Troubleshooting Tips For ColdFusion MX

Posted At : November 23, 2004 7:02 PM | Posted By : Steven Erat
Related Categories: Java, Events, Quality Assurance, ColdFusion, Macromedia

I'm posting a CFTalk thread that I'm currently engaged in regarding general CFMX troubleshooting techniques, as well as my reply to Dave Watts' follow-up questions

---

ColdFusion configurations and environments *vary widely* among users. I don't think that casting a broad net for specious reports, incomplete data, and hyperbole will serve to solve whatever particular difficulty ails your server, although there is certainly much to be said for a controlled gathering of data "from the field" for careful analytic review [in order to quantify trends in the developer community].

Some things to consider [for a particular server problem] might be the following:

- I) Is this a server crash (jvm stops), a server hang (unresponsiveness with live jvm), or a certain frequency of errors with neither a crash nor a hang?
- II) If its a crash and the jvm has exited and perhaps restarted, then:
  - A) What is in the JRun log files *from the time of the crash*?
    1) java.lang.outOfMemoryError?
    2) is there a match for Sun bugs 5048441 are 5075468?
    3) how about GC logging or JRun metrics logging? What does that tell you?
  - B) Does a hs_err_pidnnnnnn.log exist (where nnnnn is a processid) anywhere under the JRun root or in the system's Temp directory?
    1) what's in the hs_err_pidnnnnnn.log? This is a HotSpot crash log and it typically contains information about what was being called at the time of the crash, and it typically contains a HotSpot Error Code at the bottom which can be used for Googling or searching Sun's Bug Parade.
    2) does the addition of -Xint to the jvm args list make a difference?
  - C) What are the settings in the jvm.config file?
    1) have there been any unusual settings added?
    2) have the settings been altered beyond the default?
    3) how suitable are the min and max heap sizes foryour application(s)?
  - D) Known issues (Techtote 18325). Are there any matches?
    1) be sure to use the 3.3 Macromedia drivers version.
- III) If the server's jvm seems to stay up but the server stops handling requests, then:
  - A) How are the principle server settings tuned?
    1) activeHandlerThreads (a.k.a. Simultaneous Requests)
    2) Timeout, the general one in the CF Admin
  - B) Have you logged slow pages as per the CFAdmin logging page and weeded out obvious bottlenecks?
    1) if so, then begin to drill down into the pages reported to isolate what logical parts of the code to account for the long run times.
    a) use gettickcount() method of time tracking
    b) use cflog, cftrace
    c) use regular CF debug output
  - C) Is a single server suitable to the amount of load and expected

throughput? IOW, if the server is tuned as best as possible, and the application code refined, then do you still need more?
1) have you load tested to satisfaction?
2) is there a need to scale up to JRun clusters across machines?
3) is there a need for additional hardware load balancing?

- D) Have you performed thread dumps from the time of a hang to observe what code is executing at that time? They are most helpful when taken 2 or 3 at a time about 15-30 seconds apart.
- E) Known issues (Technote 18325). Are there any matches?
- F) Check out DevNet Performance articles such as Jim Schley's "Performance Under the Covers in ColdFusion MX 6.1"

- IV) Unusually high frequency of errors without actually hanging or crashing the server. This is a fairly wide topic and specific to an installation, so this would likely be tedious and specific to a given installation or environment.
- V) Have you contacted Macromedia ColdFusion support directly?
Those tickets assessed to be verifiable bugs are closed at no charge (charges are reversed or support packs incremented)

Assuming that you know nothing about Java, how do you go about interpreting garbage collection or JRun metrics?

JRun metrics provide a simple view into the number of sessions and the amount of memory allocated and free, and you might want to use this before deciding to enable the more robust GC logging. In a bottleneck situation you might see a sudden rise in the number of sessions, and if the app has reported memory errors you might see the allocated memory grow or remain at the max heap size paired with a decreasing value for free memory, where heap allocated - heap free = heap used. The jvm typically has more heap allocated than used, within the confines of the max heap size.

GC logging output can be mildly cryptic at first glance, but with a brief review of some public explanations on the topic, one could find it very helpful to rule out (or in) certain possible suspects involved in server crashing or slow down. For example, heap size is printed out and with a little accounting it can be determined if the application is close to the max heap limit or not, and trends in memory usage can be estimated to determine if there are periods of memory usage where it is accumulated gradually and released or if it was rapidly acquired in a short burst. A graph of memory usage can paint a profile of the application's behavior and reveal areas or times of interest that might be correlated to specific application or user events. The logging of GC timestamps can be used with the GC times to determine if GC behavior itself is a bottleneck where, for example, rare circumstances on single CPU machines a lengthy GC could queue up requests since no jrpp threads can run during GC, IIRC. (I think on multiproc servers GC can occur on one cpu while threads run on the other, but I'm not sure right now). If that were the case, then there are GC switches to add to the JVM arg list that which modify GC behavior in a way that might benefit the app (thru trial & error).

Both JRun metrics and GC logging are of refined diagnostic utility, but by themselves they won't actually indicate what application code or behavior is the root cause.

For more see:

Pete Freitag's Blog, **Ferdia Blog**, **CFMX Under the Hood**, and **CFMX Performance Tips**.

> [Regarding HotSpot crash logs] I've seen this once, but the crash log didn't make any sense. It
> referenced in the JDBC-ODBC bridge that comes with JRun, but that wasn't being used by any of the CFMX applications on the machine - and that's all there is. I went through all the JRun server instances, but didn't see any using JDBC data sources. Do you have any insight on this?

The HotSpot crash logs usually provide two specific, useful pieces of information - the exact point of failure, sometimes too precise to intuitive, and the error code. I haven't seen any crash logs referencing the JDBC-ODBC bridge. Otherwise, the failure might be a library call to Native Code for example, such as when a crash occurs when calling a C++ CFX tag (including Verity via libCFXNeo.dll/so). The error code is best used when fishing on the Sun Bug Parade. This public facing bug base is often littered with recommendations and advice from Sun engineers, as well as collective input from the community. Sometimes you find actual fixes or workarounds.

> [Regarding setting heap sizes] How do you recommend determining this? I've read somewhere that it's a good idea to set both [max and min heap] to the same value; is that what you generally recommend?

Assuming a code review has been performed to maximize efficiency, leaving application design worries aside, one approach towards garnering the optimal heap settings might be to load test the application on hardware with abundant resources, while initially setting a very comfortable heap size. Once the test is underway a memory profile would be established and it might become obvious that the memory high water mark is nnn MB. You can then confidently run the application on similar hardware with a min heap setting roughly equal to the average heap size discovered during the load test and the max heap equal to the high water mark plus 10 or 20% to be safe. An additional consideration would be to make sure that there is sufficient memory on the machine to accommodate the application memory requirements (high water mark + buffer), such that the application uses no more than say ~70% total real memory on the server.

I believe there is one camp that recommends setting the min and max heap sizes

equivalently due to some degree of performance overhead when the JVM dynamically adjusts the heap size at runtime. There is another camp that recommends the min heap be set equal to the expected mean memory requirement and the jvm will have to do very little heap resizing other than momentary occassions where more is needed.

Continued ...