

Re: W2K3 IIS 6.0 ASP.NET Requests Per Second Limits?

Source:

<http://www.tech-archive.net/Archive/Internet-Server/microsoft.public.inetserver.iis/2005-01/1557.html>

From: David Wang [Msft] (*someone_at_online.microsoft.com*)

Date: 01/21/05

Date: Fri, 21 Jan 2005 03:21:41 -0800

There are documents on MSDN describing the formula used to calculate ASP.Net threads in relation to CPU. I can't remember the link at the moment.

Re: question about async call

If you poll for the async return, then that is not really asynchronous. Remember, you can always fake a synchronous operation by making an asynchronous operation and then waiting for its completion function. The point of asynchronous programming on the web is to schedule some work item to be done LATER and immediately freeing up the original thread. That queued up work item can take its time, use other non-IO threads to do work, and eventually return a response. If you wait for the work item on the original thread, then it really wasn't asynchronous.

Given your example of a web page making a web service call to retrieve data and display it, here is what synchronous and asynchronous versions would look like:

Synchronous:

- IO Thread starts executing the Web Page
- Web Page makes HTTP request (sync or async) to web service
- Web Page waits for HTTP response to return (either with a callback or not)
- Web Page displays the content of the HTTP response
- Web Page finishes, returns "success", and releases IO thread to do other work

Asynchronous:

- IO Thread starts executing the Web Page
- Web Page makes async HTTP request to web service and informs it to invoke a delegate as callback
- Web Page immediately returns "pending" and releases IO thread to do other work (response is sent on completion callback...)

At some point, HTTP request completes and triggers async completion of your callback delegate

microsoft.public.inetserver.iis: Re: W2K3 IIS 6.0 ASP.NET Requests Per Second Limits?

Some thread invokes your delegate callback function
Callback function displays the content of the HTTP response
Callback function finishes, returns "success", and releases this thread

I realize that asynchronous operation is quite subtle as a concept and takes some time getting used to it.

The key thing to note is that in synchronous, the IO thread is waiting for the HTTP response to return and hence cannot handle any other incoming requests (i.e. your web page is latent). In asynchronous, the IO thread queue's up the HTTP request and immediately returns back and can handle other incoming requests (i.e. low latency). Whenever the callback happens to occur, the request processing completes on some other thread designed to be used to send the response.

Since ASP.Net depends on the worker threads to "do work", if you have high latency operations tying up the worker threads, it directly affects ASP.Net's ability to handle incoming request load. The two ways to combat this situation are to either decrease request latency by using asynchronous operations, or increase number of worker threads at the cost of increased concurrency costs (like context switching). The best solution is to decrease latency but can be hard to implement; the easy solution is to just bump worker thread count up. Just realize that the easy solution has limits and impacts the system in other ways.

```
--  
//David  
IIS  
http://blogs.msdn.com/David.Wang  
This posting is provided "AS IS" with no warranties, and confers no rights.  
//  
<roberto3214@netscape.net> wrote in message  
news:1106231032.332459.122470@f14g2000cwb.googlegroups.com...  
Hi David,  
Thanks for your comments I have since done further tests before reading  
your post mainly with increasing the number of worker processors web  
garden setting in iis 6.0 there seems to be a direct relationship  
between increasing this number and number of requests per second  
increase. As you have stated response times for pages may degrade.  
But I do have a couple of followup questions. In the above example of  
1 sec delay starving the asp.net of its worker process, I understand  
that an async call to do such a task will not hog up the worker process  
but wouldn't u still have to poll or wait for the result to come back?  
Say you wanted to get the current temperature using a webservice and it  
took say 1 second to get a response even though I use an async method  
wouldn't i have to poll on my webpage to get a result back? And if so  
does that mean I'lll be starving the asp.net of worker process?  
Secondly you mentioned there is a relationship between CPU and ASP.Net  
IO worker threads, what is the relationship? IF I have 1 cpu on average  
how many asp.net io worker threads do I have?  
Thanks for your time and effort  
Regards DotnetShadow
```