Session Damage

Posted At : March 11, 2007 11:16 AM

Related Categories: ColdFusion, .NET, Database, Site Performance, IT in General, Java

I have written about the architectural flaws I see in various applications, many focusing on poor database design and poorly written sql. I felt it necessary to write about an issue I have seen in various web applications that unbeknownst to the developer/architect can hinder and or ultimately spell disaster for the application and the customer using the application.

The title "Session Damage" came about from this very issue. When does storing information in session scope in CF, ASP.NET, JSP, etc. become a problem? I've seen a couple of scenarios that were poor approaches to utilizing session. My first experience with this was (not having seen the actual code) when monitoring the JRUN service on a UNIX (Solaris) machine I witnessed 4 megabyte of memory peeling off the server upon every new login. Initially my mind went to the idea that there was a memory leak somewhere due to the memory bloat. On the UNIX platform this caused the operating system to dump core and restart the service when memory exceeded set thresholds. This lead to customers losing shopping cart/session data left and right. YIKES!!!!

There was one scenario where a customer had purchased $4000 worth of goods and actually took the time to call the support team and have them purchase the items because he did not have the time or patience to spend another 30 minutes selecting the items all over again... The culprit, once I got a chance to look at the code was that the application was written in such a way that upon a successful login, the system cached much of the database for each user (much of it was actually never utilized). This resulted in the memory bloat. Of course this was all done in an effort to speed performance, but regardless it was a poor approach. I had difficulty explaining what was going on to the CIO because he was unable to grasp the concept and kept saying "Memory is cheap, just buy more memory". Ouch, I had to explain a server has a maximum capacity for memory and that this would not fix the problem, just mask it for awhile.

So, the system had to be reengineered due to the memory issue and the queries streamlined to speed querying of the session data. The lesson learned here is that it is a poor approach to cache data at session to save .01 seconds of round trip time to get it from the database. An architect or developer must weigh the cost/benefit to the system when looking at this challenge. I always recommend tweaking the db so that the query search yields a timely response.

Another scenario I witnessed recently was the use of session caching associated with search results. These were very large datasets getting cached at the user level. It caused the JRUN service to bloat to 600 megabytes in no time at all if there were only a dozen or so active sessions on the server. There are times when caching search info is pertinent, but rather than caching the entire result set it might be a better approach to cache the search results unique identifiers only (array or comma-delim list) and go back to the database to pull back the details when needed. The reason the system developers built the system in this way was to facilitate pagination. The solution was to stop caching and go "round trip" to the db for this process, the performance impact was slight (.02) milliseconds difference, but it is of my opinion that even if there was a 1 to 2 second difference the user would not find issue with the search, considering the big picture of server stabilization and a more an application that no longer required a restart during peak usage do to unresponsiveness. Isn't that what we all look for in an application? One that is written once and never requires intervention? 8-)