Coldfusion, MInisoft ODBC & the HP e3000

Posted At : January 25, 2005 11:46 AM | Posted By : Mark Kruger
Related Categories: HP e30000, Coldfusion & Databases

We were recently tasked with troubleshooting a sick system that consisted of a CF 4.5 server connected to an HPe3000. The HPe3000 is a legacy system (as is CF 4.5) with data stored in a native image file system. Data is keyed, queued and cached for retrieval according to a schema. In the case of CF 4.5 we were using an ODBC driver that made the data look like more typical tables.

This entry describes the problem and our efforts. I would like to acknowledge Michael Gueterman of Easy Does It Technologies for providing many useful ideas and directions to help our effort. He sent me at least 3 lengthy emails with extra insight into the HPe3000 and answers to my questions. We made a great deal of progress based on his advice. If you have an interest in troubleshooting or CF with HP, read on..

p>The system ran fine for some time (years in fact), but from the beginning it suffered from a stability problem. The CF server would occasionally restart. While this was certainly not ideal for this company, it was something they could manage and it was infrequent enough to not cause any serious customer service issues. They added some extra monitoring and put in place a reasonable contingency plan for the outages - which averaged about 1 every 7 to 10 days. At some point, however, an increase in traffic or utilization brought on a sudden "threshold" crash of the system. The CF server began to restart seemingly at random. At one point the CF server would restart every few minutes - making the entire system unusable. That is the point where we became involved.

How common is it for a system with a relative stable baseline to "suddenly" begin to act randomly unstable without any *significant* changes to the code or environment? Actually it is not as uncommon as you might think. A system that is written to be scalable is not necessarily designed to give the same performance level for a 30% increase in traffic. Instead, it is designed to be *predictable*. It should to ***degrade gracefully*** as load increases. This makes capacity planning a matter of charting the rate of increase and the rate of degredation and planning accordingly. A system with no scalabilty will ***degrade suddenly*** instead. This often prompts IT managers to scratch their heads and mumble, "...Something must have changed". Otherwise, why would a 4% increase in traffic from last week result in a sudden unstable system?

Why the Sudden Instability?

I could wax eloquent about load testing and the important of having baseline numbers at your fingertips for comparison. Still, I can hear my client saying "...but it ran fine for years!" What makes code or a system lack "scalability". What creates the condition in a system that results in usability one day and complete failure the next - with only a minor change in the environment - in this case slightly more traffic and slightly more data. At least one culprit can be the ease of use of the CFML language.

Eureka! It works!!

Coldfusion allows relatively inexperienced programs to write complex code that actually works. Many CF programmers come to the table with nothing but HTML skills.

Because it is "tag based" the learning curve is lessened and they are able to add behaviors to "pages". In fact, that is how they think of it - adding behaviors or actions to "pages". This "page" approach allows them to focus narrowly on the *beginning* to *endness* of the page. They see the entire script as a "document" with "interactivity". This useful but limited approach allows them to iterate through the page and continually add new behaviors and interactive features. Because CFML is so accessible - the code they write works. It's one of CFML's great strengths, and one of its great weaknesses - or in the words of Adrian Monk, "..it's a gift...and a curse".

Like all procedural languages it also allows code to be written with a linear task based approach. A developer can decide what he wants a template to do and begin writing and testing all at once with little plan. As the template gains in complexity, new behaviors are added to it in a linear fashion - that is they are *plugged in* to the time line of the http request. This results in code templates that can be thousands of lines long with queries, logic, HTML, Javascript, messaging etc - all assembled on the page based on the order in which the task was considered in the programmers head. This results in code that is not maintainable and hard to debug. When a new programmer - or even the same programmer - has to modify the script he has trouble even finding the area in question.

That was certainly the case here, and it brings to light still another problem. With code so obviously in need of repair, where is the problem. We found ourselves in a spot where we had ***too many*** possible solutions. There are many areas of the code that we *could* go about *fixing* right now. Would that solve our problem? Well, it couldn't hurt, but that is not what we did.

Identifying the first problem

In this case, we didn't *start* by examining the code. First, it was important to rule out environmental issues. We verified that the code base had not been changed. Working with the HP team we verified that the schema and structure of the data had not been changed. Working with networking we verified that the issue was not related to a lack of resources. We examined the hardware, application logs, CF logs etc. - before we ever got to the code. From all of that leg-work we got *only one clue*. Traffic had increased by 20 to 30 percent and there were 15% new customers on the system as the result of a merger.

In examining the CF log we saw that immediately prior to the crash the sessions began to climb and the average DB connections would double, triple and quadruple. The test data did not behave this way - but the live data *did* behave this way. We contacted the driver vendor (minisoft) about the ODBC driver and tried to get some helpful feed-back from them. They were unable to offer any help, other than to try testing queries through an ODBC test interface *other* than CF. That made some sense, so we used Access and the MS ODBC tester to run some of the more complicated queries against the live data. That's when we discovered something.

sqlPrepare( ) limitation

Like all ODBC drivers this one passed a prepared and validated (for syntax) statement to the server (or in this case an ODBC call *listener* dameon running on the HP). It receives the statement as a string, *prepares* it and then sends it to the listener for execution. When the size of the string we passed to the driver exceeded some barrier - which we determined to be 19929 characters including spaces, tabs and linefeeds - The

connection will hang. The listener is seen as "*waiting for read request from driver*" and the driver is in a wait state as well, waiting for a call-back from the prepare() function.

Because of programming decisions at the time the site was built - and because of some limitations to the HP schema - an increase in customer data had caused some of these query statement sizes to exceed that physical limitation. Whenever the statement length was exceeded a hanging thread was created. These threads exceeded thresholds on the driver, the CF server and the listener causing instability and random restarts.

When we contacted Minisoft, they would not acknowledge that there was a limitation to the statement length at all. In fact they told us pointedly that the driver accepts a pointer to a character array and any limitation must come from Coldfusion. We continued to patiently ask pointed questions like, "Why do I get the same trace reaction from the Access client or MS ODBC testor?" We sent trace files from different configurations to them for examination. Finally, a developer discovered that the statement size was overrunning the trace buffer. This is a buffer (apparently) that holds a string to send to output (file or standard IO) so that developers can view the statement. The driver *always* populates the trace buffer in anticipation of it going somewhere - and discards it if tracing is not enabled. While the statement size for the listener had no limitation, the trace buffer was working as a governor over the process - since it could not proceed if the statement exceeded a certain length.

As a work around we broke up the larger queries into smaller queries. and rejoined them in script. This solution, added to code that was already poorly optimized was hardly a panacea. We also aliased columns in the queries (SELECT U.firstname from user U - instead of SELECT user.firstname FROM user etc.). This worked well for query statements that were marginally close to the limitation. After these efforts the system would stay up for periods of time with performance degrading noticeably - then it would crash again. We got a new driver from Minisoft and rushed it into production. This helped somewhat - but the site continues to suffer from "code based" scalability problems and we are in the process of rewriting to a CFMX server as we speak.

What we learned

We learned a great deal from this troubleshooting effort.

- Don't push the minisoft driver too hard. It may go without saying for some folks, but a driver that is mimicking the windows environment is probably not going to be as capable as native access drivers.
- Pay attention to traffic and data issues.
- Know your data - The fact that customer records are what caused the increased query size highlights the fact that there was a disconnect between the HP team and the CF team.
- Provide evidence to 3rd party vendors. Many such vendors are only inclined to work with you if you can demonstrate the issue belongs to them. They need a burden of proof to take ownership of the problem. This only makes sense to folks who's business model lacks in the human equation, but it is sadly common.