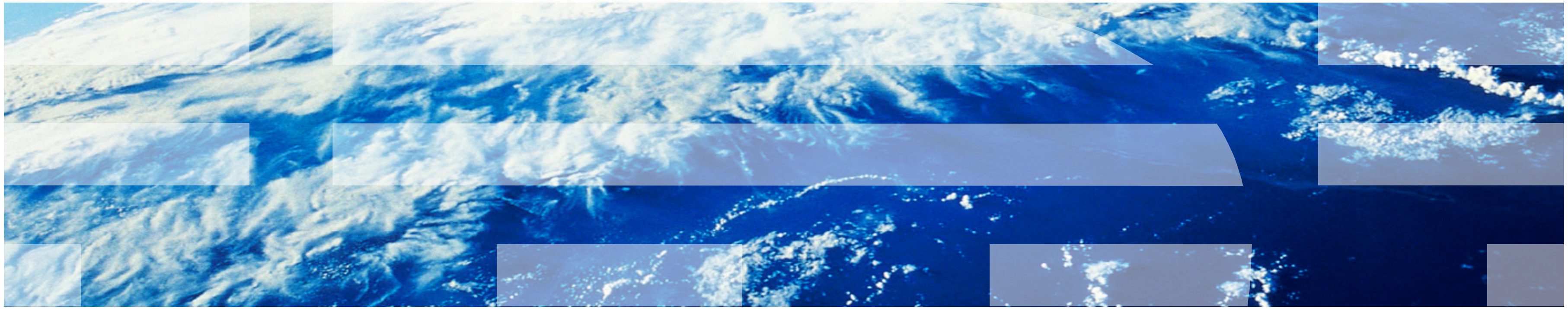
WebSphere Application Server

Troubleshoot WebSphere Application Server (The Basics)

Kevin Grigorenko – IBM WAS SWAT Software Engineer



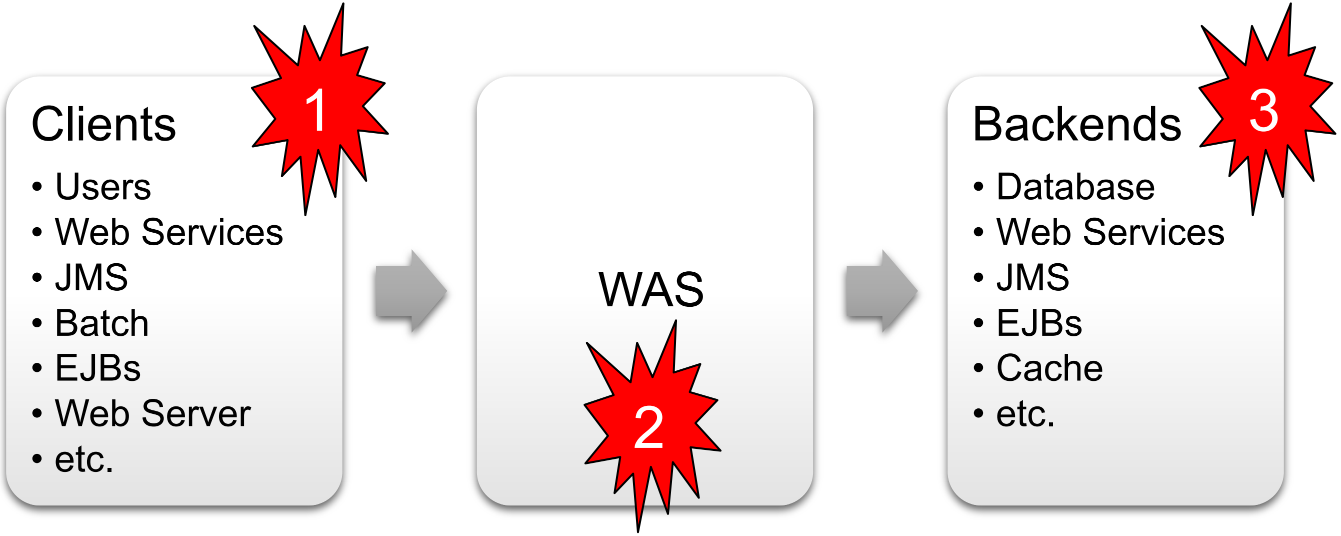
Context

- Your systems are slow, crashing, or returning errors, and your customers and management are upset.
- You don't have time to install some tool or run traces or a MustGather.
- Before you click that restart button (which might not even help), what should you do?
- This presentation covers what we recommend you do to understand most problems in just a few minutes for Traditional WebSphere Application Server and Liberty.
- We'll cover:
  - Where's the problem?
  - If it's in WAS, what's the problem?

Summary

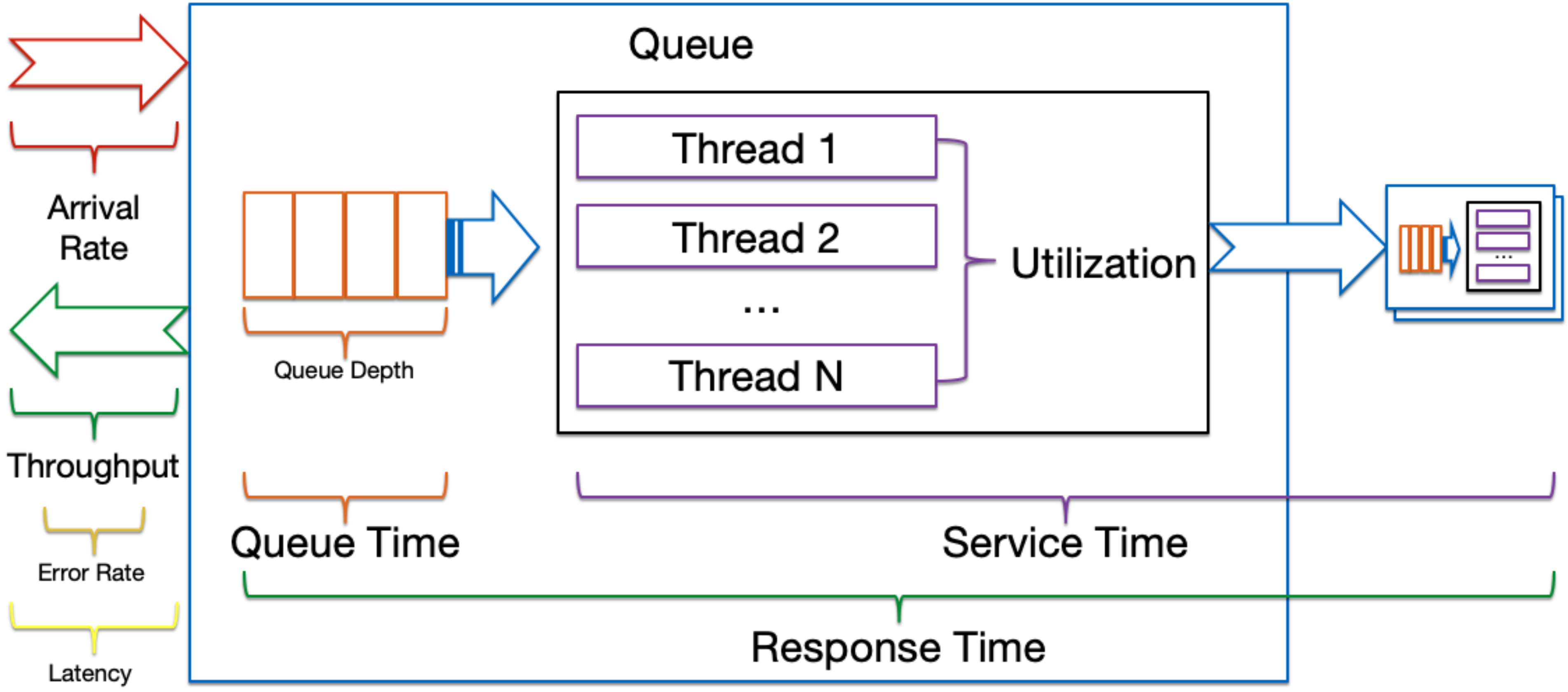
- Configure WAS Slow Request detection to your maximum expected response time + 20% and check for the warning in the logs.
- Use WAS thread dumps and logs to understand if the issue is upstream of WAS, within WAS, or downstream of WAS.
- Configure automation to easily create, analyze, and gather diagnostics across many machines at once.

The WAS Point-of-View



Quickly investigate all: WAS Thread Dumps and Logs

Queuing Network



What I’m Not Saying

- I’m not saying monitoring products aren’t great. They’re great. We recommend you use them. Otherwise, it’s like flying an airplane without instruments.
- However, to depend on them, you must create and interpret your dashboards wisely:
    - If response times are good, but users are complaining:
        - Request arrival rate might be low, or some requests are hung (thus not counting in response times)
    - If throughput is good, but users are complaining:
        - Users might be receiving responses but they’re errors (even with HTTP 200)
    - Even if you have a great dashboard, you might still need to know why WAS is stuck.
- Thread dumps on WAS tell you exactly what it’s doing.

What I’m Not Saying

- I’m not saying log centralization isn’t great. It’s great. We recommend you use it. Otherwise, it’s hard to search all logs.
- However, an event must occur to be in the logs. A hung thread doesn’t trigger a log event for a while by default:
  - 10 minutes for Traditional WAS
  - Disabled by default on Liberty (enable requestTiming-1.0)
- Thread dumps on WAS tell you about hung threads.

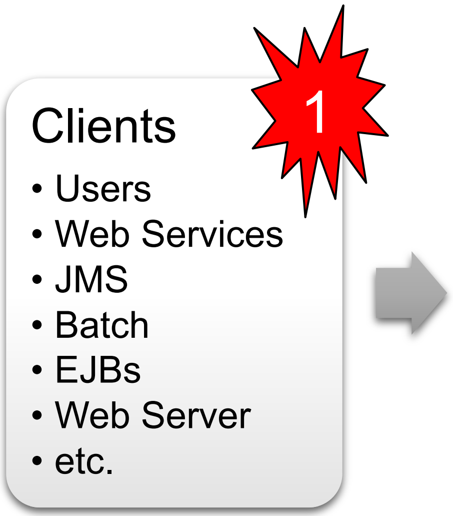
What I’m Not Saying

- I’m not saying Operating System monitoring isn’t great. It’s great. We recommend you use it to check for obvious problems like 100% CPU or RAM paging.
- However, top (and similar tools on other operating systems) can only tell you so much.
    - Low CPU might be healthy, or low request arrival rate, or high lock contention, etc.
- Netstat might show a spike in sockets, but is that because WAS is slow, or the backend is slow and things just backed up into WAS?
- Thread dumps on WAS tell you about locking, idle threads, hung threads, etc.

What I’m Not Saying

- I’m not saying MustGathers aren’t great. They’re great. If you can get them, get them.
- Search for “MustGather: Performance, hang, or high CPU issues”
    - Choose the script for your operating system.
- The performance MustGather takes about 5 minutes to run and gathers thread dumps and operating system statistics.
- Upload the resulting zip file it creates as well as the thread dumps and WAS logs to a support case for IBM to review.
- Place the performance MustGather script on all boxes ahead-of-time, e.g. /opt/ibm/linperf.sh
- But let’s be honest, it’ll take significant time to open the case, upload the files, wait for the right technician to look at them, provide a review (of varying depth), etc.
- How do you do this same analysis with thread dumps... And across a cluster...

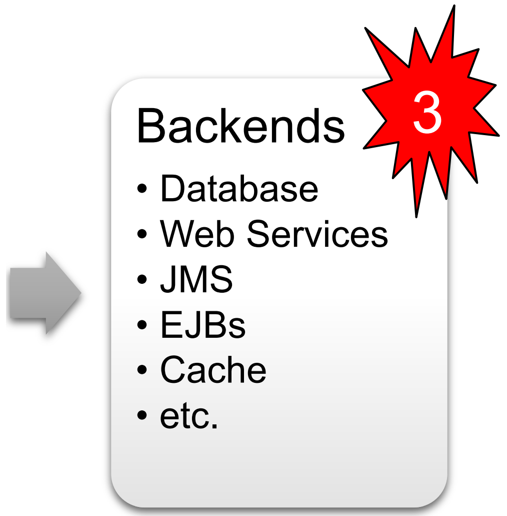
Client Issues



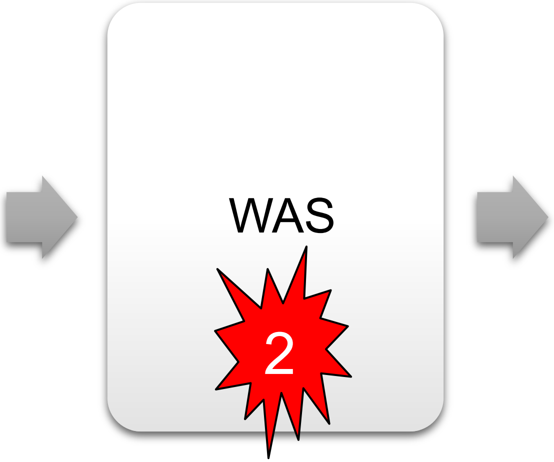
- Client issues from the point-of-view of WAS:
  - Idle threads in WAS
    - TechNote: "Identifying idle threads in thread dumps taken against WebSphere Application Server"

Backend Issues

- Backend issues from the point-of-view of WAS:
  - Waiting threads in:
    - java.net.SocketInputStream.socketRead0



Backend Issues



- Issues in WAS:
  - Thread Dumps: Lots of activity (high CPU), Lock Contention
  - WAS Logs: Errors, Garbage collection

Thread Pools

- Thread dumps have a lot of threads. Focus on where your application work happens:

Work	Traditional WAS	Liberty
HTTP	WebContainer	Default Executor
JMS (SIB)	SIBJMSRAThreadPool	Default Executor
JMS (MQ)	WMQJCAResourceAdapter or MessageListenerThreadPool	Default Executor
EJB	ORB.thread.pool	Default Executor
z/OS	WebSphere WLM Dispatch Thread	Default Executor

WAS Thread Pools Tips

- Traditional WAS:
  - Do not set application thread pools to “growable”. This risks OutOfMemoryErrors and DDoS attacks.
  - You may print periodic thread pool statistics to SystemOut.log with diagnostic trace: *=info:Runtime.ThreadMonitorHeartbeat=detail

    -
      ```
      [1/12/20 19:38:15:208 GMT] 000000d4 ThreadMonitor A   UsageInfo[ThreadPool:hung/active/size/max]={
        SIBFAPThreadPool:0/2/4/50,
        WebContainer:0/3/4/12,
        SIBJMSRAThreadPool:0/0/10/41,
        Default:0/2/7/20,
        ORB.thread.pool:0/0/10/77,
      }
      ```
- WAS Liberty:
  - By default, Liberty has an unlimited-size, self-tuning thread pool designed to maximize throughput. You can override this with <executor maxThreads=”50" />

WAS Slow Request Detection

- We’ll discuss thread dumps as active investigation mechanisms, but what can you do proactively so that WAS tells you when there might be a problem?
- Both Traditional WAS and Liberty provide slow request detection.
- Traditional WAS calls this “hung thread detection” but the threads may or may not be hung; all that can be said is that they are taking longer to execute than the configured threshold.
- We recommend figuring out with your application team and business what is your maximum expected response time, and then add 20% to that. This should be your slow request threshold.

Traditional WAS Slow Request Detection

- By default, every 3 minutes, check if request has been processing for more than 10 minutes.
- Configure these:
  - Servers > Application Servers > ${SERVER} > Administration > Custom Properties
    ```
    com.ibm.websphere.threadmonitor.threshold=${SECONDS}
    com.ibm.websphere.threadmonitor.interval=${SECONDS}
    ```
- The check is very cheap: it just compares each thread's start timestamp to the current timestamp. You can safely set the interval to very low; as low as a few seconds.

-
  ```
  [11/16/09 12:41:03:296 PST] 00000020 ThreadMonitor W   WSVR0605W: Thread
  "WebContainer : 0" (00000021) has been active for 655546 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) ...
  ```
- On z/OS, also see Dispatch Progress Monitor (DPM)

Liberty Slow Request Detection

- Not enabled by default. Add the feature and configure it:

  -
    ```
    <featureManager>
        <feature>requestTiming-1.0</feature>
    </featureManager>

    <requestTiming
        slowRequestThreshold="10s"
        hungRequestThreshold="600s"
        sampleRate="1"
    />
    ```
- More expensive than Traditional WAS but also more powerful. For high volume traffic, increase the sampleRate.
- The hungRequestThreshold automatically takes multiple thread dumps.

Liberty Slow Request Detection

- Example output:

```
TRAS0112W: Request AAC9KLwFFXT_AAAAAAAAAAN has been running on thread 0000006b for 1549.460ms. The following stack trace
   shows what this thread is currently doing.

        ${THREAD DUMP}

The following table shows the events that have run during this request.

Duration     Operation
1552.012ms + websphere.servlet.service | DayTrader Web | TradeScenarioServlet
   0.014ms       websphere.session.getAttribute | R-ObCtcDfR8Zd9riQEMCh6R | uidBean
  30.714ms       websphere.servlet.service | DayTrader Web | TradeAppServlet
   0.010ms           websphere.session.getAttribute | R-ObCtcDfR8Zd9riQEMCh6R | uidBean
  30.456ms           websphere.servlet.service | DayTrader Web | /quote.jsp
  28.903ms               websphere.servlet.service | DayTrader Web | /displayQuote.jsp
   0.194ms                   websphere.datasource.psExecuteQuery | jdbc/TradeDataSource | SELECT t0.CHANGE1, t0.COMPANYNAME, t0.HIGH,
1520.695ms +     websphere.servlet.service | DayTrader Web | TradeAppServlet
   0.013ms           websphere.session.getAttribute | R-ObCtcDfR8Zd9riQEMCh6R | uidBean
522.450ms           websphere.datasource.psExecuteUpdate | jdbc/TradeDataSource | INSERT INTO orderejb (ORDERID, COMPLETIONDATE, OPE
 230.333ms           websphere.datasource.psExecuteUpdate | jdbc/TradeDataSource | INSERT INTO holdingejb (HOLDINGID, PURCHASEDATE, P
   0.936ms           websphere.datasource.psExecuteUpdate | jdbc/TradeDataSource | UPDATE accountejb SET BALANCE = ? WHERE ACCOUNTID
```

What are Thread Dumps?

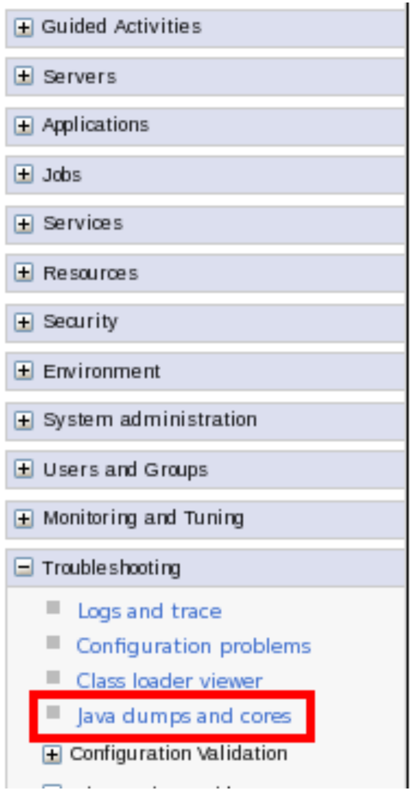
- Thread dumps primarily show what each thread is doing.
- They are a snapshot in time. The more thread dumps you have, the more you can conclude, statistically.
- IBM Java and OpenJ9: Written to a javacore.YYYYMMDD.HHMMSS.PID.ID.txt file in the current working directory of the process (you may also override the directory).
    - IBM calls a thread dump a Java Core (or Java Dump), but it's not like an OS core
    - On Linux, you can go to the CWD with: cd /proc/${PID}/cwd
- HotSpot Java: Written to stdout (Traditional WAS: native_stdout.log, Liberty: console.log)

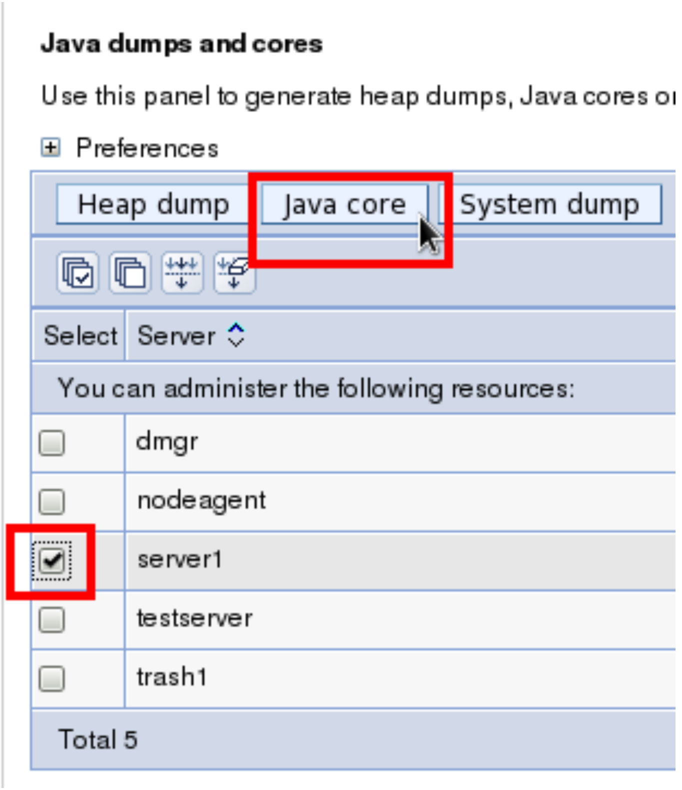
Thread Dumps Costs and Risks

- In general, thread dumps are very cheap: they're small (a few MB) and they pause the JVM for a few hundred milliseconds at most.
- However, this changes if you've overridden the default thread dump handler to also perform heapdumps:
  - IBM Java/OpenJ9: -Xdump:heap/system:events=user
  - HotSpot: -XX:+HeapDumpOnCtrlBreak
  - The above are not recommended. If you need to gather heapdumps, there are other ways that are just as easy to use. Leave thread dumps to be a fast tool.
- Everything has risks. There have been bugs in the past causing freezes and crashes: IV68447 (Fixed in Java 6SR17), IZ89711 (Fixed in Java 6SR10). I haven't heard of such issues recently.

How do you get a Thread Dump?

- For Traditional WAS, you can use the Admin Console:




Java dumps and cores

Use this panel to generate heap dumps, Java cores o

Select	Server
You can administer the following resources:	
☐	dmgr
☐	nodeagent
✓	server1
☐	testserver
☐	trash1
Total 5	

How do you get a Thread Dump?

- For POSIX-based Operating Systems (Linux, AIX, Solaris, HP-UX, IBM i, macOS):
  - kill -3 ${PID}
  - Linux: pkill -3 -f ${CMDLINESEARCH}
- Windows: windowsperf.jacl from the Performance MustGather
- Traditional WAS: wsadmin (works on Windows):
  - AdminControl.invoke(AdminControl.completeObjectName("type=JVM,process=${SERVER},*"), "dumpThreads")
- Liberty (works on Windows): bin/server javadump ${SERVER}
- IBM Java/OpenJ9 (works on Windows): Java Surgery: https://www.ibm.com/support/pages/ibm-runtime-diagnostic-code-injection-java-platform-java-surgery
  - java -jar surgery.jar -pid ${PID} -command JavaDump

How do you get a Thread Dump?

- HotSpot Java:
  - jcmd ${PID} Thread.print
- Open J9:
  - jcmd ${PID} Dump.java
- Important note for Traditional WAS on z/OS:
  - By default, the Admin Console "Java core" button and wsadmin both cause a heapdump and SVCDUMP in addition to a javacore. Change this just to a javacore with wsadmin_dumpthreads_enable_heapdump=0 and wsadmin_dumpthreads_enable_javatdump=0
  - Alternatively: MODIFY $CONTROLLER,JAVACORE

Basic Thread Dump Playbook

- Gather first thread dump
- Wait 5-30 seconds
- Gather second thread dump
- Wait 5-30 seconds
- Gather third thread dump

You have the thread dumps. Now what?

- You could download the files and open them in a tool like the IBM Thread and Monitor Dump Analyzer (TMDA).
- However, we know how time-consuming that can be, so here's a quicker approach.
- Right on the box itself, you can just use operating system tools to search the files.
    1. Figure out the thread pools where your application work runs.
    2. Search the thread dump(s) for the stack tops for all those thread pools.
    3. Aggregate the stack tops to figure out what most threads are doing.
- We'll use Linux on IBM Java as an example, but you can adapt these techniques to HotSpot Java and for Windows (Batch/PowerShell), z/OS, IBM i, etc.

Quickly analyze IBM Java thread dumps on Linux

- $ grep -H -A 11 "3XMTHREADINFO.***WebContainer**" javacore*.txt | grep -v -e "at sun/misc/Unsafe.park" -e "java/util/concurrent" -e "at java/lang/Object" | grep -A 1 3XMTHREADINFO3 | grep 4XESTACKTRACE | sed 's/(.*//g' | sort | uniq -c | sort -k2,2 -k1,1

```
 1 javacore.20200414.173551.1495.0001.txt-4XESTACKTRACE
      at com/ibm/io/async/AsyncLibrary.aio_getioev3
 5 javacore.20200414.173551.1495.0001.txt-4XESTACKTRACE
      at com/ibm/ws/util/BoundedBuffer$GetQueueLock.await
```

- If we consult the “Idle thread stacks” TechNote, we find that these are all idle, so we have 6 idle WebContainer threads.
- Remember though: A thread dump is just a snapshot. Just because it was idle at that exact moment, doesn’t mean it’s always idle. Analyze more thread dumps for clarity.

Quickly analyze IBM Java thread dumps on Linux

- $ grep -H -A 11 "3XMTHREADINFO.*WebContainer" javacore*.txt | grep -v -e "at sun/misc/Unsafe.park" -e "java/util/concurrent" -e "at java/lang/Object" | grep -A 1 3XMTHREADINFO3 | grep 4XESTACKTRACE | sed 's/(.*//g' | sort | uniq -c | sort -k2,2 -k1,1

```
 1 at com/ibm/ws/wsoc/LinkRead.processRead
 1 at java/math/DivisionLong.monPro
 1 at java/net/SocketOutputStream.socketWrite0
 1 at org/apache/soap/util/xml/DOM2Writer.normalize
 3 at com/ibm/io/async/AsyncLibrary.aio_getioev3
 6 at com/ibm/ws/security/auth/rsatoken/RSAPropagationToken.rsaSignBytes
 9 at java/net/SocketInputStream.socketRead0
19 at com/ibm/ws/util/BoundedBuffer$GetQueueLock.await
```

- 19 threads are idle and there a few suspects: 9 in socketRead and 6 in RSA
- Open the thread dump (e.g. vi/less/more) and search for those frames and see the stack.

Analyze IBM Java thread dump stacks

- After searching for that stack frame, scroll up to make sure you're looking at the right thread pool. And look down to see what is driving that stack frame. Example:

```
3XMTHREADINFO      "WebContainer : 30" J9VMThread:0x628A500, omrthread_t:0x147CB49230, java/lang/Thread:0x4B7E52F0, state:R, prio=5
3XMJAVALTHREAD            (java/lang/Thread getId:0x12F, isDaemon:true)
3XMTHREADINFO1            (native thread ID:0x11E38, native priority:0x5, native policy:UNKNOWN, vmstate:R, vm thread flags:0x000000a1)
3XMTHREADINFO2            (native stack address range from:0x0000147CC1606000, to:0x0000147CC1646000, size:0x40000)
3XMCPUTIME               CPU usage total: 1.314677751 secs, current category="Application"
3XMHEAPALLOC             Heap bytes allocated since last GC cycle=3224912 (0x313550)
3XMTHREADINFO3           Java callstack:
4XESTACKTRACE                at java/net/SocketInputStream.socketRead0(Native Method)
4XESTACKTRACE                at com/ibm/jsse2/b.a(b.java:297(Compiled Code))
4XESTACKTRACE                at org/apache/soap/util/net/HTTPUtils.post(HTTPUtils.java:603(Compiled Code))
4XESTACKTRACE                at com/example/application/CustomerHandler.execute(CustomerHandler.java:120(Compiled Code))
```

- Waiting on a Database Example:

```
3XMTHREADINFO3           Java callstack:
4XESTACKTRACE                at java.net.SocketInputStream.socketRead0(Native Method)
4XESTACKTRACE                at java.net.SocketInputStream.read(SocketInputStream.java:140)
4XESTACKTRACE                at com.ibm.db2.jcc.t4.z.b(z.java:199)
```

Analyzing Thread Stacks

- If the stacks look like gobbledygook to you, that’s understandable. Either work with your application developers or open an IBM Support case and we can help you understand what they mean.
- Over time, you can create a mapping of what the various stack frames mean for your environment.
- Tip: The command above looks at the top stack frame, but you can also do a tree analysis and search for things like common application servlets and break it down by that. For example, 10% of threads are in Servlet1 and 25% are in MDB2, etc.
- Remember to look at **all** relevant thread pools

Analyze Thread Dump Lock Contention

- Analyzing lock contention is a bit more complicate than thread stacks.
- Here’s an example of a blocked thread:

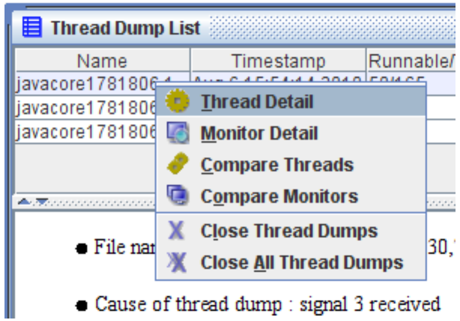
```
3XMTHREADINFO "WebContainer : 6" J9VMThread:0x0000000003880100, omrthread_t:0x0000147CD8049188, java/lang/Thread:0x49C52340, state:B, prio=5
3XMTHREADBLOCK     Blocked on: java/lang/Object@0x000000004A2F9468 Owned by: "WebContainer : 18" (J9VMThread:0x0000000005FD5B00,
java/lang/Thread:0x000000004B5E0A40)
3XMHEAPALLOC             Heap bytes allocated since last GC cycle=1227368 (0x12BA68)
3XMTHREADINFO3           Java callstack:
4XESTACKTRACE              at com/ibm/ws/security/auth/rsatoken/RSAPropagationToken.rsaSignBytes(RSAPropagationToken.java:760(Compiled Code))
```

- Then search for the ”Owned by” thread:

```
3XMTHREADINFO      "WebContainer : 18" J9VMThread:0x0000000005FD5B00, omrthread_t:0x147CB005C0F0, java/lang/Thread:0x4B5E0A40, state:R, prio=5
3XMTHREADINFO3           Java callstack:
4XESTACKTRACE              at java/math/BigInteger.modPow(BigInteger.java:1415(Compiled Code))
4XESTACKTRACE              at com/ibm/crypto/provider/RSACore.a(Bytecode PC:117(Compiled Code))
4XESTACKTRACE              at java/security/Signature$Delegate.engineSign(Signature.java:1219(Compiled Code))
4XESTACKTRACE              at com/ibm/ws/security/auth/rsatoken/RSAPropagationToken.rsaSignBytes(RSAPropagationToken.java:766(Compiled Code))
5XESTACKTRACE                (entered lock: java/lang/Object@0x000000004A2F9468, entry count: 1)
```

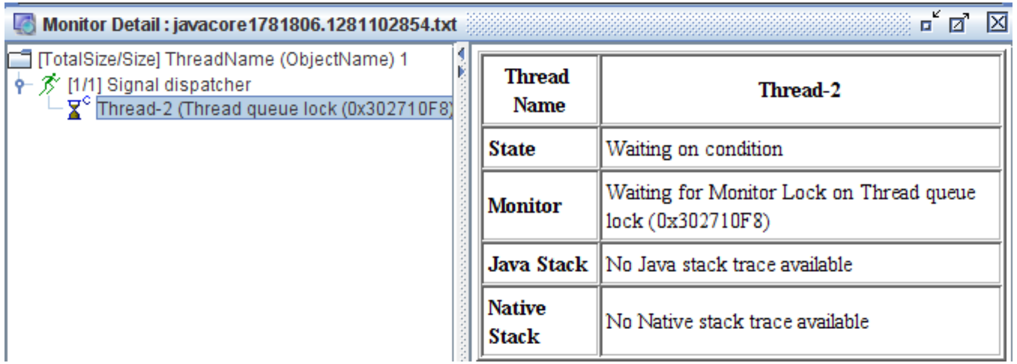
IBM Thread and Monitor Dump Analyzer

- https://www.ibm.com/support/pages/ibm-thread-and-monitor-dump-analyzer-java-tmda


Thread Dump List

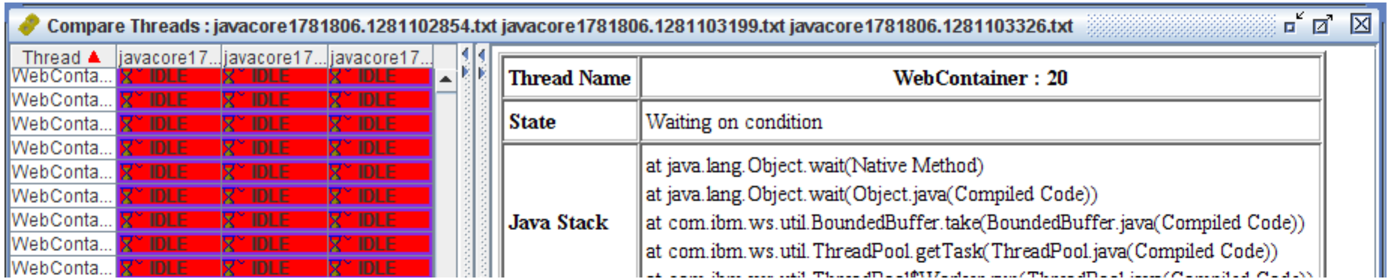
Name	Timestamp	Runnable/
javacore1781806.1...	Aug 6 15:54:14 2010	59/165
javacore1781806		
javacore1781806		

- File nar ... 30,
- Cause of thread dump : signal 3 received


Monitor Detail : javacore1781806.1281102854.txt

```
[TotalSize/Size] ThreadName (ObjectName) 1
  [1/1] Signal dispatcher
    Thread-2 (Thread queue lock (0x302710F8)
```

Thread Name	Thread-2
State	Waiting on condition
Monitor	Waiting for Monitor Lock on Thread queue lock (0x302710F8)
Java Stack	No Java stack trace available
Native Stack	No Native stack trace available

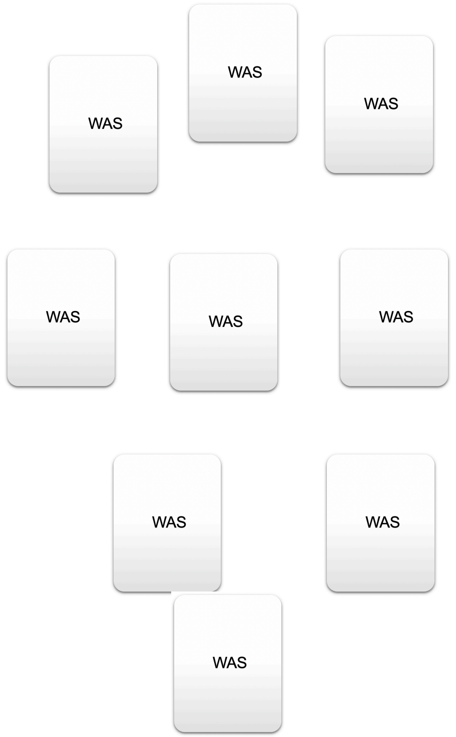

Compare Threads : javacore1781806.1281102854.txt javacore1781806.1281103199.txt javacore1781806.1281103326.txt

Thread ▲	javacore17...	javacore17...	javacore17...
WebConta...	IDLE	IDLE	IDLE
WebConta...	IDLE	IDLE	IDLE
WebConta...	IDLE	IDLE	IDLE
WebConta...	IDLE	IDLE	IDLE
WebConta...	IDLE	IDLE	IDLE
WebConta...	IDLE	IDLE	IDLE
WebConta...	IDLE	IDLE	IDLE
WebConta...	IDLE	IDLE	IDLE
WebConta...	IDLE	IDLE	IDLE

Thread Name	WebContainer : 20
State	Waiting on condition
Java Stack	at java.lang.Object.wait(Native Method) at java.lang.Object.wait(Object.java(Compiled Code)) at com.ibm.ws.util.BoundedBuffer.take(BoundedBuffer.java(Compiled Code)) at com.ibm.ws.util.ThreadPool.getTask(ThreadPool.java(Compiled Code))

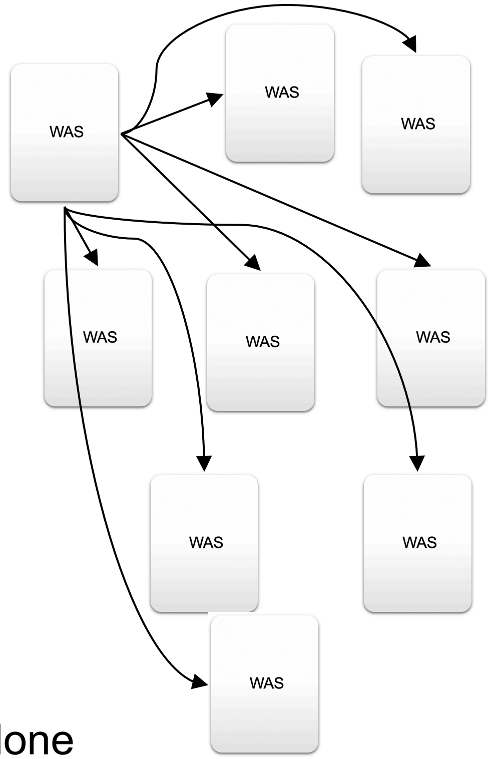
Automation

- Given complexities of running commands (jump boxes, security, approval, etc.), this causes slow and/or incomplete data collection. Moreover, how does one reasonably get this sort of data on 10s or 100s of machines?
- Invest time into automation. It has a huge return on investment in reducing time to resolution and reducing downtime.
- Given security restrictions, the easiest way to do this is to have an “automation” box on the same LAN as your boxes. For example, the deployment manager.
- Consult your security team to properly plan this out. They may have an alternative product that does this (with auditing, for example).
- For example, on Linux, you can use a password-protected SSH key and ssh-agent.



Automation

- Generate an “orchestrator” SSH key on some box:
  - ssh-keygen -t rsa -b 4096 -f ~/.ssh/orchestrator
- Push the public key to all the boxes:
  - ssh-copy-id -i ~/.ssh/orchestrator user@host
- When you need to use it, start ssh-agent:
  - eval $(ssh-agent)
- Load the key into the agent and enter its password:
  - ssh-add ~/.ssh/orchestrator
- Perform actions on multiple boxes automatically:
  - for i in host1 host2; do ssh user@$i ”pkill -3 -f AppServer"; done



WAS Logs

- Search for warnings and errors in SystemOut: grep " [W|E] ” SystemOut* messages.log
- Search for JVM errors or warnings: grep JVM native_stderr.log console.log
- You should have verbose garbage collection enabled (<1% overhead, enabled by default starting in Traditional WAS 9):
  - IBM Java/OpenJ9: -Xverbosegclog:verbosegc.%seq.log,20,50000
  - Hotspot: -Xloggc:verbosegc.log -XX:+UseGCLogFileRotation -XX:NumberOfGCLogFiles=5 -XX:GCLogFileSize=20M -XX:+PrintGCDateStamps -XX:+PrintGCDetails
- Example searching IBM Java/OpenJ9 verbosegc for full GCs:

  -
    ```
    $ grep -A 3 "cycle-end.*type=.global." verbosegc.001.log | grep durationms
    <exclusive-end id="46934" timestamp="2019-11-06T19:51:38.287" durationms="116.092" />
    <exclusive-end id="46958" timestamp="2019-11-06T19:51:38.422" durationms="135.308" />
    <exclusive-end id="46982" timestamp="2019-11-06T19:51:38.521" durationms="97.973" />
    <exclusive-end id="47006" timestamp="2019-11-06T19:51:38.631" durationms="109.742" />
    ```

Linux: top -H

- You can use $(top -H): CPU usage by thread.

-
```
top - 16:58:14 up 2 min,  5 users,  load average: 1.34, 0.50, 0.18
Tasks: 799 total,   5 running, 792 sleeping,   0 stopped,   2 zombie
Cpu(s): 37.8%us,  0.2%sy,  0.0%ni, 61.9%id,  0.0%wa,  0.0%hi,  0.0%si,  0.0%st
Mem:  15943268k total,  3366124k used, 12577144k free,    87824k buffers
Swap:     65532k total,        0k used,    65532k free,  1715160k cached

  PID USER       PR  NI  VIRT  RES  SHR S %CPU %MEM    TIME+  COMMAND
 6607 user1      20   0 1992m  18m 9164 R 100.1  0.1   0:20.67 WebContainer :
 6730 user1      20   0 1992m  19m 9144 R 100.1  0.1   0:12.40 WebContainer :
 6806 user1      20   0 1992m  27m 8900 R 99.8   0.2   0:07.38 WebContainer :
```

- Thread names may be cut off, but still often give a good idea

- nohup top -b -d 30 -H | grep -A 50 "top - " >> top`date +%Y%m%d%H%M%S`.out &

Take-Aways

- Configure WAS Slow Request detection to your maximum expected response time + 20% and check for the warning in the logs.
  - On Traditional WAS, the interval and threshold can be very low. On Liberty, tune sampleRate as needed.
- Use WAS thread dumps and logs to understand if the issue is upstream of WAS, within WAS, or downstream of WAS.
  - Thread dumps are cheap: a few MB each and take just a few hundred milliseconds to create.
- Configure automation to easily create, analyze, and gather diagnostics across many machines at once.
  - Check with your security team to do it the way your organization prefers.

Thank you.
Questions?

- April 22nd: Self-paced WebSphere Application Server Troubleshooting and Performance Lab
  - This presentation will discuss and demonstrate the WebSphere Application Server Troubleshooting and Performance Lab on Docker which is a free, publicly downloadable Traditional WAS and WAS Liberty troubleshooting and performance lab.

    This presentation will cover the basics of the lab itself, how to install and use it, and demonstrate some of the key WAS-related troubleshooting and performance tools that administrators should use day-to-day.
  - http://ibm.biz/troubleshoot2

Appendix

Garbage Collection Policies

- Choose the Garbage Collection policy that meets your needs
- IBM Java:

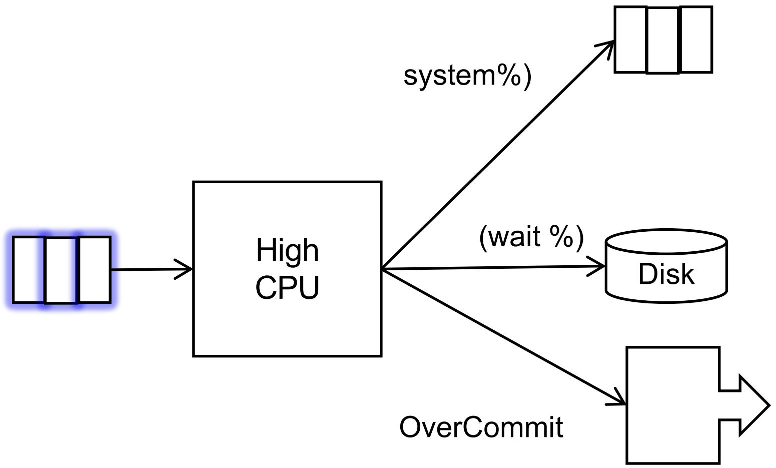
gencon	balanced	optthruput	optavgpause	metronome
•General •Balances throughput and pause times •Occasional long pauses	•Heaps > 10GB •Also generational •Deals with long pauses in gencon •Optimize for NUMA	•Batch •Deals with long generational pause times	•When optthruput has long pauses	•Very low, consistent GC pauses

- Oracle Java:

ParallelOld	ConcMarkSweep	G1GC
•General •Balances throughput and pause times •Occasional long pauses	•Deals with long pauses in ParallelOld •Often hard to tune	•Heaps > 10GB •Strategic future

Linux: CPU

- High CPU: Tasks waiting in line



- High CPU: Tasks waiting in line
  - Monitor:
    - 100 - Idle CPU % > ~80%
- Run queue > # Core Threads
  - Know your interval
  - Example: If a monitoring product interval is 15 minutes, spikes might be averaged out
- Monitor per-CPU utilization