Troubleshooting and Optimizing Solr on ColdFusion 9

Posted At : April 4, 2010 8:53 PM | Posted By : Mark Kruger
Related Categories: Coldfusion Troubleshooting

I had an interesting troubleshooting session recently with a customer. This customer had a very high powered server - SAS drives, 16 gigs of RAM, 64 bit throughout, Coldfusion 9 and an 8 gig Java Heap. The site had 70 or 80 search collections and they had switched to CF9 specifically to get beyond the limitations of Verity. Everything was performing well when *suddenly* the search service stopped responding to requests and simply started throwing "collection not found" errors. Coldfusion seemed fine and dandy. It continued to be responsive and it had no hanging threads. It was as if the search service had lost it's handles to the various collections. Restarting Solr solved the issue, but why was Solr locking up?

A closer look showed that Solr wasn't crashing. Indeed, both of the services associated with Solr ("ColdFusion 9 Solr Service" and ColdFusion 9 Search Server") were running. In fact, on reflection it was clear that the Solr service was actually still responding to requests - it was simply throwing errors. So Solr actually *had* to be running, or we would see the "service not found" type error - right? Now the Muse is not yet a Solr guru, so this was trial and error. But the first thought was to go and see if there were any Solr log files. Sure enough I found them under /Coldfusion9/solr/logs. There were 3 logs listed - a "request" log, a "stderr" log and a "stdout" log, each with a date as part of the name. Hmmmm..... 2 of those logs look sort of familiar don't they? The "standard out" log or stdout (never been comfortable with the abbreviation "std") and the "standard error" log or stderr both remind me of conventional java logging.

Of course! Solr is a Java application under the hood. So troubleshooting Solr will be like troubleshooting any other Java application. I took a look at stderr and sure enough - recent errors were all "out of memory" Java heap errors. I know how to handle that - increase the heap size and fiddle with the Java arguments. I was well on my way to editing the JVM.config file when I had a different thought. Solr was running out of heap space but *ColdFusion was **not** running out of heap space*. After all, if I was having an overall memory problem I would expect ColdFusion to lock up as well. That means Solr runs in a separate JVM. Of course it does (doh!). That's why it has a service and a connection socket. What I really need is to find the jvm.config file (or whatever) for Solr and adjust the memory settings there.

In the root of /ColdFusion9/solr I found a little file called "solr.lax". You might remember a similar .lax file that I edited to get a 64bit install to complete on a windows 2k3 "web edition" (details **here**). It's actually a config file for an "executable jar" file compiled by a product called "launchAnwhere" - which I gather is basically "installsheild" for Java. Anyway, this file is bundled with an executable jar and contains the useful stuff the exe file will need to instantiate a JVM and get up and running.

In this case there were 2 lines we were interested in. The first one had a setting like this:

```
#    LAX.NL.CURRENT.VM

#    ----------------
```

```
#   the VM to use for the next launch

lax.nl.current.vm=C:\\ColdFusion9\\runtime\\jre\\bin\\javaw.exe
```

This line tells us that the Solr search service is going to use the same JVM as the ColdFusion engine. That's pretty good news because we already know a good deal about preventing and troubleshooting out of memory errors in Sun JRE version 1.6 (the default shipping with CF 9).

The other line of interest was further down and looked like this:

```
lax.nl.java.option.additional=-server -Xmx256m

  -XX:+AggressiveOpts -XX:+ScavengeBeforeFullGC

  -XX:-UseParallelGC -DSTOP.PORT=8079

  -DSTOP.KEY=cfstop -Dsolr.solr.home=multicore
```

Now this looks mighty familiar. It's our Java arguments to instantiate our JVM and configure it for use. What can we deduce from this set of arguments? Well, to start with, Solr is configured to use a minimum of zero and a maximum of 256 megs of memory. It's also set up to aggressively recover heap space. In our special situation we need to get Solr more memory and more resources to work with. We changed the GC to -XX:+UseConcMarkSweepGC, removed the aggressive options, and set the minimum and maximum both to one Gig (-Xms1024m -Xmx1024m). Sure enough after a restart our memory problems went away and Solr started performing splendidly.

Lessons Learned

Solr is a big step up from the old Verity services, but in the words of Uncle Ben (Peter Parker's uncle - not the rice guy) "With great power comes great responsibility". Now that we have something we can scale for our use we have one more thing we have to carefully tune and account for in our resource planning.