Tracking Down a Performance Problem

Posted At : September 23, 2014 3:15 PM | Posted By : Steve
Related Categories: Brownfield Development

The Problem

The answer always seems obvious once you find it. One of our clients had one of their server start going down nearly every day. It quickly escalated to happening about twice a day.

The Process

We looked at recent commits for anything that could have caused a problem and found nothing. We verified that the code on the server that went down matched the code on the other servers in the cluster and that all of the server settings were the same. When we took down the offending server, another server started going down instead. At that point, we knew we were likely facing a code problem.

The site in question is one we inherited from the previous vendor, so we had no instinctive sense about where the problem might be. Fortunately, we have a good process for tracking down these sorts of problems.

We quickly got **SeeFusion** installed and running on the servers and made sure we had database logging set up. I love SeeFusion generally, but for me the database logging feature is where the rubber meets the road. I can write queries to see trends and patterns over time. One report in particular seemed a likely culprit for the problems. It frequently ran long and was often called just before the trouble started.

That file had a little over 300 lines of code, including some custom tag calls. Nothing jumped out as an obvious cause of trouble, but a few places looked like possible culprits. We didn't want to go in and make a bunch of code changes without knowing that we were addressing the problem. So we used a custom version of cf_timer.

The cf_timer tag basically keeps track of how long it takes for a block of code to execute. Our custom version has the ability to track this data in the database along with any custom data that we want to pass in (so we might keep track of variables that might impact how long a block of code might take to execute). The tag itself is very fast so that it doesn't cause a major impact on performance.

We put cf_timer tags around every major area of the page that we thought might possibly cause a performance issue. This included areas setting variables, queries, custom tag calls, even the output section.

We let this run for a few days, during which the page was run a few thousand times (lesson learned: I'll check quicker next time). Looking at the averages for each section was informative, but not alarming. However, sorting all of our time entries by the longest times was alarming. The five slowest times were the table display. After the five slowest times, all of the rest of the time entries ran in 5 seconds or less. The five slowest times, however ranged from 20 seconds to 8 minutes! That was just to display the data. So, it was taking 5 seconds or less to get the data and then up to 8 minutes to display it. Something was wrong with the display code.

The report potentially had a lot of data, so taking a few seconds to display it all was acceptable, but ballooning up to 20 seconds or exploding up to 8 minutes to display the data seemed to indicate a problem. Worse than that, we knew that this wasn't the upper limit as the page sometimes failed to complete and instead brought down an entire server.

The Culprit

Once we knew where to look, however, finding the problem was easy. Here were two separate pieces of code that make up the entire problem.

Each row of the output was surrounded by this conditional.

```
<cfif NOT ListFind(UsersWritten, UserList.user_id)>
```

At the bottom of the code within that conditional was this:

```
<cfset UsersWritten = ListAppend(UsersWritten, UserList.user_id)>
```

"UserList" is the name of the query for the report. What the developer wanted to do was to make sure that the report showed each user only once. Instead of doing this in the database, it was done in the ColdFusion code. That meant that the query was retrieving potentially much more data than it needed to retrieve and the ColdFusion code was doing potentially drastically more work than it needed to.

Let's look at what is happening at around the 5,001st row. At that point, the "UsersWritten" list holds 5,000 items. Therefore the usually cheap ListFind() operation has suddenly become very expensive as ColdFusion has to compare the user_id to each of 5,000 different items in the list. This is true for that row and every other row after (except that each gets potentially more expensive that the previous). So, the amount of work needed doesn't grow linearly with increased data. Which is why the time eventually explodes with the potential to bring the entire server down.

The Fix

We ditched the list code and instead altered the query to ensure that it would only return one record per user. This made the query a little bit faster and the display code vastly faster.

The Results

In the month since we adjusted the code, the display code has topped 10 seconds several times, but only hit 15 seconds once and has never exceeded that. In all cases these slow times where when it was displaying over 50,000 records on the report.

Before the change the display code averaged 526 ms and maxed out at 519 seconds. Since, it has averaged 82 ms and maxed out at 15 seconds. The query went from an average time of 114 ms to 102 ms. The max time on the query did go up from 7 seconds to 16, but it only exceeded 6 seconds two times in the 74,000 times it has run since the change whereas it had done so three times in 7,000 times it ran while we were testing.

The server quit going down.

The Lesson

When facing a major performance problem it can be useful to attack low-hanging fruit with no testing if you know what it is. Sometimes a performance problem hits right after you make a code change and you know just what the problem is. In that case, there is no point in wasting time in collecting a bunch of data.

Other times, however, it isn't obvious what the problem is. In our case, the problematic file hadn't been changed in over a year (and we didn't write it). It was just reporting steadily more data until it became a problem.

In those cases, it is good to have a plan to find the problem. In our case, we used SeeFusion to point us to the problematic page and then our own custom cf_timer to find the problematic code on the page and to verify our results when we changed the page. I should note that we tested the changes in timer

results locally before we deployed them.

Another lesson, of course, is that it is usually a good idea to let the database do as much of the data work as you can.