How to troubleshoot hangs

Unit objectives

After completing this unit, you should be able to:

- Describe what a hang is
- Detect a hang condition
- Trigger and analyze javacore files for hung threads
- Use the WebSphere Application Server hang detection facility
- Use the IBM Thread and Monitor Dump Analyzer for Java

Topics

- Application server hangs
- Hung thread detection
- IBM Thread and Monitor Dump Analyzer

WebSphere Education

Application server hangs

Application server hangs defined

Clarify the nature and extent of the hang

- A “hang” is not the same as a “crash”
- Is the entire process truly hung, or does it still respond to some requests?
  - Test with sample “Snoop” servlet, wsadmin commands, and other commands
- Deadlocks:
  - Often, one process fails to respond to a request because it called another process that is itself hung; sometimes it is hard to find the true culprit
  - Deadlocks also can occur within the same Java process, where one thread is deadlocked on another

Thread hangs

- WebSphere Application Server threads can become hung due to running poorly coded Java EE applications
- WebSphere administrators might not know that the application is slow or hung until a customer calls and complains
- WebSphere administrators should be alerted before significant problems arise
- Detecting if a thread is hung or just taking a long time to respond can be a difficult problem to solve correctly

Thread hang examples in application code

- Understand root causes
  - Customer code enters an endless loop such as:

```
while (true) {
  i++;
}
```

  - Customer code waits infinitely such as:

```
run() {
  object1.wait();
}
```

  - Customer code creates a deadlock such as:

```
synchronized (object1) {
  synchronized (object2) {
    // Work with objects
  }
}
```

```
synchronized (object2) {
  synchronized (object1) {
    // Work with objects
  }
}
```

WebSphere process hangs detection steps

1. Collect OS statistics, processor usage by other processes, and virtual memory paging
   - The JVM might be hung because OS resources are exhausted
2. When hung threads are suspected, manually trigger a thread dump
   - Use wsadmin or OS facilities; see next slide
   - Create a script or use script in IBM Support Assistant to collect MustGather data when the process that is suspected hangs
   - Distinguish the 100% CPU cases from idle CPU cases
3. For a typical hang, collect three javacore dumps a few minutes apart
   - To see whether anything is moving within the process (but slowly)
4. Examine the thread dumps manually or with tools
   - Look for deadlocks
   - Look for large number of threads that are blocked
   - Look for threads that are waiting after sending a request to some other process, now awaiting a response

How to manually trigger a thread dump

- Use operating system facilities:
  - `kill -3 <WAS PID>` (UNIX or Linux)

- Use the Java core button in the administrative console:
  - Click **Troubleshooting > Java dumps and cores > *server_name* > Java core**

- Explicitly tell WebSphere to generate a thread dump
  - From a command line, start the wsadmin shell
  - Specify `–lang jython` on the command line
  - Run the following Jython commands:

```
jvm = AdminControl.completeObjectName(
             'type=JVM,process=<server_name>,*')
AdminControl.invoke(jvm, 'dumpThreads')
```

Thread dump analysis

What are the significant pictures in the thread dump?

- Many threads are waiting in the same method for some resource
  - Probably a synchronization issue
  - Might be an outage or slow response from remote server

- No activity
  - WebSphere Application Server is not receiving traffic for some reason
  - Check front-end resources, networks, test clients, and so forth
  - Also, check timing of the thread dump

- Hundreds of threads
  - Shared resource not available
  - Customer must control web container threads better
  - Might need more capacity

Javacore hang indicators

- Look for the string “Deadlock detected”
- JVM monitor information:
  - Shows synchronization locks
  - Indicates blocked threads
- Active threads
  - Look for `state:R`, which indicates runnable threads
  - This example shows that the thread is doing I/O
  - If this thread is doing the same operation across multiple javacore files, there might be a network interface issue

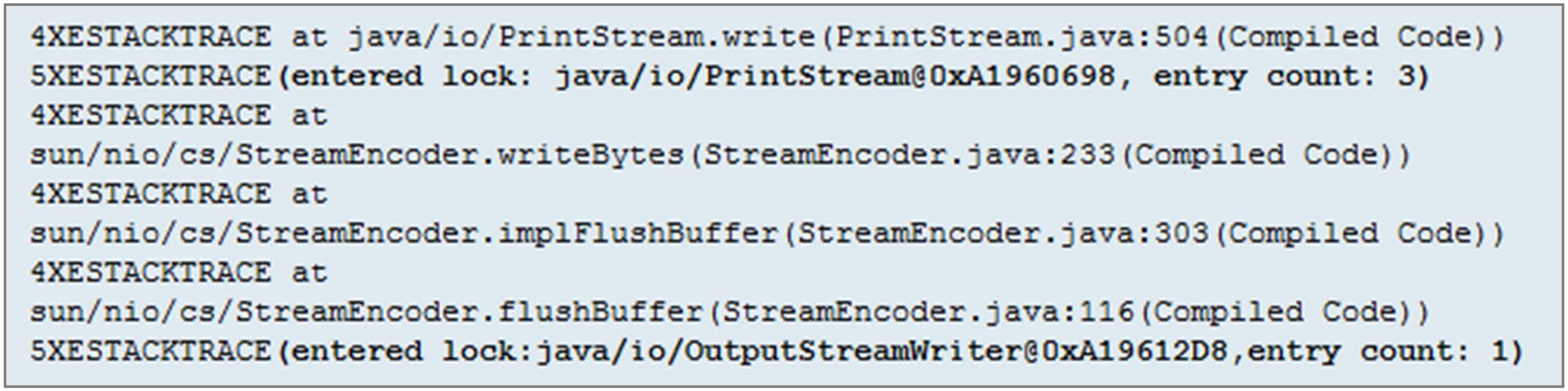
```
"Servlet.Engine.Transports:239" (TID:0x34B94018,sys_thread_t:0x7CD4E008,
state:R, native ID:0x10506) prio=5 at
java.net.SocketInputStream.socketRead(Native Method)
at java.net.SocketInputStream.read(SocketInputStream.java(Compiled Code))
at
com.ibm.ws.io.Stream.read(Stream.java(Compiled Code))
at
com.ibm.ws.io.ReadStream.readBuffer(ReadStream.java(Compiled Code))
```

Thread state values

- The values of state can be:
- R – runnable: The thread is able to run when given the chance
- CW – condition wait: The thread is waiting
- For example, the thread might be waiting because:
  - A `sleep()` call is made
  - The thread is blocked for I/O
  - A synchronized method of an object that is locked by another thread was called
  - The thread is synchronizing with another thread with a `join()` call
- S – suspended: The thread suspends another thread
- Z – zombie: The thread was killed
- P – parked: The new concurrency API (`java.util.concurrent`) parks thread
- B – blocked: The thread is waiting to obtain a lock that something else currently owns

Java 7: More details about monitor usage

- The Java stack trace might include extra lines that begin with `5XESTACKTRACE`
  - Shows the locks that were taken in the method on the previous line in the trace
  - Shows a cumulative total of how many times the locks were taken within that stack at that point
  - This information is useful for determining the locks that are held by a thread, and when those locks are released
- Example Java stack trace for a thread that calls `java.io.PrintStream` methods:



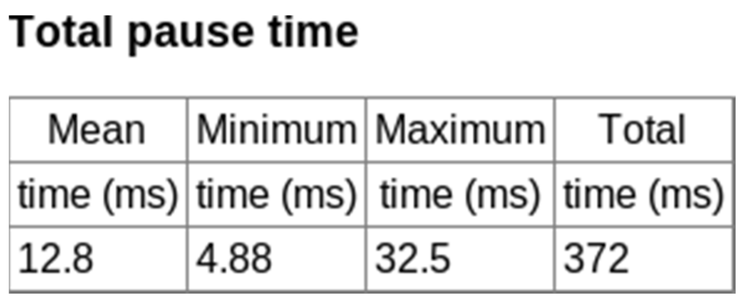
```
4XESTACKTRACE at java/io/PrintStream.write(PrintStream.java:504(Compiled Code))
5XESTACKTRACE(entered lock: java/io/PrintStream@0xA1960698, entry count: 3)
4XESTACKTRACE at
sun/nio/cs/StreamEncoder.writeBytes(StreamEncoder.java:233(Compiled Code))
4XESTACKTRACE at
sun/nio/cs/StreamEncoder.implFlushBuffer(StreamEncoder.java:303(Compiled Code))
4XESTACKTRACE at
sun/nio/cs/StreamEncoder.flushBuffer(StreamEncoder.java:116(Compiled Code))
5XESTACKTRACE(entered lock:java/io/OutputStreamWriter@0xA19612D8,entry count: 1)
```

Javacore hang symptoms (1 of 2)

- Check to see whether threads are blocked waiting on monitors
  – Might indicate bottleneck on unavailable resources or poor synchronization logic
  – Deadlocks within the process are noted in the javacore
- If threads are in a running state
  – Check method across multiple javacore files
  – If individual threads in the same method, might indicate looping logic
- If threads are in wait states, might indicate that a resource (local or remote) is causing the hang
- Technical note:
  – Some threads that are Runnable, are shown as in a Conditional Wait
  – Since IBM JVM version 5, when a javacore is taken, some threads in a Runnable (R) state enter Conditional Wait (CW) state during the javacore
  – This behavior is by design and it is meant to maximize internal consistency during the process of creating the javacore
  – This behavior increases the quality of the javacore and lessens the potential for a crash or data corruption

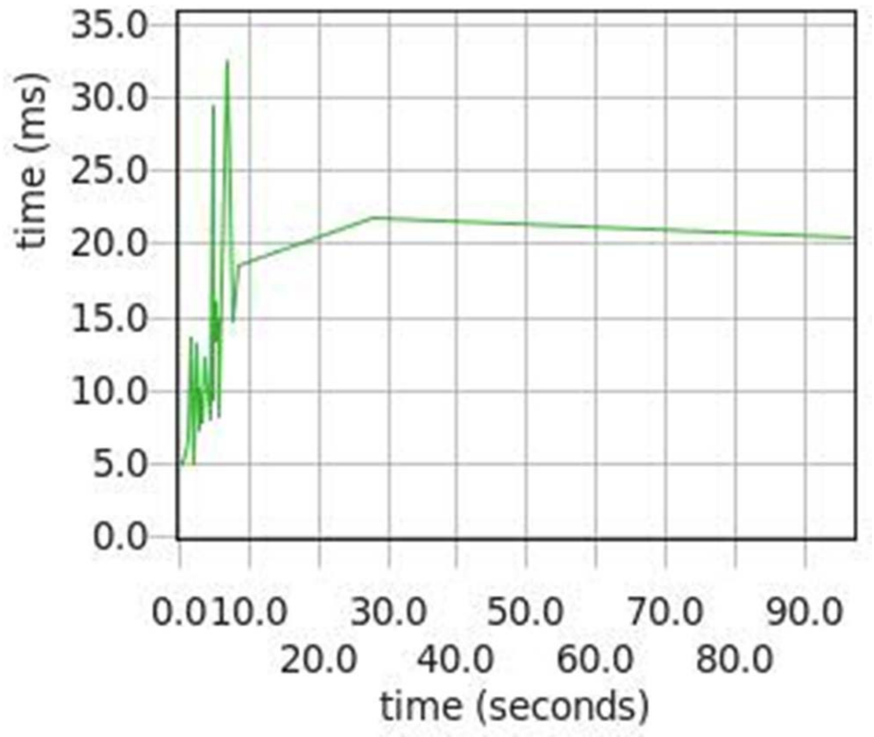
Javacore hang symptoms (2 of 2)

- The JVM might be hung because it is spending too much time in garbage collection
- An IBM javacore file can be loaded directly into the Garbage Collection and Memory Visualizer tool to review the garbage collection usage for the last few cycles before the javacore was triggered
  - Enable the Total Pause Time statistic in the Memory template



Total pause time

Mean	Minimum	Maximum	Total
time (ms)	time (ms)	time (ms)	time (ms)
12.8	4.88	32.5	372



Hang detection tools

- **ThreadMonitor**: ThreadMonitor architecture was created to monitor thread pools within WebSphere
- Notification of potentially hung threads is logged
- Monitored pools include:
  - Web container thread pool
  - ORB thread pool
  - Others
- IBM Thread and Monitor Dump Analyzer
  - GUI-based tool
  - Gathers and analyzes thread dumps from a WebSphere Application Server
  - Provides tuning recommendations that are based on analysis
- IBM Monitoring and Diagnostic Tools for Java – Garbage Collection and Memory Visualizer
  - A verbose GC data visualizer
  - Parses and plots various log types that include verbose GC logs, `-Xtgc` output, native memory logs (output from `ps`, `svmon`, and `perfmon`)

Hung thread detection

WebSphere hung thread detection

- WebSphere contains a built-in hung thread detection function
- ThreadMonitor architecture was created to monitor thread pools within WebSphere
  - The ThreadMonitor monitors web container, ORB, and asynchronous bean thread pools
  - Turned on by default
- Unmanaged threads are not monitored
  - Threads that applications create (not allowed in Java EE)
  - Some internal threads
- Upon notification of a hung thread:
  - Obtain a javacore and see what the thread is doing
  - You can configure WebSphere to generate a javacore automatically when a hung thread is detected
  - Investigate the nature of the thread

Hung thread detection internals

- When the thread pool gives work to a thread, it notifies the thread monitor
  - Thread monitor notes thread ID and time stamp

- Thread monitor compares active threads to time stamps
  - Threads active longer than the time limit are marked “potentially hung”

- Performance impact is minimal (< 1%)

Hung thread detection notification

- No action that is taken to kill the thread – only a notification mechanism
  - Thread dump can be taken when hung thread is detected
- If thread is suspected to be hung, notification is sent to:
  - JMX listeners
  - PMI clients for ThreadPool metric (counters are updated)
  - Message that is written to `SystemOut.log`:

```
[7/10/13 11:51:30:243 EST] 00000020 ThreadMonitor W WSVR0605W: Thread
"WebContainer : 2" (00000028) has been active for 64828 milliseconds
and may be hung. There is/are 1 thread(s) in total in the server that
may be hung.
 at java.lang.Thread.sleep(Native Method)
 at java.lang.Thread.sleep(Thread.java:850)
 at
com.ibm.websphere.samples.plantsbywebspherewar.ShoppingServlet.perfor
mTask(ShoppingServlet.java:141)
. . . [remaining stack traces have been truncated]
```

Hung thread detection false alarms

- What about false alarms?
  - For example, a thread that takes several minutes to complete a long-running query
- If a thread previously reported to be hung completes its work, a notification is sent:

```
[7/10/13 11:51:47:210 EST] 00000020 ThreadMonitor W WSVR0605W:
Thread "WebContainer : 2" (00000028) was previously reported to be
hung but has completed.  It was active for approximately 65900
milliseconds.  There are 0 threads in total in the server that
still may be hung.
```

- The monitor has a self-adjusting system to make a best effort to deal with false alarms

Hung thread detection configuration

- Custom properties for hung thread detection configuration
  - Select **Servers > Application Servers > server_name**
  - Under Server Infrastructure, click **Administration > Custom Properties**
  - Add the following properties (or change if present):

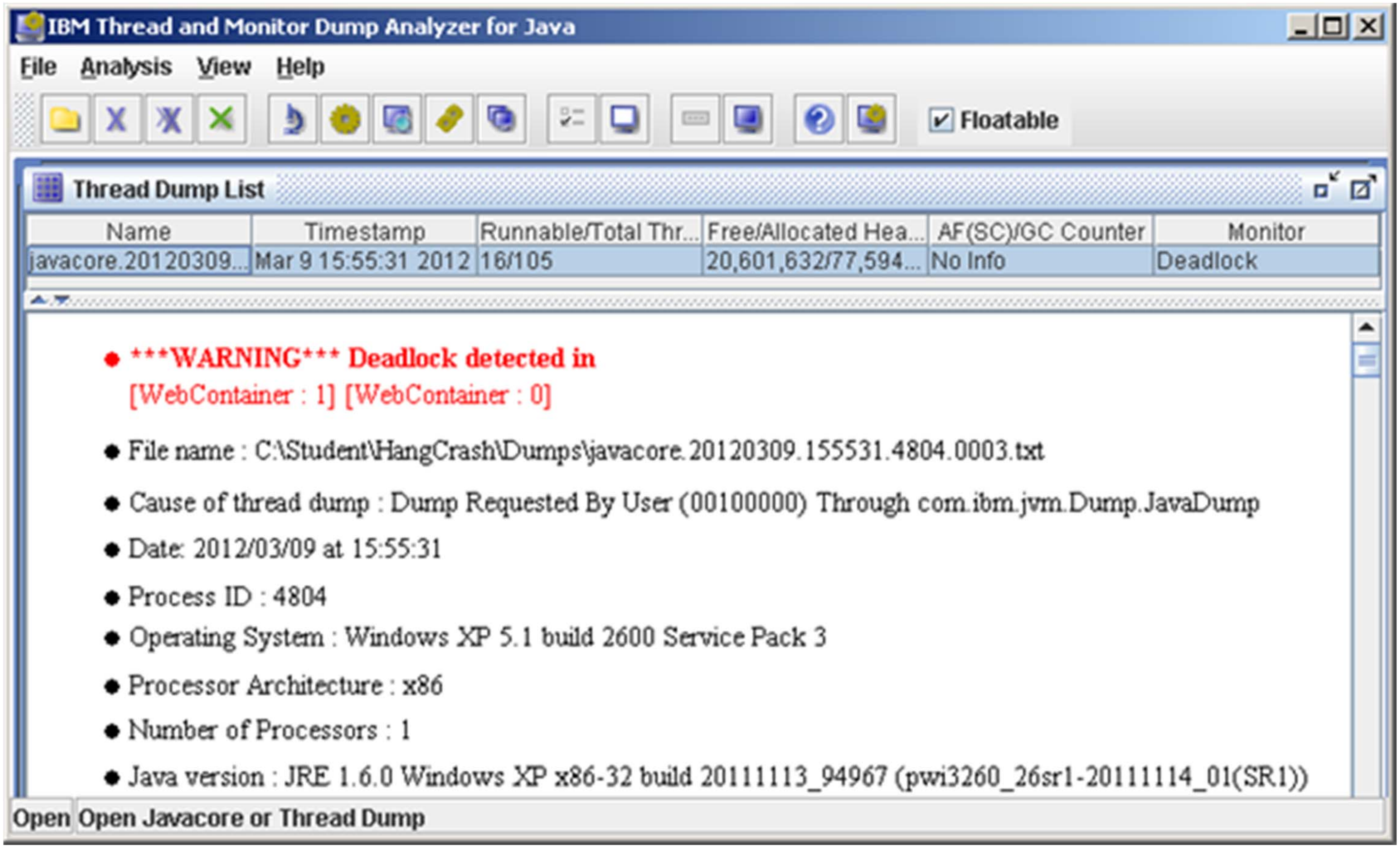
Property	Units	Default	Description
`com.ibm.websphere.threadmonitor.interval`	seconds	180	The interval at which the thread pools are polled for hung threads
`com.ibm.websphere.threadmonitor.threshold`	seconds	600	The length of time that a thread can be active before being marked as "potentially hung"
`com.ibm.websphere.threadmonitor.false.alarm.threshold`	N/A	100	The number of false alarms that can occur before automatically increasing the threshold by 50%
`com.ibm.websphere.threadmonitor.dump.java`	N/A	False	Set to true to cause a javacore to be created when a hung thread is detected and a WSVR0605W message is printed

IBM Thread and Monitor Dump Analyzer

What is Thread and Monitor Dump Analyzer (TMDA)?

- TMDA is a tool to analyze thread dumps
  - Available from the IBM Support Assistant
- Used for:
  - Analyzing threads and monitors in javacore files
  - Comparing multiple javacore files from the same process
- Friendlier interface for novice thread dump readers
  - Provides graphical interface to view contents of the thread dump
- Used to analyze threads for:
  - Performance bottlenecks
  - Determining whether deadlocks exist
  - Determining whether threads are being blocked on monitors (might not be a deadlock)

TMDA: Open a thread dump

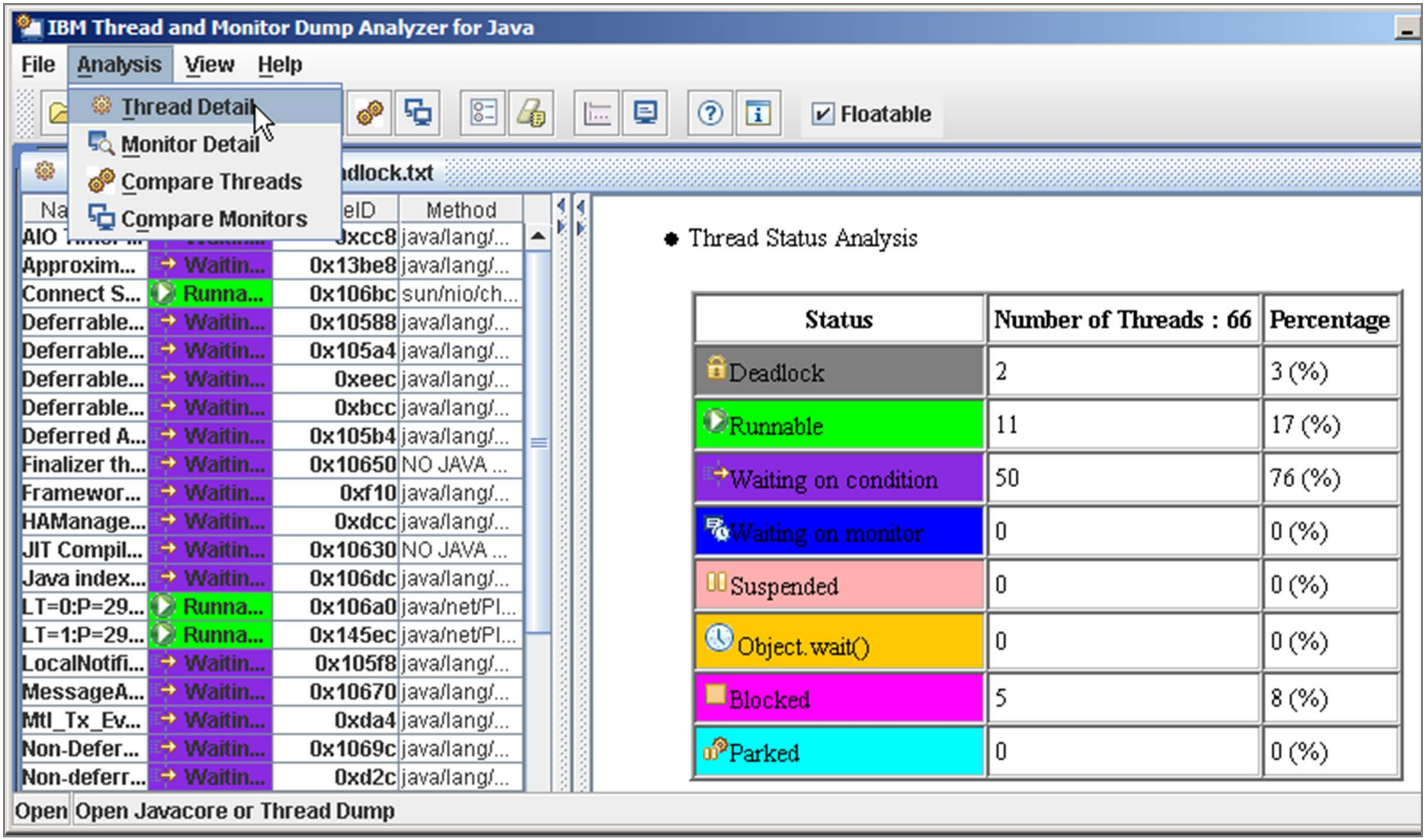


Name	Timestamp	Runnable/Total Thr...	Free/Allocated Hea...	AF(SC)/GC Counter	Monitor
javacore.20120309...	Mar 9 15:55:31 2012	16/105	20,601,632/77,594...	No Info	Deadlock

- ***WARNING*** Deadlock detected in [WebContainer : 1] [WebContainer : 0]
- File name : C:\Student\HangCrash\Dumps\javacore.20120309.155531.4804.0003.txt
- Cause of thread dump : Dump Requested By User (00100000) Through com.ibm.jvm.Dump.JavaDump
- Date: 2012/03/09 at 15:55:31
- Process ID : 4804
- Operating System : Windows XP 5.1 build 2600 Service Pack 3
- Processor Architecture : x86
- Number of Processors : 1
- Java version : JRE 1.6.0 Windows XP x86-32 build 20111113_94967 (pwi3260_26sr1-20111114_01(SR1))

Open Open Javacore or Thread Dump

Thread detail: Thread status analysis

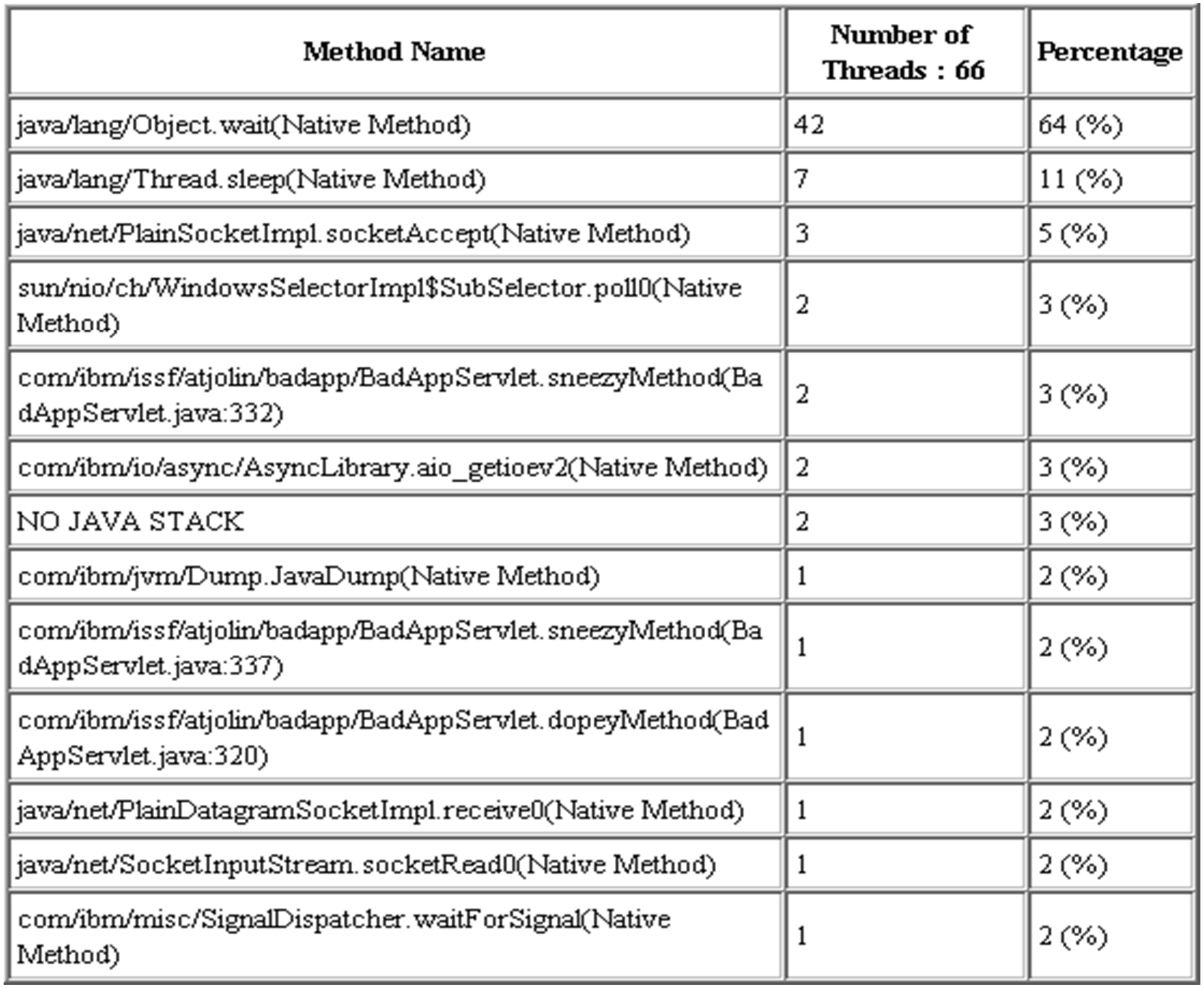


Na		eID	Method
AIO T...		0xcc8	java/lang/...
Approxim...	Waitin...	0x13be8	java/lang/...
Connect S...	Runna...	0x106bc	sun/nio/ch...
Deferrable...	Waitin...	0x10588	java/lang/...
Deferrable...	Waitin...	0x105a4	java/lang/...
Deferrable...	Waitin...	0xeec	java/lang/...
Deferrable...	Waitin...	0xbcc	java/lang/...
Deferred A...	Waitin...	0x105b4	java/lang/...
Finalizer th...	Waitin...	0x10650	NO JAVA ...
Framewor...	Waitin...	0xf10	java/lang/...
HAManage...	Waitin...	0xdcc	java/lang/...
JIT Compil...	Waitin...	0x10630	NO JAVA ...
Java index...	Waitin...	0x106dc	java/lang/...
LT=0:P=29...	Runna...	0x106a0	java/net/Pl...
LT=1:P=29...	Runna...	0x145ec	java/net/Pl...
LocalNotifi...	Waitin...	0x105f8	java/lang/...
MessageA...	Waitin...	0x10670	java/lang/...
Mtl_Tx_Ev...	Waitin...	0xda4	java/lang/...
Non-Defer...	Waitin...	0x1069c	java/lang/...
Non-deferr...	Waitin...	0xd2c	java/lang/...

- Thread Status Analysis

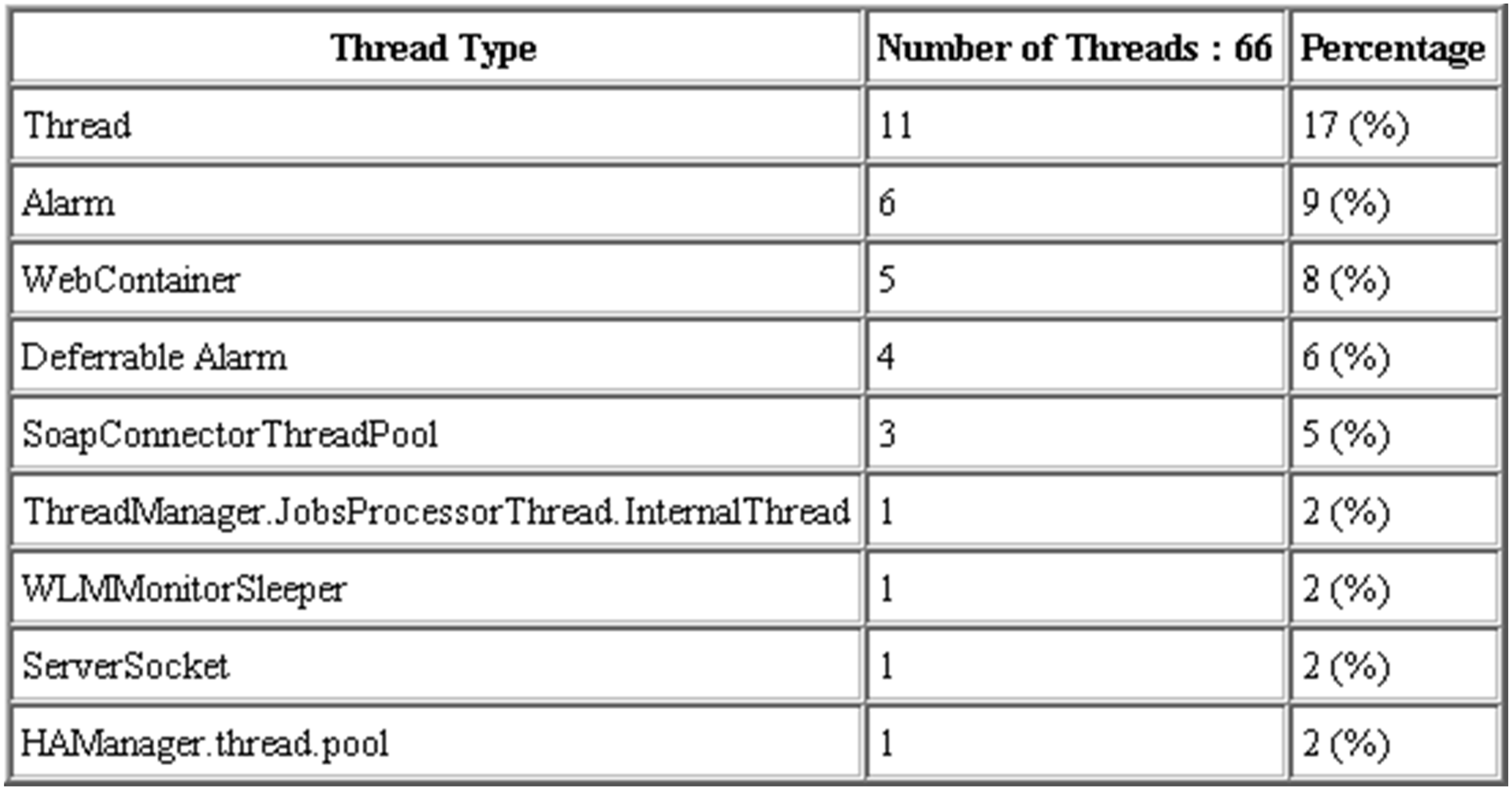
Status	Number of Threads : 66	Percentage
Deadlock	2	3 (%)
Runnable	11	17 (%)
Waiting on condition	50	76 (%)
Waiting on monitor	0	0 (%)
Suspended	0	0 (%)
Object.wait()	0	0 (%)
Blocked	5	8 (%)
Parked	0	0 (%)

Thread detail: Thread method analysis



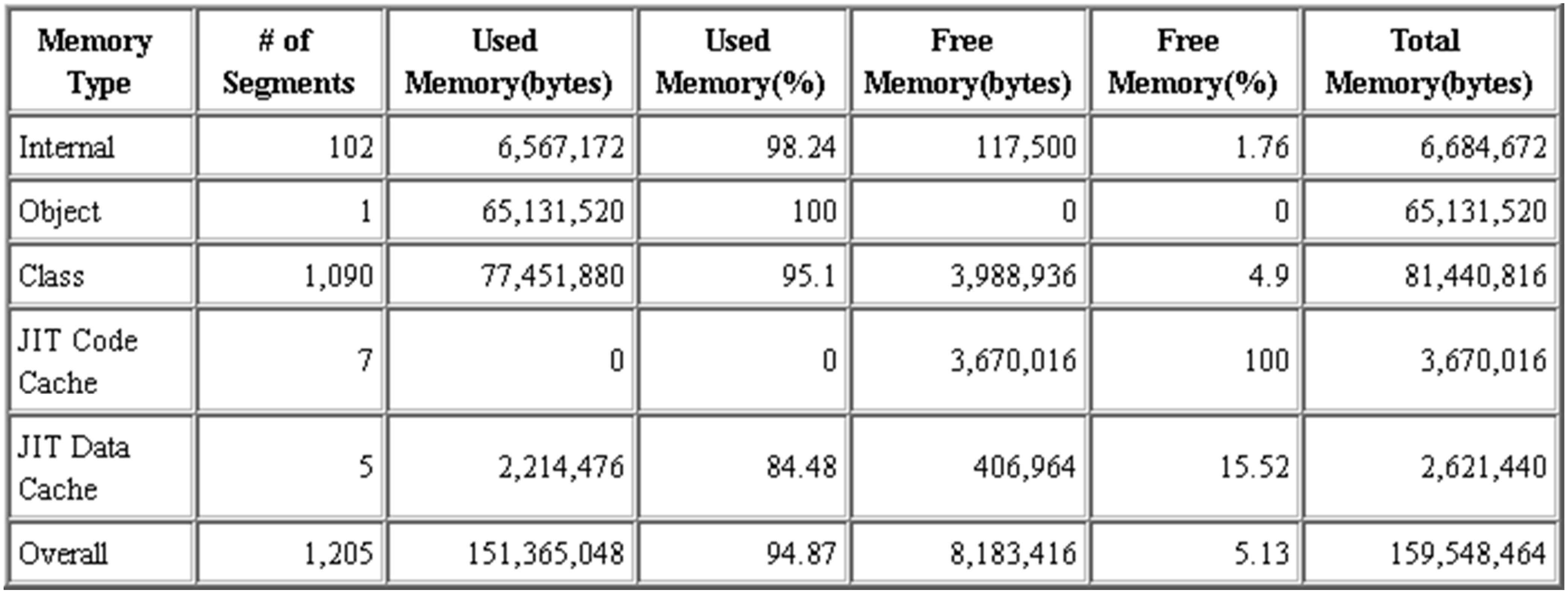
Method Name	Number of Threads : 66	Percentage
java/lang/Object.wait(Native Method)	42	64 (%)
java/lang/Thread.sleep(Native Method)	7	11 (%)
java/net/PlainSocketImpl.socketAccept(Native Method)	3	5 (%)
sun/nio/ch/WindowsSelectorImpl$SubSelector.poll0(Native Method)	2	3 (%)
com/ibm/issf/atjolin/badapp/BadAppServlet.sneezyMethod(BadAppServlet.java:332)	2	3 (%)
com/ibm/io/async/AsyncLibrary.aio_getioev2(Native Method)	2	3 (%)
NO JAVA STACK	2	3 (%)
com/ibm/jvm/Dump.JavaDump(Native Method)	1	2 (%)
com/ibm/issf/atjolin/badapp/BadAppServlet.sneezyMethod(BadAppServlet.java:337)	1	2 (%)
com/ibm/issf/atjolin/badapp/BadAppServlet.dopeyMethod(BadAppServlet.java:320)	1	2 (%)
java/net/PlainDatagramSocketImpl.receive0(Native Method)	1	2 (%)
java/net/SocketInputStream.socketRead0(Native Method)	1	2 (%)
com/ibm/misc/SignalDispatcher.waitForSignal(Native Method)	1	2 (%)

Thread detail: Thread aggregation analysis



Thread Type	Number of Threads : 66	Percentage
Thread	11	17 (%)
Alarm	6	9 (%)
WebContainer	5	8 (%)
Deferrable Alarm	4	6 (%)
SoapConnectorThreadPool	3	5 (%)
ThreadManager.JobsProcessorThread.InternalThread	1	2 (%)
WLMMonitorSleeper	1	2 (%)
ServerSocket	1	2 (%)
HAManager.thread.pool	1	2 (%)

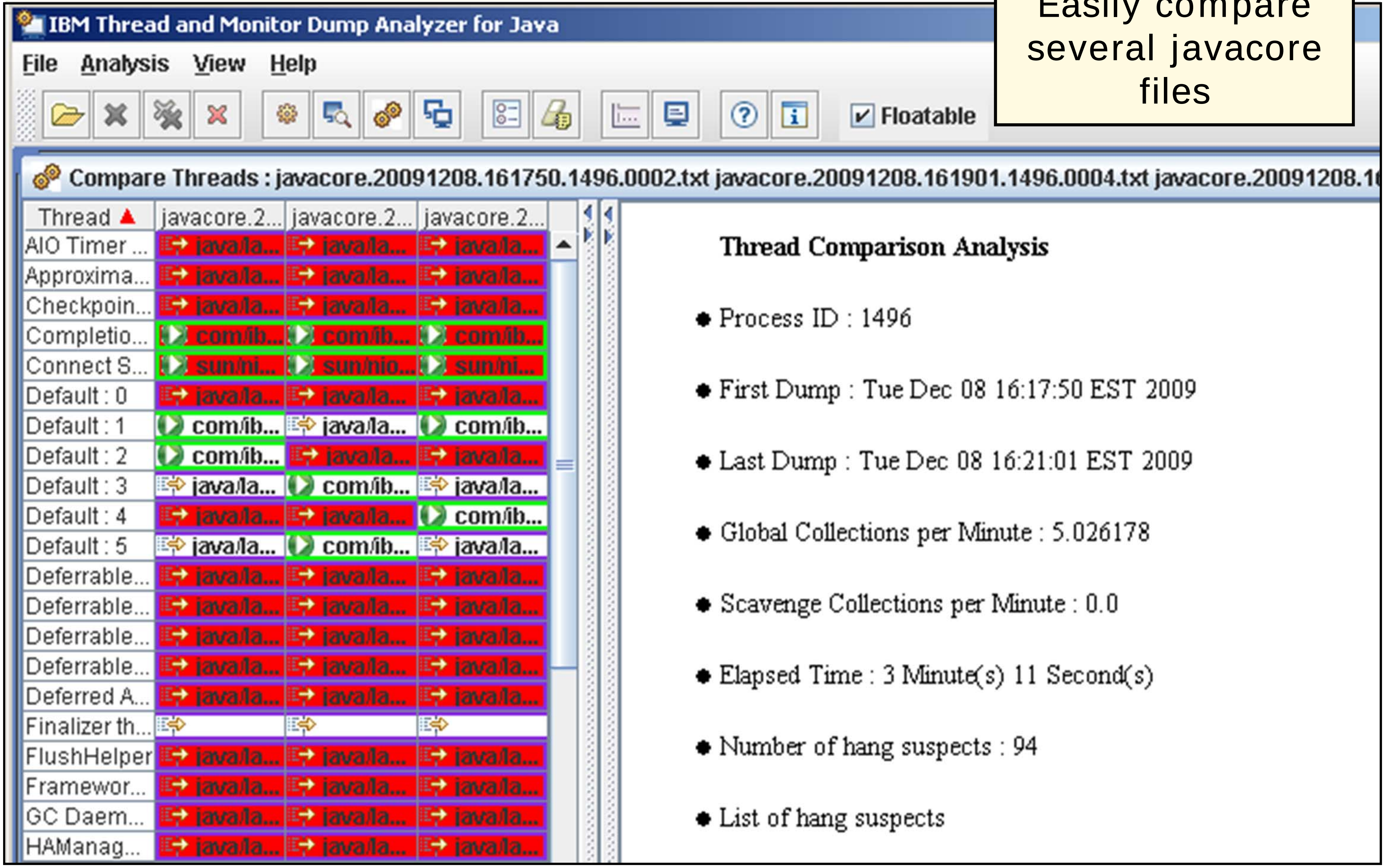
Thread detail: Memory segment analysis



Memory Type	# of Segments	Used Memory(bytes)	Used Memory(%)	Free Memory(bytes)	Free Memory(%)	Total Memory(bytes)
Internal	102	6,567,172	98.24	117,500	1.76	6,684,672
Object	1	65,131,520	100	0	0	65,131,520
Class	1,090	77,451,880	95.1	3,988,936	4.9	81,440,816
JIT Code Cache	7	0	0	3,670,016	100	3,670,016
JIT Data Cache	5	2,214,476	84.48	406,964	15.52	2,621,440
Overall	1,205	151,365,048	94.87	8,183,416	5.13	159,548,464

Multiple dump comparative analysis

Easily compare several javacore files



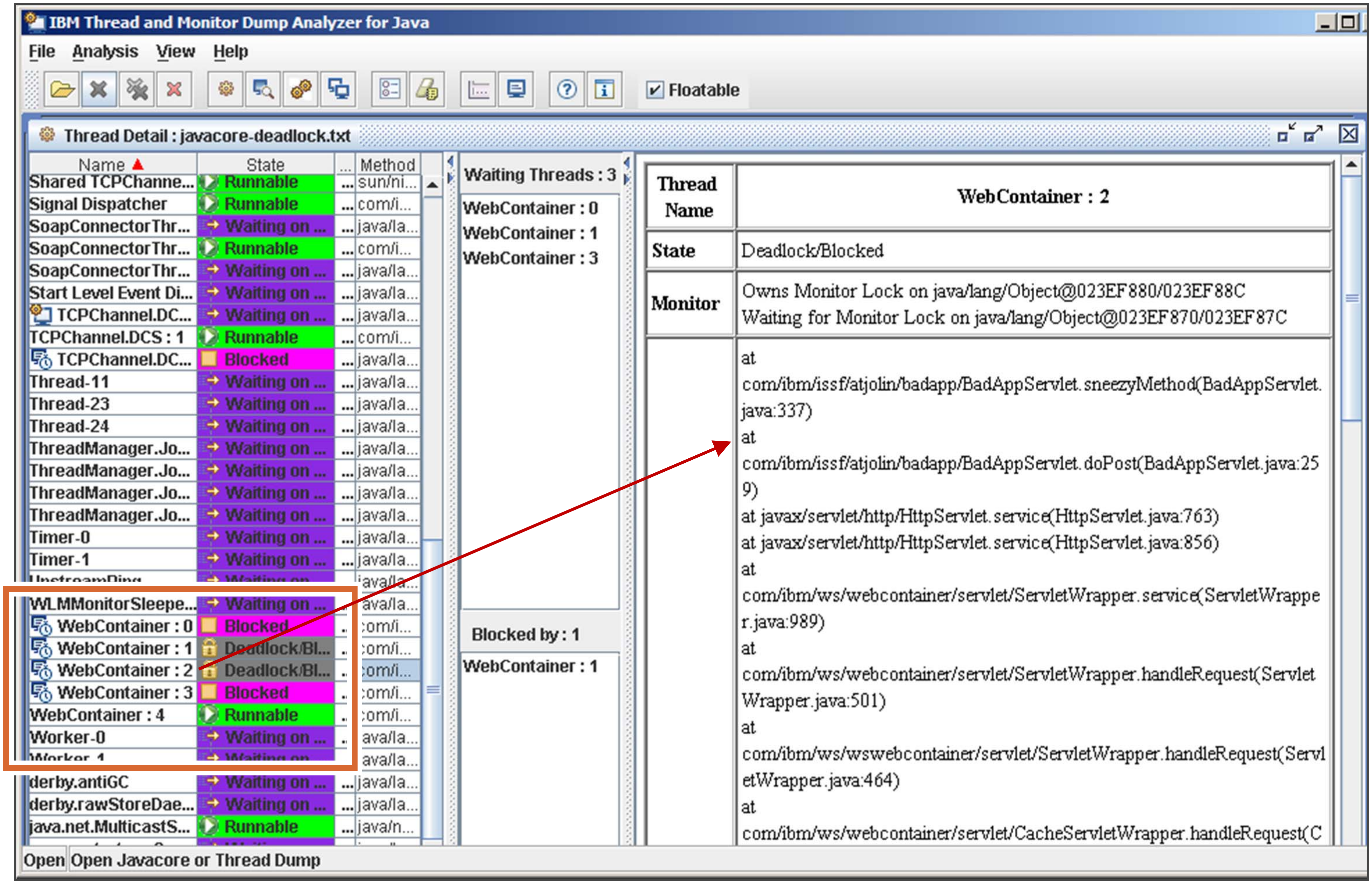
Compare Threads : javacore.20091208.161750.1496.0002.txt javacore.20091208.161901.1496.0004.txt javacore.20091208.16

Thread ▲	javacore.2...	javacore.2...	javacore.2...
AIO Timer ...	java/la...	java/la...	java/la...
Approxima...	java/la...	java/la...	java/la...
Checkpoin...	java/la...	java/la...	java/la...
Completio...	com/ib...	com/ib...	com/ib...
Connect S...	sun/ni...	sun/nio...	sun/ni...
Default : 0	java/la...	java/la...	java/la...
Default : 1	com/ib...	java/la...	com/ib...
Default : 2	com/ib...	java/la...	java/la...
Default : 3	java/la...	com/ib...	java/la...
Default : 4	java/la...	java/la...	com/ib...
Default : 5	java/la...	com/ib...	java/la...
Deferrable...	java/la...	java/la...	java/la...
Deferrable...	java/la...	java/la...	java/la...
Deferrable...	java/la...	java/la...	java/la...
Deferrable...	java/la...	java/la...	java/la...
Deferred A...	java/la...	java/la...	java/la...
Finalizer th...			
FlushHelper	java/la...	java/la...	java/la...
Framewor...	java/la...	java/la...	java/la...
GC Daem...	java/la...	java/la...	java/la...
HAManag...	java/la...	java/la...	java/la...

Thread Comparison Analysis

- Process ID : 1496
- First Dump : Tue Dec 08 16:17:50 EST 2009
- Last Dump : Tue Dec 08 16:21:01 EST 2009
- Global Collections per Minute : 5.026178
- Scavenge Collections per Minute : 0.0
- Elapsed Time : 3 Minute(s) 11 Second(s)
- Number of hang suspects : 94
- List of hang suspects

Thread analysis: Deadlocked thread details


IBM Thread and Monitor Dump Analyzer for Java
File Analysis View Help
Floatable
Thread Detail : javacore-deadlock.txt

Name	State	...	Method
Shared TCPChanne...	Runnable	...	sun/ni...
Signal Dispatcher	Runnable	...	com/i...
SoapConnectorThr...	Waiting on ...	...	java/la...
SoapConnectorThr...	Runnable	...	com/i...
SoapConnectorThr...	Waiting on ...	...	java/la...
Start Level Event Di...	Waiting on ...	...	java/la...
TCPChannel.DC...	Waiting on ...	...	java/la...
TCPChannel.DCS : 1	Runnable	...	com/i...
TCPChannel.DC...	Blocked	...	java/la...
Thread-11	Waiting on ...	...	java/la...
Thread-23	Waiting on ...	...	java/la...
Thread-24	Waiting on ...	...	java/la...
ThreadManager.Jo...	Waiting on ...	...	java/la...
ThreadManager.Jo...	Waiting on ...	...	java/la...
ThreadManager.Jo...	Waiting on ...	...	java/la...
ThreadManager.Jo...	Waiting on ...	...	java/la...
Timer-0	Waiting on ...	...	java/la...
Timer-1	Waiting on ...	...	java/la...
UpstreamPing	Waiting on		java/la...
WLMMonitorSleepe...	Waiting on ...		java/la...
WebContainer : 0	Blocked	..	com/i...
WebContainer : 1	Deadlock/Bl...	..	com/i...
WebContainer : 2	Deadlock/Bl...	..	com/i...
WebContainer : 3	Blocked	..	com/i...
WebContainer : 4	Runnable	..	com/i...
Worker-0	Waiting on ...		java/la...
Worker-1	Waiting on		java/la...
derby.antiGC	Waiting on ...	...	java/la...
derby.rawStoreDae...	Waiting on ...	...	java/la...
java.net.MulticastS...	Runnable	...	java/n...

Waiting Threads : 3

WebContainer : 0
WebContainer : 1
WebContainer : 3

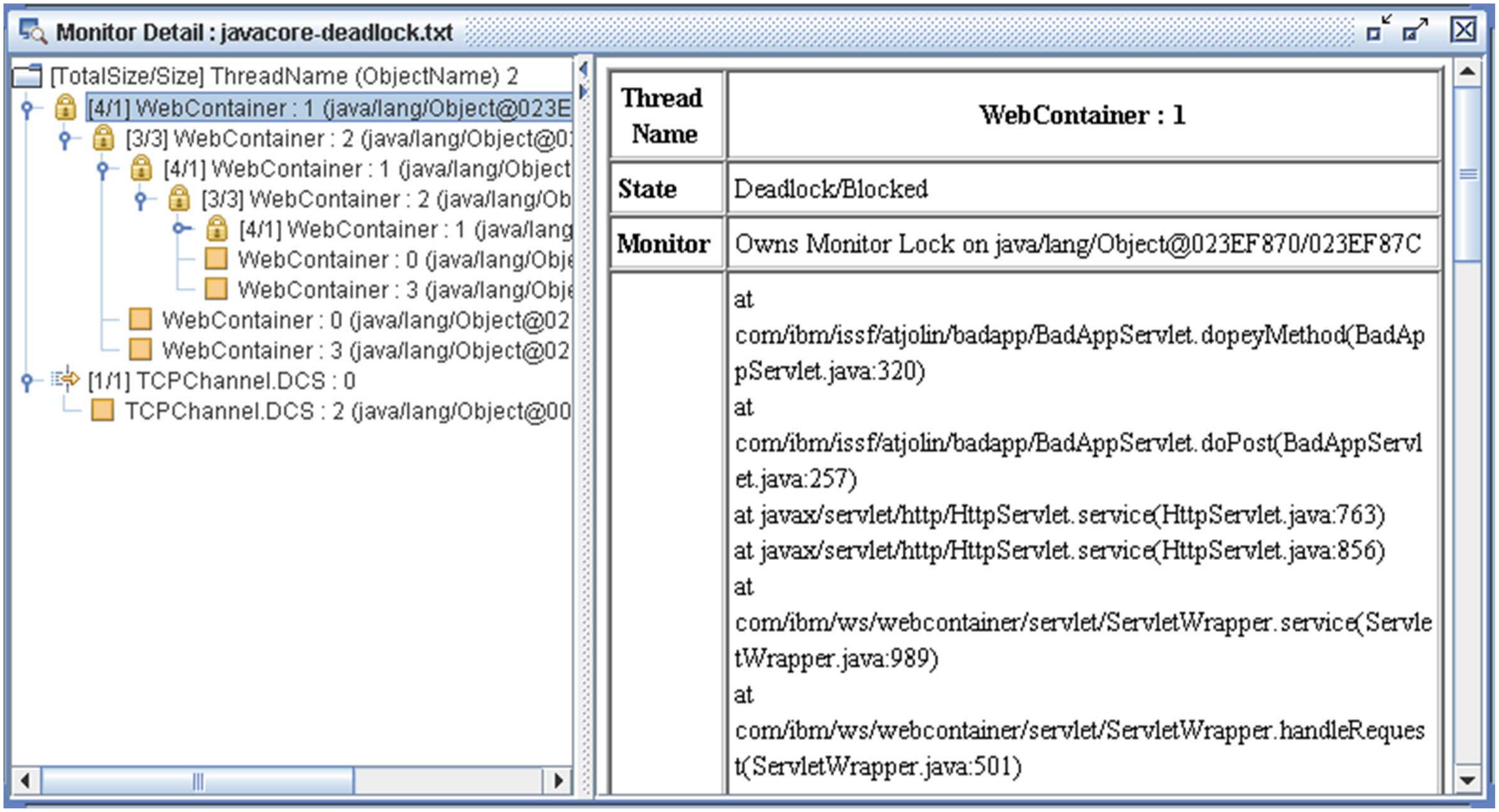
Blocked by : 1

WebContainer : 1

Thread Name	WebContainer : 2
State	Deadlock/Blocked
Monitor	Owns Monitor Lock on java/lang/Object@023EF880/023EF88C Waiting for Monitor Lock on java/lang/Object@023EF870/023EF87C
	at com/ibm/issf/atjolin/badapp/BadAppServlet.sneezyMethod(BadAppServlet.java:337) at com/ibm/issf/atjolin/badapp/BadAppServlet.doPost(BadAppServlet.java:259) at javax/servlet/http/HttpServlet.service(HttpServlet.java:763) at javax/servlet/http/HttpServlet.service(HttpServlet.java:856) at com/ibm/ws/webcontainer/servlet/ServletWrapper.service(ServletWrapper.java:989) at com/ibm/ws/webcontainer/servlet/ServletWrapper.handleRequest(ServletWrapper.java:501) at com/ibm/ws/wswebcontainer/servlet/ServletWrapper.handleRequest(ServletWrapper.java:464) at com/ibm/ws/webcontainer/servlet/CacheServletWrapper.handleRequest(C

Open Open Javacore or Thread Dump

Thread analysis: Monitor details



```
[TotalSize/Size] ThreadName (ObjectName) 2
[4/1] WebContainer : 1 (java/lang/Object@023E
  [3/3] WebContainer : 2 (java/lang/Object@01
    [4/1] WebContainer : 1 (java/lang/Object
      [3/3] WebContainer : 2 (java/lang/Ob
        [4/1] WebContainer : 1 (java/lang
        WebContainer : 0 (java/lang/Obje
        WebContainer : 3 (java/lang/Obje
    WebContainer : 0 (java/lang/Object@02
    WebContainer : 3 (java/lang/Object@02
[1/1] TCPChannel.DCS : 0
  TCPChannel.DCS : 2 (java/lang/Object@00
```

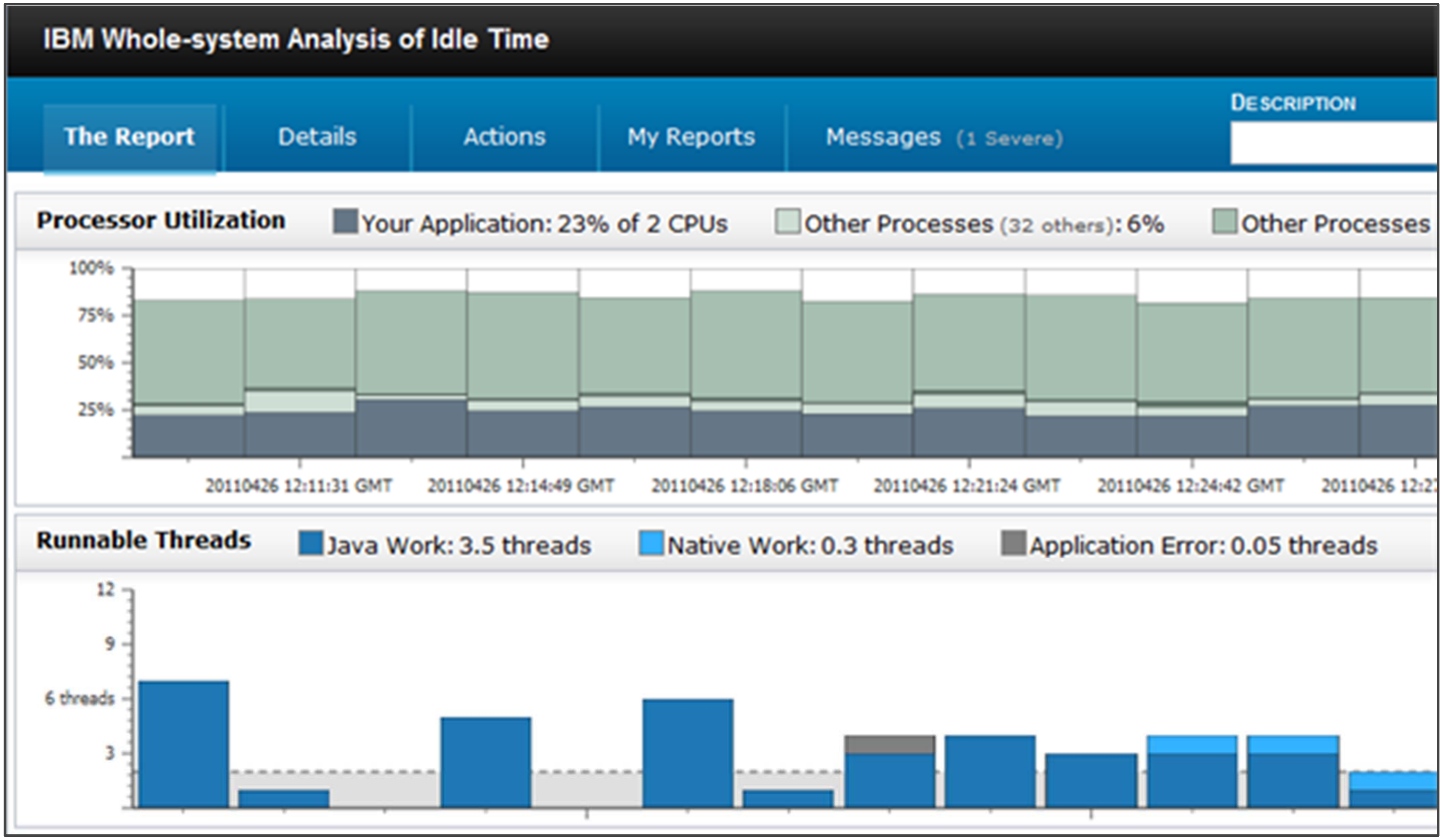
Thread Name	**WebContainer : 1**
State	Deadlock/Blocked
Monitor	Owns Monitor Lock on java/lang/Object@023EF870/023EF87C
	at com/ibm/issf/atjolin/badapp/BadAppServlet.dopeyMethod(BadAppServlet.java:320) at com/ibm/issf/atjolin/badapp/BadAppServlet.doPost(BadAppServlet.java:257) at javax/servlet/http/HttpServlet.service(HttpServlet.java:763) at javax/servlet/http/HttpServlet.service(HttpServlet.java:856) at com/ibm/ws/webcontainer/servlet/ServletWrapper.service(ServletWrapper.java:989) at com/ibm/ws/webcontainer/servlet/ServletWrapper.handleRequest(ServletWrapper.java:501)

IBM Whole-system Analysis of Idle Time (WAIT)

- New web-based tool from IBM Research
- WAIT report identifies bottlenecks such as:
  - Waiting for database data
  - Waiting on hot locks
  - Garbage collection degradation of performance
  - Inefficient code
- The report shows you the full stack context responsible for each bottleneck
- Access at `http://wait.ibm.com`
- Using WAIT
  - Generate javacore files on your system
  - Upload to WAIT-data is encrypted in transit
  - View report interactively in your browser

Example WAIT report

- The report can be viewed interactively by hovering the cursor over sections for more detail



Unit summary

Having completed this unit, you should be able to:

- Describe what a hang is
- Detect a hang condition
- Trigger and analyze javacore files for hung threads
- Use the WebSphere Application Server hang detection facility
- Use the IBM Thread and Monitor Dump Analyzer for Java

Checkpoint questions

1. True or false: The Thread Monitor facility is turned on by default for every application server.

2. True or false: A javacore file contains a dump of a JVM's threads at the time the file was generated.

3. True or false: An application can seem to be in a hung state if two threads are deadlocked.

4. True or false: The Thread and Monitor Dump Analyzer is part of the administrative console.

Checkpoint answers

1. True
2. True
3. True
4. False: The Thread and Monitor Dump Analyzer runs in the IBM Support Assistant.

Exercise 4

Troubleshooting hung threads

Exercise objectives

After completing this exercise, you should be able to:

- Use the thread monitor hang detection facility
- Generate and analyze various hung thread scenarios
- Use wsadmin to trigger a javacore file
- Analyze the javacore file for hung threads by using the IBM Thread and Monitor Dump Analyzer for Java tool