

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

Source:

<http://www.tech-archive.net/Archive/Internet-Server/microsoft.public.inetserver.iis/2005-01/1657.html>

roberto3214_at_netscape.net

Date: 01/22/05

Date: 22 Jan 2005 14:41:51 -0800

Thanks David for all your help much appreciate it. Now I understand what your saying :-)

Regards DotnetShadow

David Wang [Msft] wrote:

- > *I highly recommend asking future ASP.Net questions on:*
- > *microsoft.public.dotnet.framework.aspnet or Forums at*
- > <http://www.asp.net> --
- > *there will be lots of experts who can instantiate this in ASP.Net code.*
- > *But, I'm mostly interested in explaining the underlying concepts because*
- > *once you get it, you can apply it to lots of places with no limitations of*
- > *using ASP.Net.*
- >
- >> *The way I understand async programming is if u need to do other*
- >> *kinds of long process work that aren't needed straight away to*
- >> *continue processing the job. Say if u had to log something to the*
- >> *database. But in the above example its clear that if you fire a*
- >> *request to webservice nothing more can be done until the result*
- >> *is back.*
- >
- > *Correct, but nothing sais that the main thread needs to be the one*
- > *waiting*
- > *for the result. The point of having an asynchronous ASP.Net web page*
- > *is to:*
- > *1. allow the page to tell ASP.Net "don't send a response yet until I*
- > *tell*
- > *you" and then immediately return the IO thread to ASP.Net and*
- > *essentially*
- > *defer execution of itself. This is very important.*
- > *2. obviously, the page has to have made some async function call*
- > *prior to*

> returning the IO thread (else, this request is "leaked" and will never finish), which will call the page's specified callback function when it completes.

> 3. It is when the callback executes with both the async function call results AND ASP.Net context that the page can resume and tell ASP.Net "alright, this is the response you should send"

>

> ASP.Net exposes specific asynchronous primitives and pagetypes for you to use to handle the chores. In other words, async pages can be easily made into sync pages, but not vice versa -- because async pages do not assume a common thread executes all code (so it can conveniently "wait" on the same thread to make things synchronous). Sync pages assume a command thread execution so it must be re-written to become asynchronous.

>

>

> Let me give another example using pseudo code for conceptual clarity (I'm not going to tie the idea to any particular implementation, such as ASP.Net, since that is only syntax):

>

> *SendResponseUsingFunction1AndFunction2Data()* returns to ASP.Net "I'm done"

> *SyncFunction1()* and *SyncFunction2()* return "I'm done" and take a long time

> *AsyncFunction1()* and *AsyncFunction2()* return "wait"

>

>

> 1. With synchronous function calls, the sequence looks like:

> // This is where the response is actually sent to the client

> return *SendResponseUsingFunction1AndFunction2Data*(

> *SyncFunction1()*,

> *SyncFunction2()*);

>

>

> The thread executing code literally calls into *Function1()*, execute code,

> then return and call *Function2()*, wait for it to finish, and finally generate the response using *SendResponseUsingFunction1AndFunction2Data()*

> using the implicit context from the thread since it returns "I'm done". Very easy, straightforward to explain, and horrible for performance because

> *ASP.Net worker thread is tied up waiting for SyncFunction1() and*
> *SyncFunction2() which take a long time*
>
>
> *2. With asynchronous function calls common on the web, logical the*
sequence
> *looks like:*
> *AsyncFunction1Context.MainPageContext = Context*
> *return AsyncFunction1(CallbackFunction, AsyncFunction1Context);*
>
> *CallbackFunction(AsyncFunction1Context)*
> {
> *AsyncFunction2Context.AsyncFunction1Context =*
AsyncFunction1Context
> *return AsyncFunction2(AnotherCallbackFunction,*
AsyncFunction2Context);
> }
>
> *AnotherCallbackFunction(AsyncFunction2Context)*
> {
> *// This is where the response is actually sent to the client*
> *SendResponseUsingFunction1AndFunction2Data(*
> *AsyncFunction2Context.AsyncFunction1Context,*
> *AsyncFunction2Context,*
> *AsyncFunction2Context.AsyncFunction1Context.MainPageContext*
> *);*
> }
>
>
> *The original implicit context needs to be persisted until the end of*
the
> *async operation, so save it off. Then, the thread executes*
AsyncFunction1,
> *which simply queues up work to be done with a notice to call*
> *"CallbackFunction" when that work is done and then releases the IO*
thread..
> *The function returns "wait", so ASP.Net will wait to send a response*
until
> *told "I'm done". After a long time for AsyncFunction1 to finish,*
other
> *threads can pick up the queue'd item, execute it, and call*
> *"CallbackFunction" -- which in this example quickly queue's up*
another
> *asynchronous AsyncFunction2 and returns control of the thread. After*
a long
> *time for AsyncFunction2 to finish, other threads can pick up the*
queue'd
> *item, execute it, and call "AnotherCallbackFunction" -- where we*
finally do
> *the act of sending back the response using data that has been*
gathered and

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

> *tell ASP.Net "I'm done" -- so the response gets sent.*
>
> *Notice that in async model, the initial thread is NOT tied up to execute*
> *Function1 nor Function2. AsyncFunction1 and AsyncFunction2 still take a long*
> *time to finish, but it doesn't tie up the calling thread due to its queuing.*
> *Other threads can be used to do the queued items. Yet the response is still*
> *sent using data from Function1 and Function2, and clients do not notice any*
> *difference Time-wise because we're still executing the same sort of*
> *instructions -- though in terms of scalability, you just improved a whole*
> *lot since incoming threads are not tied to do expensive work.*
>
>
> *Yes, we are jumping through more hoops -- no one said asynchronous was*
> *easier than synchronous -- but the advantages gained are tremendous for*
> *people that take the time to understand .*
>
>
> *Substitute "Call to temperature Web Service" with "AsyncFunction2", ignore*
> *AsyncFunction1 (for completeness reasons), and substitute "Send response"*
> *with "SendResponseUsingFunction1AndFunction2Data()", and the picture should*
> *be clear.*
>
> --
> //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:1106345615.899119.146250@f14g2000cwb.googlegroups.com...
> Thanks David,
>
> *I think I grasp the concept between the two scenarios sync and async.*
> *One thing I'm still unclear about is what happens if your main thread*
> *has no other work to do then you are forced to wait for the async*
> *result or else your page will complete without the desired result.*
Here
> *is an example of what I mean:*
>

- > *IO Thread starts executing the Web Page*
- > *Web Page makes HTTP request (async) to web service say in the case a*
- > *temperature webservice*
- > *Webpage then does some processing on the result I get from the*
- > *webservice, say convert temp from fahrenheit to celcius.*
- > *Web Page displays the content of the HTTP response*
- > *Web Page finishes, returns "success", and releases IO thread to do*
- > *other work*
- >
- >
- > *Now with this example when I send the request to temperature*
- > *webservice*
- > *(async), my main IO thread can't do anything else unless it gets the*
- > *return value of the webservice since so in this case I would have to*
- > *wait for the request to finish is this correct? Also say I didn't*
- > *wait*
- > *for the result in my main page then what will happen is the request*
- > *will be fired off to webservice the page will return back to the*
- > *client*
- > *with incorrect result since we haven't got the result from*
- > *webservice,*
- > *then at some point the request from webservice will come back but it*
- > *will be too late because the response has already been displayed to*
- > *the*
- > *client. Is my thinking incorrect here?*
- >
- > *The way I understand async programming is if u need to do other kinds*
- > *of long process work that aren't needed straight away to continue*
- > *processing the job. Say if u had to log something to the database.*
- > **But**
- > *in the above example its clear that if you fire a request to*
- > *webservice*
- > *nothing more can be done until the result is back.*
- >
- > *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.*
- >
- > *I agree with the above point*
- >
- > *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.*
>
> *In regards to this point does this mean for my temperature example
> above that once I have fired an async request to the webservice my IO
> thread isn't tied anymore as long as I don't wait for the result? If
so
> wouldn't the page just end and then eventually the result will come
> back but the user may never see it?*
>
> *Sorry for asking so many questions just want to get really clear on
> this, as it looks like its a key concept to effective programming on
> the web
> Thanks for your time and effort
> Regards Rob*
>
>
>
>
> *David Wang [Msft] wrote:
> > 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 asnc completion
> of your
> > callback delegate
> > 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

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

> 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'll be starving the asp.net of worker process?

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

> >

> > *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*