**WebLogic Stuck Threads: Creating, Understanding and Dealing with them**

APRIL 10, 2012 BY [FRANK](http://www.munzandmore.com/author/frank) [19 COMMENTS](http://www.munzandmore.com/2012/ora/weblogic-stuck-threads-howto#comments)

Using the time off during the bank holidays over Easter I spent some time coding and looking into more unknown details of WebLogic stuck thread behavior. (Actually I started to write this posting because I was told by my doctor to keep my mouth shut for some days, but that’s another story…).

My personal task was to answer some of the most common questions I’ve encountered while consulting and running WebLogic 12c workshops. As often with my postings, this article is not meant to explain the basic concept of [thread pools](http://docs.oracle.com/cd/E24329_01/web.1211/e24432/self_tuned.htm#i1069235) or [workmanagers](http://docs.oracle.com/cd/E24329_01/web.1211/e24432/self_tuned.htm" \l "autoId1" \o "WLS 12c workmanagers). I recommend to read the Oracle WebLogic 12c documentation about [stuck thread handling](http://docs.oracle.com/cd/E24329_01/web.1211/e24432/self_tuned.htm#CNFGD121) first which explains how you can deal with stuck threads by configuring a workmanager.

Also there are some excellent details about [stuck threads (including WLST scripting and monitoring)](http://middlewaremagic.com/weblogic/?tag=stuck-thread) to be found at the [Middleware Magic site](http://middlewaremagic.com/) - a site run by a group of really knowledgeable guys.

Now, typically customers tell me that they *“observe some stuck threads”*, *“sometimes”*, but often they are *“not sure what caused them”* and typically they *“don’t know what exact state these thread are in”*and in addition nobody seems to know if *“the stuck threads ever clear up again without rebooting”*. I am a pragmatic guy. I enjoy having little applications or tools to demonstrate and measure how WebLogic is working. Keen to play around with the newest edition of [Netbeans](http://netbeans.org/downloads/" \o "Netbeans download)(I used to be an Eclipse guy) and EJB 3.1 in WLS12c I built a small application to easily test WebLogic stuck thread settings and countermeasures.

Here are some more details about the StuckThreadForFree application:

* The application allows you to create threads which are busy or which are waiting long enough to be detected as “stuck” by WebLogic.
* This little application will only work with WLS12c. I intentionally avoided JSF, so a plain JSP page is used to set your parameters. The JSP is calling a simple Servlet which in a for loop is calling an asynchronous business method of an injected  stateless session bean. @Asynchronous and no-interface session beans are only available in EJB3.1 so you have to run it on WLS12c. Unlike in previous versions, the EJB is directly packaged into the .war file for deployment.
* Every call to the stateless session bean is serialized by the EJB container, so every EJB method is executing in its own thread.
* Depending on which method was called on the EJB is either waiting n seconds using Thread.sleep() or calculating some trigonometric function for n seconds. Both methods will cause stuck threads.
* There is zero configuration in the deployment descriptors for the EJB! Only context-root for the web part is set (which could be avoided as well).
* Building the StuckThreadForFree app with Netbeans was a smooth ride and a real pleasure.
* The app is provided as is. You can have it for free, yet there is no guarantee for anything but it shouldn’t cause any problems either. Better don’t run it on your production system.
* It’s just a hack. It demonstrates what it should, nothing else.

***DOWNLOAD***: for your conveninience you can [download the StuckThreadForFree.war from here](http://www.munzandmore.com/downloads/StuckThreadForFree.war) and follow the example yourself (here is the link to the [whole Netbeans project](http://www.munzandmore.com/downloads/StuckThreadForFree.7z)). After downloading you can easily deploy it to WebLogic. To follow the example it’s good enough to run it the admin server. Then you can start with the following URL:

**http://localhost:7001/StuckThreadForFree**

[](http://www.munzandmore.com/wp-content/uploads/2012/04/st21.jpg)

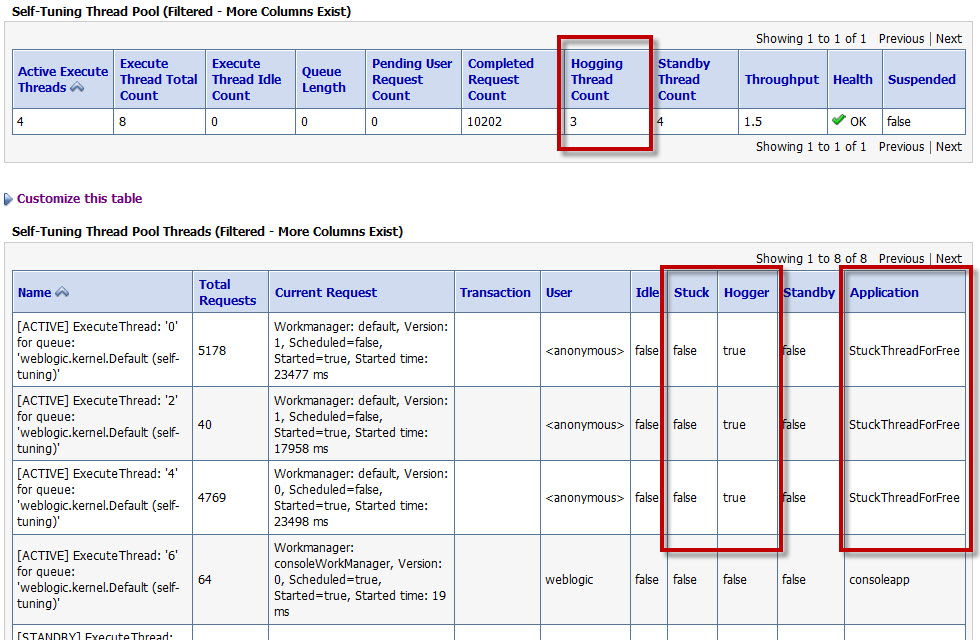
Now, lets use the app to answer some typical questions.

***What are hogging threads? When do threads become hogged? After what period of time?***

According to the Oracle doc hogging threads *“.. will either be declared as stuck after the configured timeout or will return to the pool before that. The self-tuning mechanism will backfill if necessary.”*

So how long does it take for them to become hogged? Nobody (including Google) seemed to know. Trust me I did some research and asked plenty of colleagues about this. Here is the answer:

If you run the application with 3 threads / 100 seconds / Thread.sleep() and immediately switch to the WebLogic 12c admin console Admin Server / Monitoring / Threads you will observe the following:

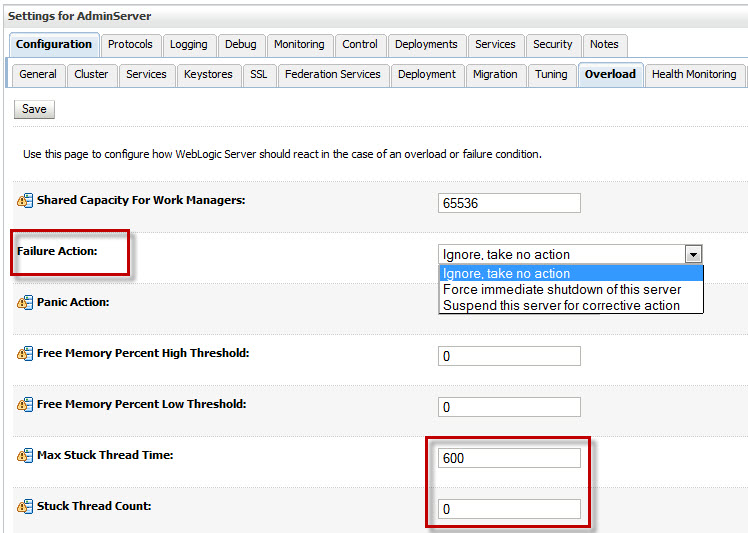
[](http://www.munzandmore.com/wp-content/uploads/2012/04/st3.jpg)

So interestingly hogging threads are detected *right away*! In my case it took about 2 seconds (I had to hit reload once).

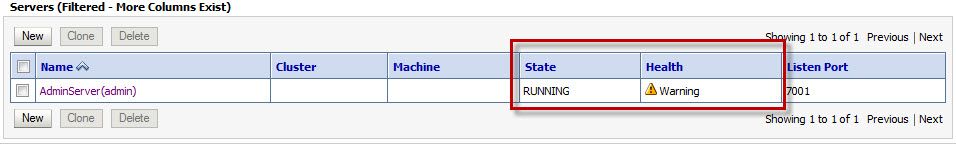
***So WebLogic transitions into FAILED state when a certain number of stuck threads are detected, right?***

That’s a common misconception! The default configuration of WLS 12c (I also checked for WLS 11 = 10.3.3) is Stuck Thread Count = 0, which means the server *“never transitions into FAILED server irrespective of the number of stuck threads”*. You will only see the FAILED state only when you set the value to a positive number of threads!

Once the server transitions into FAILED, you can define if WLS should be shut down (and restarted by WLS nodemanager) or suspended.

[](http://www.munzandmore.com/wp-content/uploads/2012/04/st4.jpg)

Remember: WLS will **not** transition into FAILED state when StuckThreadCount is set to zero. Only the health runtime value is set to Warning (but this will be cleared if the hogging thread conditions clears) as shown below:

[](http://www.munzandmore.com/wp-content/uploads/2012/04/st5.jpg)

***What exactly causes a stuck thread? What state does a thread have to be in to be marked as stuck?***

In general there is a number of different [thread states](http://docs.oracle.com/javase/1.5.0/docs/api/java/lang/Thread.State.html) in Java: [NEW](http://docs.oracle.com/javase/1.5.0/docs/api/java/lang/Thread.State.html#NEW), [RUNNABLE](http://docs.oracle.com/javase/1.5.0/docs/api/java/lang/Thread.State.html#RUNNABLE), [BLOCKED](http://docs.oracle.com/javase/1.5.0/docs/api/java/lang/Thread.State.html#BLOCKED), [WAITING](http://docs.oracle.com/javase/1.5.0/docs/api/java/lang/Thread.State.html#WAITING), [TIMED\_WAITING](http://docs.oracle.com/javase/1.5.0/docs/api/java/lang/Thread.State.html#TIMED_WAITING), [TERMINATED](http://docs.oracle.com/javase/1.5.0/docs/api/java/lang/Thread.State.html#TERMINATED).

But which state has a thread to be in to be marked as stuck later? If you run the StuckThreadForFree application and create a stack trace with WebLogic admin console under Server / ServerName / Monitoring / Threads you can observe that the thread state is **ACTIVE/TIMED\_WAITING** when using the Thread.sleep() method to block it:

"[**ACTIVE**] ExecuteThread: '5' for queue: 'weblogic.kernel.Default (self-tuning)'" **TIMED\_WAITING**

java.lang.Thread.sleep(Native Method)

com.munzandmore.stuckthread.LongRunningEJB.threadSleep(LongRunningEJB.java:26)

com.munzandmore.stuckthread.LongRunningEJB\_x9v26k\_NoIntfViewImpl.\_\_WL\_invoke(Unknown Source)

when using the calc() method to keep the threads busy they are state **ACTIVE/RUNNABLE**:

"[**ACTIVE**] ExecuteThread: '0' for queue: 'weblogic.kernel.Default (self-tuning)'" **RUNNABLE**

com.munzandmore.stuckthread.LongRunningEJB.threadCalc(LongRunningEJB.java:40)

com.munzandmore.stuckthread.LongRunningEJB\_x9v26k\_NoIntfViewImpl.\_\_WL\_invoke(Unknown Source)

weblogic.ejb.container.internal.SessionLocalMethodInvoker.invoke(SessionLocalMethodInvoker.java:31)

So both states can become stuck. Also, I am pretty sure I could also show the **BLOCKED** state when using a monitor lock for synchronization but due to time restrictions this is not included in the app.

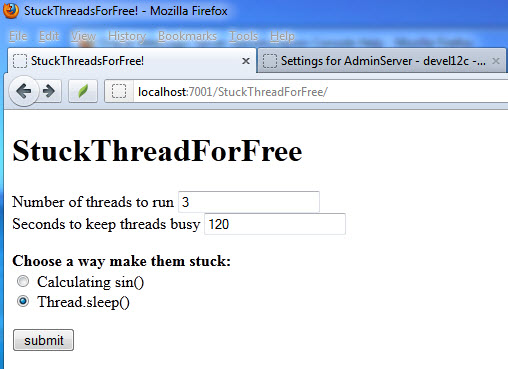
***Can a stuck thread still do reasonable work?***

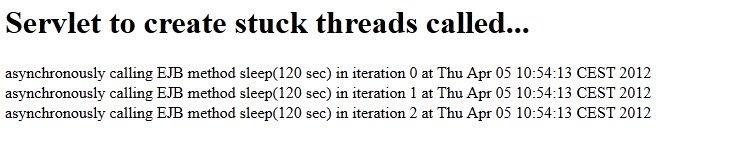
Absolutely! Just because a thread is marked as stuck it doesn’t mean it is frozen or unusable. Imagine you wanted to calculate PI, you are creating PDFs, distance maps, mapping the human genome or you have deployed some JCA adapter talking to MQ-Series, SAP or PeopleSoft which is internally using a Thread.sleep() method call. All of this is are reasonable usages likely to occur in the wild.

***Do stuck threads ever dissapear? Can they be cleared somehow? Are they stuck forever?***

First of all you cannot get rid of a stuck thread by simply “killing it”. You cannot cancel or kill any thread in Java. However, stuck threads automatically will disappear if the condition clears up which caused them to be marked as stuck (e.g. the sleep period is over or the calculation is done).

To prove the point, switch to the WebLogic admin console and under Server / ServerName / Configuration set**StuckThreadCount to 3** and **StuckThreadTime to 60 seconds** then restart the server and run the StuckThreadForFree app to create 3 threads running for 120 seconds using the Thread.sleep() method (the other method will work as well, there is no difference, but keeping 3 threads busy by doing math proves to be a fan test of your machine as well):

[](http://www.munzandmore.com/wp-content/uploads/2012/04/st7.jpg)

[](http://www.munzandmore.com/wp-content/uploads/2012/04/st6.jpg)

In the WebLogic log file you will find three entries logging the stuck thread state after a while:

<05.04.2012 10:55 Uhr MESZ> <Critical> <WebLogicServer> <BEA-000385> <**Server health failed. Reason: health of**

**critical service 'Thread Pool' failed**>

<05.04.2012 10:55 Uhr MESZ> <Error> <WebLogicServer> <BEA-000337> <[STUCK] ExecuteThread: '4' for queue: 'webl

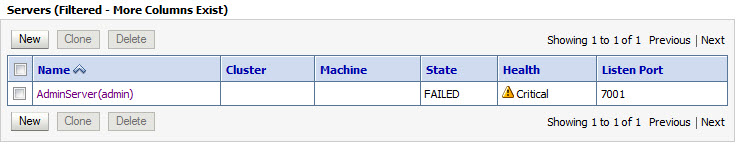
ogic.kernel.Default (self-tuning)' **has been busy for "85" seconds working** on the request "Workmanager: default

, Version: 1, Scheduled=false, Started=true, Started time: 85443 ms

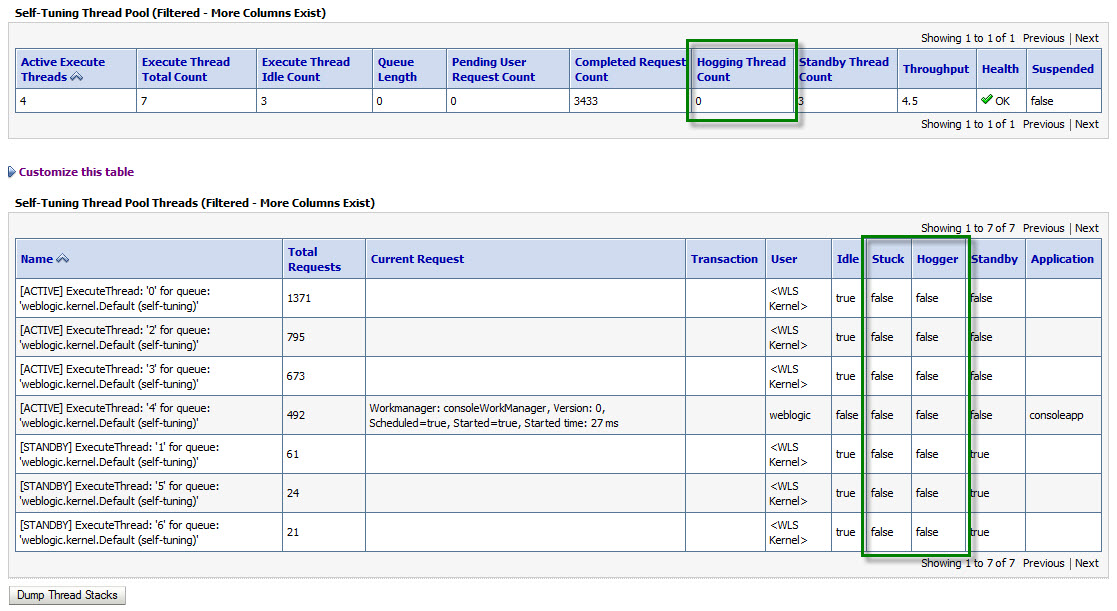
", **which is more than the configured time (StuckThreadMaxTime) of "60" seconds**. Stack trace:

java.lang.Thread.sleep(Native Method)

After waiting about one minute you will observe that WebLogic  is transitioning into FAILED state as configured:

[](http://www.munzandmore.com/wp-content/uploads/2012/04/st8.jpg)

Wait another minute, then check the thread states under Server / ServerName / Monitoring / Threads which reveals the following:

[](http://www.munzandmore.com/wp-content/uploads/2012/04/st9.jpg)

So once the condition causing the stuck threads is cleared also the stuck threads will disappear again! Stuck threads are not stuck forever. Phew!

***When should I use StuckThreadCount in the admin console or a Workmanager stuck-thread setting then?***

Very good question. Use StuckThreadCount from the WebLogic admin console or with a <work-manager-shutdown-trigger>[definition moving the application into ADMIN mode](http://docs.oracle.com/cd/E24329_01/web.1211/e24432/self_tuned.htm#i1071827) if you ***can react*** on the FAILED state.

Do **not** use StuckThreadCount if the threads might be doing something useful and you cannot react on the situation anyway. Obviously transitioning into FAILED state and restarting WLS with the nodemanager is counterproductive if you threads are doing something useful.