSocket.java:421)
at weblogic/socket/WeblogicServerSocket.accept(WeblogicServerSocket.java:38)
at weblogic/server/channels/DynamicListenThread\$SocketAcceptor.accept(DynamicListenThread.java:535)
at weblogic/server/channels/DynamicListenThread\$SocketAcceptor.access\$200(DynamicListenThread.java:420)

```

at weblogic/server/channels/DynamicListenThread.run(DynamicListenThread.java:171)
at java/lang/Thread.run(Thread.java:619)
at jrockit/vm/RNI.c2java(IIIII)V(Native Method)
-- end of trace

```

```

"DynamicSSLListenThread[DefaultSecure]" id=40 idx=0x94 tid=3148 prio=9 alive, in native, daemon
at java/net/PlainSocketImpl.socketAccept(Ljava/net/SocketImpl;)V(Native Method)
at java/net/PlainSocketImpl.accept(PlainSocketImpl.java:390)
^-- Holding lock: java/net/SocksSocketImpl@0x13897FB0[biased lock]
at java/net/ServerSocketImpl.accept(ServerSocket.java:453)
at javax/net/ssl/impl/SSLServerSocketImpl.accept(Ljava/net/Socket;(Unknown Source)
at weblogic/server/channels/DynamicListenThread$SocketAcceptor.accept(DynamicListenThread.java:535)
at weblogic/server/channels/DynamicListenThread$SocketAcceptor.access$200(DynamicListenThread.java:420)
at weblogic/server/channels/DynamicListenThread.run(DynamicListenThread.java:171)
at java/lang/Thread.run(Thread.java:619)
at jrockit/vm/RNI.c2java(IIIII)V(Native Method)
-- end of trace

```

Jolt Connection Threads

WebLogic Server and the Tuxedo Application Server use Jolt to communicate with each other. PIA creates two threads inside the WebLogic's JVM per Jolt connection. For each Jolt connection made between WebLogic and the Tuxedo Application Servers, you will see a LLENwReader and a LLENwWriter thread in the thread dump:

```

"LLENwReader" id=52 idx=0xc4 tid=4408 prio=5 alive, in native, daemon
at jrockit/net/SocketNativeIO.readBytesPinned(Ljava/io/FileDescriptor;[BIII)I(Native Method)
at jrockit/net/SocketNativeIO.socketRead(SocketNativeIO.java:32)
at java/net/SocketInputStream.socketRead0(Ljava/io/FileDescriptor;[BIII)I(SocketInputStream.java)
at java/net/SocketInputStream.read(SocketInputStream.java:129)
at java/io/DataInputStream.readFully(DataInputStream.java:178)
at bea/jolt/NwReader.run(NwHdlr.java:3979)
at jrockit/vm/RNI.c2java(IIIII)V(Native Method)
-- end of trace

```

```

"LLENwWriter" id=53 idx=0xc8 tid=7828 prio=5 alive, waiting, native_blocked, daemon
-- Waiting for notification on: bea/jolt/OutQ @0x1DA95200[fat lock]
at jrockit/vm/Threads.waitForNotifySignal(JLjava/lang/Object;)Z(Native Method)
at java/lang/Object.wait(J)V(Native Method)
at java/lang/Object.wait(Object.java:485)
at bea/jolt/OutQ.getFromQ(OutQ.java:89)
^-- Lock released while waiting: bea/jolt/OutQ @0x1DA95200[fat lock]
at bea/jolt/NwWriter.run(NwHdlr.java:4341)
at jrockit/vm/RNI.c2java(IIIII)V(Native Method)
-- end of trace

```

Threads waiting on Application Server

If the web server is waiting on the app server to process a request, you will see the following thread (below)

```

"[ACTIVE] ExecuteThread: '2' for queue: 'weblogic.kernel.Default (self-tuning)'" id=35 idx=0x80 tid=2304 prio=5 alive,
waiting, native_blocked, daemon
-- Waiting for notification on: bea/jolt/IOBuf @0x11730730[fat lock]
at jrockit/vm/Threads.waitForNotifySignal(JLjava/lang/Object;)Z(Native Method)
at java/lang/Object.wait(J)V(Native Method)
at bea/jolt/IOBuf.waitOnBuf(IOBuf.java:119)
^-- Lock released while waiting: bea/jolt/IOBuf @0x11730730[fat lock]
at bea/jolt/NwHdlr.recv(NwHdlr.java:1564)
at bea/jolt/CMgr.recv(CMgr.java:185)
at bea/jolt/JoltSession.recv(JoltSession.java:547)
at bea/jolt/JoltRemoteService.call(JoltRemoteService.java:323)
at bea/jolt/JoltRemoteService.call(JoltRemoteService.java:267)
at psft/pt8/net/NetReqRepSvc.sendRequest(NetReqRepSvc.java:613)

```

```
at psft/pt8/net/NetService.requestService(NetService.java:153)
at psft/pt8/net/NetReqRepSvc.requestService(NetReqRepSvc.java:350)
^-- Holding lock: psft/pt8/net/NetSession@0x1170BF50[biased lock]
[20-30 more lines]
```


Examples of Different Issues you may Observe in Thread Dump

Below are examples of different issues and the thread stacks you may observe.

Many threads waiting on App Server

If you see a lot of threads such as the one below, then this means that many of the WebLogic threads are waiting on the application server to finish processing the request:

"[ACTIVE] ExecuteThread: '2' for queue: 'weblogic.kernel.Default (self-tuning)'" id=35 idx=0x80 tid=2304 prio=5 alive, waiting, native_blocked, daemon

```
-- Waiting for notification on: bea/jolt/IOBuf@0x11730730[fat lock]
at jrockit/vm/Threads.waitForNotifySignal(JLjava/lang/Object;)Z(Native Method)
at java/lang/Object.wait(J)V(Native Method)
at bea/jolt/IOBuf.waitOnBuf(IOBuf.java:119)
^-- Lock released while waiting: bea/jolt/IOBuf@0x11730730[fat lock]
at bea/jolt/NwHdlr.recv(NwHdlr.java:1564)
at bea/jolt/CMgr.recv(CMgr.java:185)
```

Typically, you may only see one or two threads waiting on the application server. But if you see a lot of these threads, then this is not a web server issue, but rather there is some sort of performance issue on the application server and/or the database that needs addressed: perhaps there are long running SQL's and or database locks. Or perhaps there are insufficient app server domains and/or psappsrv processes.

Many threads processing the same call stack

If you see many threads all processing the same call stack, then you may need to review contents of the call stack in order to troubleshoot the issue. For example, in one case, the web server hung and the thread dump showed hundreds of threads like the one below. This was caused by an issue with a proxy server configuration, causing all threads to get hung up at logout:

```
"[ACTIVE] ExecuteThread: '387' for queue: 'weblogic.kernel.Default (self-tuning)'" RUNNABLE native
java.net.SocketInputStream.socketRead0(Native Method)
java.net.SocketInputStream.read(SocketInputStream.java:129)
com.sun.net.ssl.internal.ssl.InputRecord.readFully(InputRecord.java:293)
com.sun.net.ssl.internal.ssl.InputRecord.read(InputRecord.java:331)
com.sun.net.ssl.internal.ssl.SSLSocketImpl.readRecord(SSLSocketImpl.java:798)
com.sun.net.ssl.internal.ssl.SSLSocketImpl.readDataRecord(SSLSocketImpl.java:755)
com.sun.net.ssl.internal.ssl.AppInputStream.read(AppInputStream.java:75)
java.io.BufferedInputStream.fill(BufferedInputStream.java:218)
java.io.BufferedInputStream.read1(BufferedInputStream.java:258)
java.io.BufferedInputStream.read(BufferedInputStream.java:317)
java.io.FilterInputStream.read(FilterInputStream.java:116)
java.io.PushbackInputStream.read(PushbackInputStream.java:169)
psft.pt8.pshttp.https.HttpClient.parseHTTPHeader(HttpClient.java:596)
psft.pt8.pshttp.https.HttpClient.parseHTTP(HttpClient.java:568)
psft.pt8.pshttp.https.HttpURLConnection.getInputStream(HttpURLConnection.java:564)
java.net.HttpURLConnection.getResponseCode(HttpURLConnection.java:379)
psft.pt8.portal.ContentGetter.getContent(Unknown Source)
psft.pt8.portal.ContentGetter.getContent(Unknown Source)
psft.pt8.portal.ContentGetter.getContent(Unknown Source)
psft.pt8.psp.logoutAccessedPIAs(Unknown Source)
psft.pt8.psp.cleanup(Unknown Source)
psft.pt8.psp.onExpire(Unknown Source)
[more lines]
```

All threads busy and waiting on one thread

By design, the PIA does not allow more than one request per HTTP session, to be submitted to the application server. If the PIA receives multiple requests from the same HTTP session, it will queue up all subsequent requests and process just one at a time. Typically, there should not be situations where the PIA receives multiples requests from the same HTTP session. However, this can occur in the following situations:

1. You are using a proxy server that is re-submitting requests to the web server if a response is not received within a certain time.
-OR-
2. A user submits a long-running request, and while waiting for the request to finish, the user continuously attempts to submit more requests.

When one of the above scenarios occurs, in the thread-dump you see one request waiting on Jolt to get response from the App-Server and many other threads waiting for the lock on the session to be released. Below are excerpts from a thread dump, showing this situation:

1) There are many threads like this that are “blocked”, and all the threads are waiting on the same lock #.

```
"[ACTIVE] ExecuteThread: '5' for queue: 'weblogic.kernel.Default (self-tuning)'" id=50 idx=0xbc tid=4488 prio=5 alive, blocked, native_blocked, daemon
```

```
-- Blocked trying to get lock: java/lang/String@0x27D36AC0[thin lock]
  at jrockit/vm/Threads.sleep(I)V(Native Method)
  at jrockit/vm/Locks.waitForThinRelease(Locks.java:946)
[10-20 more lines]
```

2) The thread that is holding the lock on “0x27D36AC0” (that all blocked threads are waiting on), is usually processing a jolt request (ie it is waiting on the application server):

```
"[ACTIVE] ExecuteThread: '0' for queue: 'weblogic.kernel.Default (self-tuning)'" id=16 idx=0x48 tid=180 prio=5 alive, waiting, native_blocked, daemon
```

```
-- Waiting for notification on: bea/jolt/IOBuf@0x27E2C300[fat lock]
  at jrockit/vm/Threads.waitForNotifySignal(JLjava/lang/Object;)Z(Native Method)
  at java/lang/Object.wait(J)V(Native Method)
  at bea/jolt/IOBuf.waitOnBuf(IOBuf.java:119)
[10-15 lines]
  at psft/pt8/psc.onAction(psc.java:1264)
  at psft/pt8/psc.service(psc.java:681)
^-- Holding lock: java/lang/String@0x27D36AC0[thin lock]
[10-15 lines]
```

3) At the end of the thread dump, you may see a list of “blocked locked chains”. In this list, you’ll notice that all threads are waiting on one thread: “Thread #0” in this example. Which happens to be a jolt request (ie it is waiting on application server)

Blocked lock chains

=====

Chain 2:

```
"[ACTIVE] ExecuteThread: '2' for queue: 'weblogic.kernel.Default (self-tuning)'" id=35 idx=0x80 tid=3964 waiting for java/lang/String@0x27D36AC0 held by:
```

```
"[ACTIVE] ExecuteThread: '0' for queue: 'weblogic.kernel.Default (self-tuning)'" id=16 idx=0x48 tid=180 in chain 1
```

Chain 3:

```
"[ACTIVE] ExecuteThread: '3' for queue: 'weblogic.kernel.Default (self-tuning)'" id=44 idx=0xa4 tid=4620 waiting for java/lang/String@0x27D36AC0 held by:
```

```
"[ACTIVE] ExecuteThread: '0' for queue: 'weblogic.kernel.Default (self-tuning)'" id=16 idx=0x48 tid=180 in chain 1
```

Chain 4:

```
"[ACTIVE] ExecuteThread: '4' for queue: 'weblogic.kernel.Default (self-tuning)'" id=49 idx=0xb8 tid=1120 waiting for java/lang/String@0x27D36AC0 held by:
```

"[ACTIVE] ExecuteThread: '0' for queue: 'weblogic.kernel.Default (self-tuning)'" id=16 idx=0x48 tid=180 in chain 1

When the above occurs, the web server may become bogged down by the hundreds of threads that are waiting to be processed. So you will need to determine what is causing issue (proxy with incorrect timings? Impatient user?). Note that the PIA_weblogic.log will give you clues as to the user causing the issue, because you will most likely see many “stuck threads”, and the log provides the cookie information (including user id) for each of the stuck threads.