Robert L. Bogue
MS MVP, MCSE, MCSA:Security
317-844-5310
Rob.Bogue@ThorProjects.com

Performance Improvement: Understanding

One of my least favorite discussions in development is the discussion about performance. It's one of my least favorite because it requires a ton of knowledge about how systems work, and either a ton of guesswork or some very detailed work with load testing. I generally say that the results you get out of any performance prediction exercise are bound to be wrong. The goal is to make them as least wrong as possible.

I'm going to try to lay out some general guidelines for performance improvement through improving understanding about what performance is, how to measure it, and finally solutions to common problems. This article will cover the core understanding of the performance conversation. The second and third articles will cover session management and caching because they have such a great impact on performance — and on what solutions you can use to improve performance. The final article in the series will focus specifically on ways to improve performance.

No series of article (or book for that matter) could cover every possible situation that you can get into with computer system performance. My background includes nearly 20 years of work with systems from a VAX running VMS to more current projects which are based on Microsoft products such as .NET, Microsoft SQL Server, and Microsoft SharePoint. The concepts in this article are applicable to any complex system; however, I use the Microsoft platform including Windows, .NET, and SQL Server.

Performance and Scalability are often discussed together because they're linked. As you add more users (scalability) you reduce performance. While the conversation here is titled performance because the focus is around maintaining the performance of individual requests, thus scalability techniques are often discussed to reduce the amount of load on a server (number of users) to a point where performance is acceptable.

Of course, you can make decisions that will be good for performance but potentially bad for scalability—particularly around caching—however, for the most part when you improve performance you improve scalability and vice versa.

The reason for this approach is that performance is measurable. Scalability isn't measurable—or at least isn't easy to measure. As I'll discuss in the load testing section, without historic data, scalability assessments turn into educated guesses.

H2BACKGROUND

Before the specifics of performance can be talked about, a few of the infrastructure things that can impact performance (and reliability) have to be discussed. As I get into talking about session state management I'll have to talk about how reliability factors into some trade offs between performance and resiliency. The first stop on the journey to understand how infrastructure impacts these discussions is network load balancing.

Network Load Balancing

Any solution of significant size will include a network load balancer. Whether that's the free Network Load Balancer (NLB) that is a part of Windows or whether it's dedicated hardware from a provider like F5 BigIP, multiple servers means balancing the load between them. It should be said that clustering, which is a fault tolerant strategy for ensuring availability is often confused with load balancing. As a general statement you load balance your application layers including web servers and set of back-end application servers and you cluster the databases. Load balancing improves reliability and scalability and clustering, in the Microsoft definition, only improves reliability.

Related Articles

- Hardware's Dirty Little Secret, or Why Software Can be Mass Produced
- Android XML Parser Performance
- Getting Along Within a Team or on a Project

In general, network load balancers have two basic modes. The first mode is session independent. In other words, a decision is made at the time of a request based solely on the perceived load to the servers. The server with the least perceived load gets the request.

I say perceived because the load balancer doesn't **really** know which server is the least busy, it can just take a guess by historic CPU time, responsiveness to requests, etc. It basically makes a guess at the right answer. Most of the time the guess is probably right but sometimes it's wrong. On the whole it works out, and generally in this mode the requests are serviced with a relatively even response time.

However, the goal of performance planning isn't generally to service all requests equally. That's like saying we want to serve everyone equally badly. In other words, if in order to serve people equally you have to serer them all poorly, then perhaps serving some folks better than others is acceptable. (You may remember "Harrison Bergeron", Kurt Vonnegut's story about a world where everyone was equal.) Generally speaking, the goal of any performance exercise is to service all requests with the best overall performance. This presumes that there's not an overriding factor like transaction resiliency that take precedence. So in performance tuning, the total seconds to respond should be lower, even if occasionally a request or two takes a bit longer. That's where the idea of sticky sessions—or targeted sessions —fits in.

In a sticky sessions mode, also called pinning or affinity, the load balancer ensures that a user who starts on one web server stays on that web server for their entire browsing experience except in the case of a failure of the web server. This is done a variety of different ways including by the SSL session key—which is best, or as simply as the incoming IP address. So why would anyone want to do this. It would seem on the surface that distributing to the least busy server is the best answer.

The reason this isn't always the best answer is because the server that last serviced a user can cache a lot of information about that user and the content they're interested in. If Joe is looking at Product 123, then both Joe's profile and the product information for product 123 are likely going to be cached at some level on the server that Joe last used. This

means that the next request can best —as in with less resources—be serviced by the last server Joe was on—not a different server in the farm. This reduces the workload for servicing Joe's request and in aggregate across thousands of requests reduces the overall load of the servers and the supporting infrastructure.

If you can accept that you want sticky sessions, despite the natural inclination that you want to spread requests equally between a set of servers, then you have a foundation to be able to make some intelligent choices about what has to be persisted about a session and where it needs to be persisted to be protected. You have the ability to decide whether to persist session data in memory on the server a user is on, store the session information in a database, or transmit it back and forth as a part of the page. Deciding that you want sticky sessions can also impact your caching strategies because you may be less concerned with managing distributed cache issues if users will be on the same server for all of their requests.