Hanging Jrun Threads and MS SQL Parallelism

Posted At : November 18, 2011 1:59 PM | Posted By : Mark Kruger
Related Categories: MS SQL Server, Coldfusion & Databases

Recently one of our systems started misbehaving. In this system we had 2 ColdFusion 8 servers connecting to a single MSSQL 2005 server. All the hardware was quite good - plenty of RAM, Fast disks, moderate traffic etc. The system had been in place for some time. But (and isn't this often the case) we moved a new design in place with some changes to the query code and suddenly our well-behaved system started acting like a sugar-laden toddler in the cereal aisle.

Watching the "running request" counter for ColdFusion I noticed that it was slowly accumulating requests. When that happens (threads slowing building up over time) you usually have to prepare for some frustrating troubleshooting ahead. When a server is "crashing" you can often pinpoint the error. Crashing servers tend to suddenly fill up running requests and the request queue and the log files will generally have some clues occurring right around that time. But this was different. In this case the request count climbed slowly and was seemingly random. And these threads did not show up in the list of "active requests" in the CF monitor either. Aha! I thought. This is my old networking issue! You might not remember this but a few years ago we discovered that auto-synching ports can sometimes cause phantom connections to hang on a DB intensive application (see **this post**).

But a quick checkup of network settings showed that this was not the case. Network connectivity was excellent and both DB and the 2 servers were connected through the same Cisco switch. So it was on to the database. Why the DB? Why not scour through JVM settings and fiddle with CF request settings? For one thing, 80% of the time it's not CF or the DB but some combination of the 2 (bad query writing, resource constrained DB, drivers etc). In this case the 2 common denominators were the database and the new code - but I believed the DB was our lowest hanging fruit.

Processor Usage

Sure enough a check of the database showed processor usage that did not look normal. Wait a minute Muse... don't you have any baseline numbers for that assumption? Nope, not at this point. I'm letting my experience guide me. When you have 4 cores and 2 of them are at a flat lined 50% you generally know something is wrong. In fact a quick check of the *accumulating* requests on CF showed a 20-25% per thread correlation. In other words, each of my hanging threads was using 20% or so of one core *on the SQL server*. Once it was hung that thread continued to use 20% of one SQL server core in perpetuity until CF was restarted.

The funny thing was that under regular load the DB processors was *extremely underutilized* until one of these threads was produced. The DB processors would stay at between 1 and 5 percent most of the time - practically idle. When one of these "special threads" came along, one proc out of the 4 of would "jump up" to 20 or 25 percent but the rest would idle along as before.

Finding the Problem

We tried a great many things. We patched and hot fixed, shrunk and optimized files, added and removed indexing etc - all of which was helpful and necessary, but none of which permanently "fixed" our problem). Finally, I was looking at the "activity monitor"

in MSSQL05. The activity monitor "process info" view shows a list of connections along with some extra data, process ID, database, status etc. If you double click on an item in the row you will see the currently running query or task. You have to sort of "get lucky" to see it since most of them fly by pretty fast.

In any case I was watching this view (refreshing every 10 seconds) while there were *no* hanging threads and suddenly I saw something that made me scratch my head. A process ID was duplicated about 3 or 4 times. Each of the duplications had a "wait time" and a "wait type" of CXPACKET. So this process ID was spawning multiple threads under a single ID. And the wait time made me think that this might be our offending process. Looking at the processor utilization I noticed that sure enough, I had a 20% utilization on one core. Going back to my CF servers my suspicions were confirmed. We had a hanging thread on one of the servers - so this CXPACKET thing required some more investigation.

First however, I thought I might try to mitigate the problem from within the activity monitor by killing off this process ID. If I was successful I would have a new mitigation technique that would *not* involve any potential user disruption, with the exception of whoever was running the query that was locking up these threads (and they were probably tapping their fingers on the desk waiting anyway). So I tried the "kill process" button from the activity monitor, but I had to kill all of them individually and I couldn't catch them all before they re-spawned - or maybe I'm just too old. Turning to SQL Studio I ran the query *KILL 55* (where 55 was the process ID in question). That did the trick and it was indeed a magic bullet. As soon as I "killed" that process ID - all the sub-processes were terminated as well. My CF server dropped the hanging thread and SQL Processor usage went back down to normal.

The Fix

Ok so now what? I could hire a temp to sit in front of the activity monitor all day and kill off any process ID with a CXPACKET Wait type that correlated to a CF hanging thread. I could *probably* write a complicated SQL script to find these threads and terminate them (I kind of liked that idea actually). In the end I chose to do a little research into CXPACKET wait types. I was fortunate to stumble onto **this post** by Pinal Dave. It turns out that a CXPACKET wait is related to parallelism. Now parallelism is how MS SQL chops up the work load of a query and makes full uses of your processors to get the work done. Much like cfthread splits work out and then joins it back together, SQL splits the work out and then an "organizing" thread "waits" for all the individual threads to complete. Once they have all completed it assembles the data for return to the client. Make sure and read the full article as well as the comment by Jonathan Kehayias at the bottom - excellent stuff!

In any case *my* SQL server was suffering from not being able to reassemble threads from this division of labor. I'm not sure why that might be (I have some ideas) but the long and short of it is that attempts at parallelism for query execution were causing hanging Jrun threads on my CF server. Following Pinal's guide (and a couple of MS resource pages) I tried setting the max degree to 2 and the threshold to 20, 25, 30... looking for a "sweet spot" where most of my queries would execute without parallelism, reserving it for the report or aggregation queries in the admin section of this site. Unfortunately that didn't work. The issue here was likely a specific query with some new joins in it that was *always* going to trigger parallelism and *ofen* fail to complete - causing our hanging thread issue.

Finally, I set the "max degree" to 1. Doing this meant that there would be a 1 to 1 relationship between threads, process IDs and queries. In other words, a given query would never use more than one core execution thread. Now you might think this is problematic because it doesn't make full use of SQL's tuning engine. Technically you are right. I would only say that in a typical web application the query traffic generally consists of dozens of *very short* queries where parallelism would actually add additional time to the process. So in a *typical* web application you lose very little by minimizing the degree of parallelism. And indeed that appeared to be the case in our web application. Our CXPACKET waits, hanging threads and egregious processor usage all went away and things have been functioning smoothly since then.

The Aftermath

The Muse knows his readers well. Some of you want to hammer me about not fixing the real problem - that specific query in the code. Not to worry. Using SQL's performance dashboard we teased out the worst offenders and set our ColdFusion developers to analyzing the code. But I suspect the version of SQL or something about the hardware, hyperthreading or NUMA to be a more likely culprit. I have never seen SQL's execution planner cause a problem when it turns to parallel execution before. Still - it's always a good idea to fine tune that query code.