

ASP/IIS Optimization Tips

Juan T. Llibre

Introduction

As racing car drivers know, top performance can only be achieved if the car's engine is tuned to perfection. The same principle applies to Internet Information Server (IIS) and its operating system platform, Windows. Unless your hardware, operating system, web server, data access, and scripting are finely tuned, you won't be able to extract top performance and scalability from your web platform.



Juan T. Llibre heads both the Computer Sciences and the Distance Education Departments at Universidad Nacional Pedro Henríquez Ureña in Santo Domingo, Dominican Republic. He co-authored the "Beginning ASP" Wrox series and has been a Technical Reviewer for several ASP-related Wrox books.

He is a Microsoft Most Valuable Professional (MVP), for Internet Information Server and Active Server Pages, who has gained a reputation at ASP Developer conferences as a knowledgeable and entertaining speaker, and has provided consulting services on ASP to the Caribbean Common Market, the Dominican Republic's Central Bank, and the National Technological University of Argentina.

He also runs two ASP resource centers, in English and Spanish, at <http://asptracker.com/> and <http://aspespanol.com/>, and can often be found answering ASP-related questions, in both languages, in the Microsoft public news servers.

For consultancies, he's available at j.llibre@codetel.net.do

ASP/IIS Optimization Tips

I'll mainly deal with IIS 5.0 running on Windows 2000. This setup has been implemented by quite a few leading enthusiasts and has been stealing the limelight from IIS 4.0 running on NT 4.0. I'll cover this older configuration too (albeit to a lesser extent), because it is still the workhorse for Microsoft web servers, although it is quickly becoming out of date. I will NOT cover ASP+ and the Microsoft.NET platform, since I want to provide tips that you can use *now* to improve your web server's performance. ASP+ will not make its official debut for a few months, precluding its use in production servers for a while, and extensive changes are bound to be necessary before that happens.

Web developers all over the world cheered when IIS 4.0 was introduced. The new platform offered incredible performance enhancements over IIS 3.0. Windows 2000 Server and IIS 5.0 offer even higher performance gains. With the tighter integration built into Win2K and IIS, and with the optimization of many server processes, you can now tune your servers to perform much faster and more efficiently than in previous incarnations.

One word of warning before I start: comprehensive coverage of ALL the things you can do to optimize a web server and its hardware would take a lot more than 1 ½ hours. In this limited time, I can only point the way and research will be necessary on your part.

Basic Considerations

RAM

Install as much RAM as possible. Efficient web processing takes RAM – lots of it.

Processors

Add as many, and the fastest, processors as you can. Faster processors will, obviously, process your ASP code and component execution faster and IIS scales effectively over 2 to 4 processors. A single-processor box will severely limit your scalability, but you can start with a single processor in a multi-processor enabled box and add processors as you monitor your server and determine that you need them.

When choosing the CPUs for your server, get them with a large L2 cache. The larger the L2 cache, the better the server's CPU performance because it shrinks the wait time when reading and writing data to memory.

Network Cards

I really shouldn't have to mention that you should get the best network cards you can get. Good NICs take over some of the I/O burden from the OS.

Developing Applications

Never develop applications on the server's console. Always use dedicated developing boxes and then transfer/install your applications to the server.

Bandwidth

Contract for as much bandwidth as you can. There's no use having a finely tuned server if you're sending the http data stream out through a straw.

RAID

Use disk subsystems that perform optimally. A Redundant Array of Inexpensive Disks (RAID) is the best option, although some might not find it as "inexpensive" as the name implies. Don't use NT Server's software-controlled RAID. Use hardware-based RAID, with a dedicated controller card, or even -if cost is no barrier- multiple RAID controllers and multiple disk arrays. Software-based RAID is much slower because it can't distribute the workload to separate processors, as do hardware-based RAID solutions. RAID controller cards usually have a large dedicated RAM cache (32MB is typical), which dramatically increases performance. Some form of RAID is necessary to achieve fast disk access times. RAID 0 (striping) is the fastest form of RAID disk access because it optimizes both reading and writing. RAID 5 (striping with parity) reads equally as fast as RAID 0 but writes much more slowly. RAID 0+1, usually called RAID 10, is the slowest RAID, but lets you sleep well, because it offers both mirroring and striping. Single-disk systems are the slowest performers and the most risky storage solution. Fault-tolerance capability is a must! And don't forget to get a good controller – buy the best I/O controller you can get. Top-notch controllers send much of the I/O work onto their own CPU, freeing up the server's CPU to perform other tasks, such as writing log files to a different hard disk.

Solid State Disks

While we are on the hardware front, Solid State Disks (SSDs), with their near-instantaneous information access and increase in I/O capabilities, boost the scalability of servers to new heights.

The real-time performance of an SSD can boost I/O performance by 10 to 20 times, improving existing single or multi-processor system performance without adding additional CPU or memory resources. An SSD features I/O capability of up to 8,000 I/Os per second! Maximum I/O rates of typical hard disks are less than 50 to 200 I/Os per second. Up to 10 to 20 high-performance hard disks operating synchronously might be needed to match a single SSD's transaction performance.

With access times that are more than 100 or 200 times faster than hard disks (60 - 100 *microseconds* versus 8 - 12 *milliseconds*), and transfer rates that are up to 15 times faster (13 - 30 MB/second for an SSD versus 2 - 12 MB/second for a hard disk), an SSD makes it possible for mission-critical information to be (almost) instantly available on demand.

For more information on SSDs, go to
http://www.quantum.com/src/whitepapers/wp_ntscalesd.htm

IDE RAID controllers

For smaller servers, used by developers who can't afford SCSI RAID disk arrays or SSDs, here's wonderful news. There are now IDE RAID controllers!

AMI has released an ATA/100 IDE RAID chipset. Iwill (<http://www.iwillusa.com/>) has boards which include the chipset, and which provide RAID 0, 1, and 0+1 capabilities. Look for motherboard Model KV200-R. They also make an ATA/66 IDE Raid chipset.

IFT Europe also has IDE RAID :
<http://www.iftraid.com/support/fqna.html>

NTFS

While we are talking about hard disks, don't forget that data in NTFS-formatted partitions should not exceed 80% of their capacity. NTFS needs room to work, and when you exceed 80% of the disk's capacity, NTFS becomes less efficient and I/O can suffer for it. Also, don't forget to defrag your drives regularly! That also helps.

Configuration

Hardware Needs

It should go without saying that, when traffic climbs, a single server is not enough. You will need multiple boxes to host an efficient website. A dedicated box for SQL Server is the most obvious example and multiple web server boxes are common. For that, you'll need dedicated hardware, like Cisco Director, or software-based solutions, like Microsoft's Network Load Balancing System (NLBS).

Unnecessary Applications

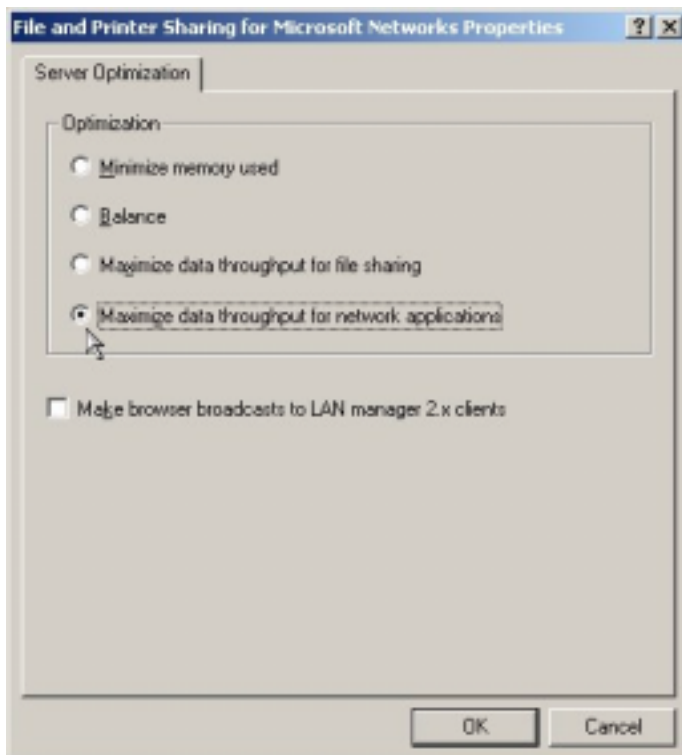
Don't install unneeded applications on the server. I chuckled quite a bit last month when called upon to find out why a box was serving pages very slowly. Turns out that they were running Office 2000 on the Internet server and using the server to store and serve Office documents. Plus, they had not turned off Office's "Fast Find". Bad idea!

Optimize Memory Usage for Internet Server Tasks

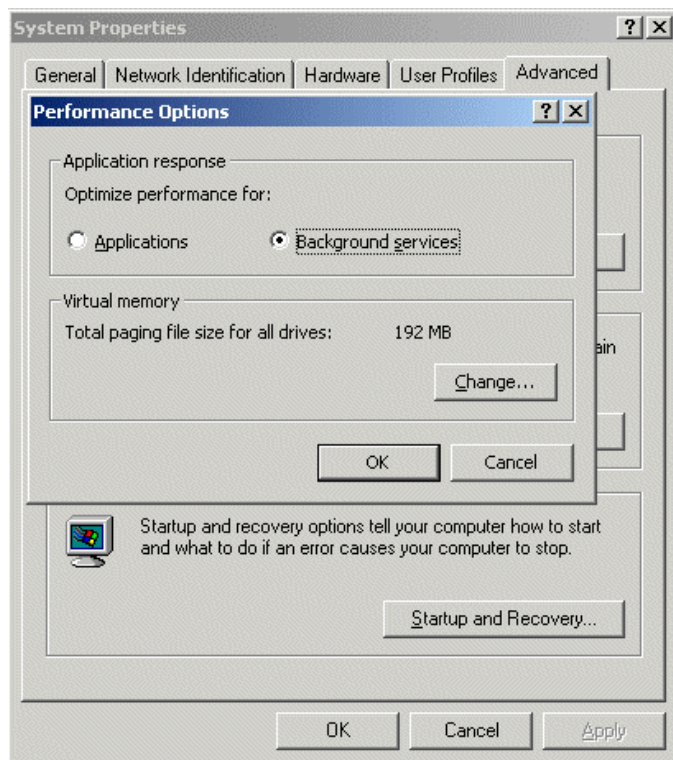
Under NT4, there are two settings that control application processes and memory usage. First, you should instruct NT4 to optimize memory for network applications instead of optimizing the serving of files. Open the Control Panel Network applet. Next, Click **Services** and right-click on the **Server** service. In the **Server** dialog box, select **Maximize Throughput for Network Applications**. This prevents NT4 from paging IIS processes to disk.

Secondly, you should prioritize background processes (e.g. IIS) instead of foreground processes (e.g. Notepad). Open Control Panel's System applet and open the Performance tab. Drag the slider to None and click OK.

With Windows 2000 click the Start button, select Settings, then Network and Dial-up Connections, and right-click Local Area Connection. Then open Properties. Select File and Printer Sharing for Microsoft Networks, and click Properties. On the Server Optimization tab (the only tab available), select Maximize data throughput for network applications. You will need to reboot the server for this setting to become operational.



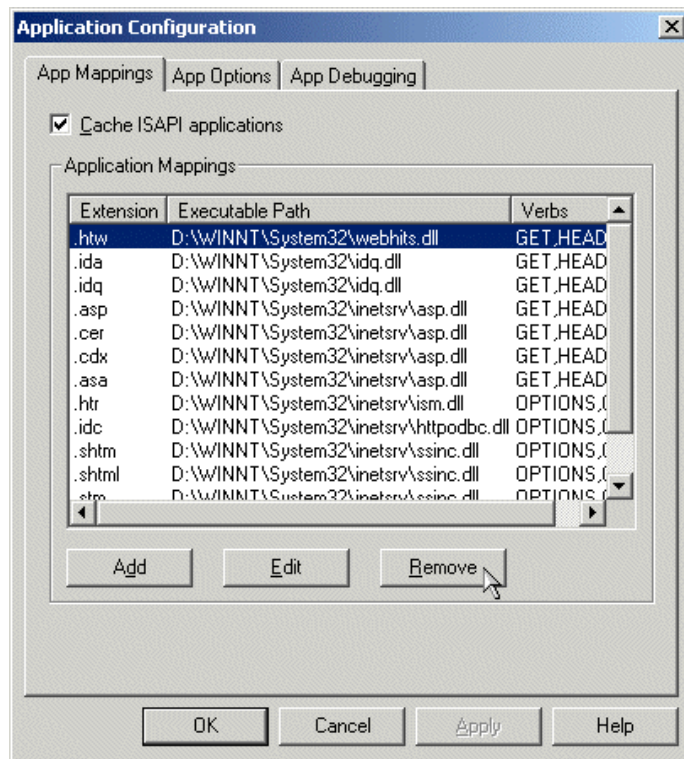
Additionally, you can optimize performance for either "Applications" or "Background Services" by right-clicking My Computer on the desktop. Select Properties, then Advanced, and finally Performance Options, where you can choose the appropriate radio button.



Disable Unused Application Mappings

Disable any Application Mappings that you are not currently using. In the IIS 4.0's Internet Service Manager, highlight the Default Web Site and select Properties. Then select Configuration and the Application Mappings tab. Remove any unused application extensions (highlight and push the Remove button).

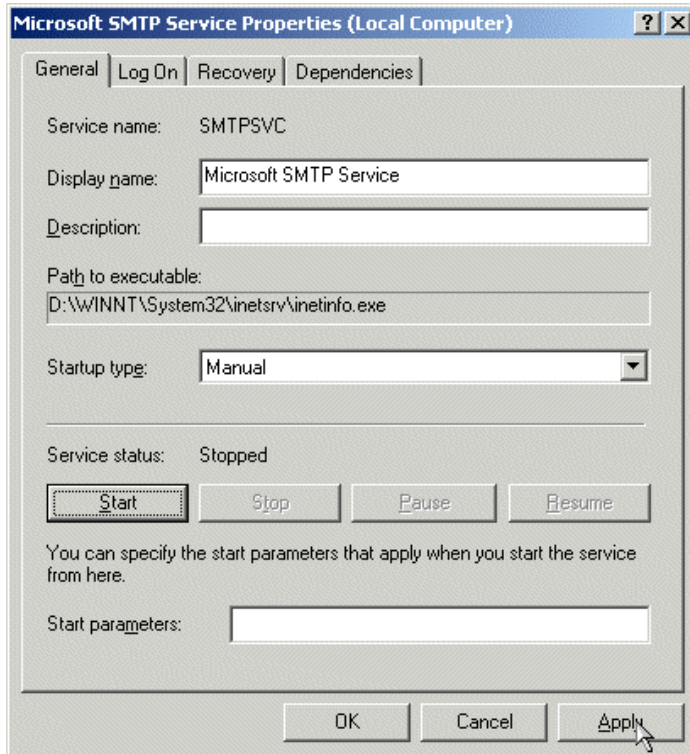
Under IIS 5.0 you need to the Internet Service Manager, highlight the Default Web Site, and select Properties. Then select Home Directory, push the Configuration button, and select the App Mappings tab. Remove any unused application extensions (highlight and push the Remove button).



Don't Run Unnecessary Services

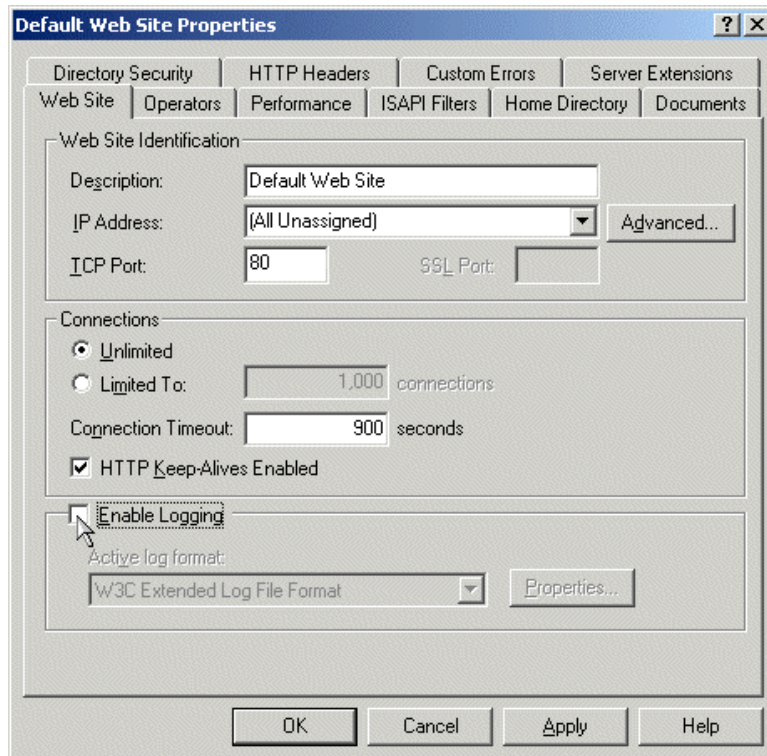
For instance, if you are not running news services, don't have the NNTP process running. If you don't run a mail server on the same box as the web server, don't run the SMTP service (unless you need to provide http-mail services).

To stop unneeded services, open the Administrative Tools menu. Click Services. Double-click the desired service, and configure.



Disable Logging

I was surprised to notice that, under high-traffic conditions, my web server was choking. Then I realized that almost everything the server was getting from visiting browsers was being logged, and that logging takes quite a bit of time and resources. When I turned logging off, the server breathed a sigh of relief. To disable logging when it's not needed, open the Internet Service Manager (ISM). Right-click Default Web Site and select Properties. On the Default Web Site Properties page, uncheck Enable Logging. Click OK.



Of course, logging is invaluable if your website is suffering denial of service attacks or you need detailed user stats, so consider carefully whether you should disable logging. If you decide to keep logging enabled, log to a striped partition with a controller that allows write-back caching or to a separate disk. This will decrease contention for the log file. Of course, if you log, you should log only critical data that you really need.

Member and Stand-Alone Servers

Configure NT Server 4.0 to be a *member server*, not a Primary Domain Controller (PDC) or a Backup Domain Controller (BDC). The task of being a domain controller drains away resources from serving data to the Internet. If you're using Windows 2000, configure your IIS box as a *stand-alone server* (there are no member servers any more in Win2K).

This completes our configuration tips. An excellent, comprehensive, list of NT4.0 configuration tips, and instructions for executing them is found at:
<http://www.pureperformance.com/NT/winNT.htm>

Most of them apply also to W2K. You will find it enlightening.

Performance Differences Between IIS 4.0 and 5.0

Now let's look at a few performance-related differences in the ways that IIS 4.0 and IIS 5.0 work.

COM Objects are Released Earlier

In IIS 4.0, COM objects are not released until page processing has finished. `set object = Nothing` releases the object reference for the Script Engine, but ASP does not release the object until the page goes out of scope.

In IIS 5.0, a COM object is released instantly – as soon as `set object = Nothing` executes and before the page goes out of scope. This makes it very important, with IIS 5.0, to clean up your objects as soon as you no longer need them, in order to be able to take advantage of this new feature.

Close and set to **Nothing** any connections, recordsets, or any other object instances you have created as soon as they are no longer required.

ASP Buffering is On by Default

In IIS 5.0, ASP buffering is on by default. Buffering improves performance by reducing network roundtrips. If you upgrade from IIS 4.0, you will need to turn on buffering because the setting is preserved. If you do a clean install, it will be set as the default.

IsClientConnected can be Used Before Sending Content to the Browser

In IIS 4.0, you had to send content to the browser before `IsClientConnected` would respond accurately. With IIS 5.0, you can call `IsClientConnected` before sending content to the browser. This assures that the client is still connected, before heavy-duty processing occurs.

You can Save Scriptless Pages as .asp

Under IIS 4.0, saving scriptless pages with an ".asp" extension would impact performance negatively. In IIS 5.0, .asp scripts that do not contain ASP code are processed almost as fast as if they were saved with .htm or .html extensions. This lets you save all your pages with the .asp extension, preventing the need to redirect if you later add ASP code, and simplifying page maintenance.

New IIS Settings

AspQueueConnectionTestTime

In IIS 4.0, requests in the ASP queue were always processed, even if the client was not waiting around for the response. In IIS 5.0, if the request has been queued for longer than the 3 default seconds, ASP checks to determine if the client is still connected before executing the request. If the client is not connected, the request will be removed from the queue, saving processing time.

AspThreadGateEnabled

IIS 5.0 performs thread gating when `AspThreadGateEnabled` is on. This automatically controls the number of threads that can execute concurrently when ASP detects blocked external resources during request execution.

Though self-tuning sounds great, it can be a source of problems. This setting is off by default, and you should leave it off until you've tested usage with a stress test tool, like the Web Application Stress Tool (<http://webtool.rte.microsoft.com/>).

Changed Settings

ProcessorThreadMax

ProcessorThreadMax (the number of allocated threads per CPU) has been moved from the Registry to the metabase and is now called `AspProcessorThreadMax`. Its new default is 25, instead of 10 as in IIS4.0. Microsoft recommends a maximum value of 100.

If ASP scripts make more requests to a COM object than the component can handle, exceeding the component's capacity to execute, blocking occurs. The KB article at <http://support.microsoft.com/support/kb/articles/Q238/5/83.ASP> contains information that should help you tune `AspProcessorThreadMax` to avoid blocking.

"To get the statistics needed to tune the `AspProcessorThreadMax` metabase property, observe the following System Monitor counters at peak load time using a one-second chart interval:

- Processor: %Processor Time (for each processor)
- Active Server Pages: Requests/Sec
- Active Server Pages: Requests Rejected
- Active Server Pages: Requests Queued
- Web Service: Connections/Sec

Following are some possible scenarios:

- If the Requests Queued counter never increases and the processor(s) utilization is low, the site has more capacity than currently needed.
- If the Requests Queued counter moves up and down, and the processor(s) are running below 50 percent, this indicates that some requests are blocking, and therefore, an increase in the `AspProcessorThreadMax` metabase entry may be in order.
- The Requests Queued counter grows uncontrollably along with CPU utilization. Check custom or third-party components. A component may have failed, and ASP is waiting for a response from the component.
- The Requests Queued counter grows and CPU utilization increases to an unacceptable level. Check the connectivity to databases that ASP is calling. A slow network connection, a large query, or a slow back-end computer can cause blocking.

When you run ASP under heavy load, the processor utilization should be above 50%. You should increase the size of `AspProcessorThreadMax` until the processor utilization reaches at least 50%.

Note that just as with the Registry, incorrectly modifying the metabase can cause serious problems that may require you to reinstall IIS 5.0. Modify the metabase at your own risk.

To change the `AspProcessorThreadMax` metabase property, "Run the `adsutil.vbs` utility from the `<%SYSTEMDRIVE%\inetpub\adminscripts` directory. To re-configure the `AspProcessorThreadMax` metabase property, type the following command:

```
adsutil.vbs set w3svc/AspProcessorThreadMax <NewValue>
```

Where <NewValue> is the number of threads that ASP should use per processor. This sets the value at the Master WWW Properties level where it is inherited by all new Web applications and all existing applications that have not explicitly set a different value for AspProcessorThreadMax."

Alternatively, run MetaEdit 2.1 and make the appropriate change to the metabase (make sure you use MetaEdit 2.1 and not 2.0, since it has been documented that version 2.0 corrupts IIS 5.0's metabase).

"ASPPROCESSORTHREADMAX should not be changed unless blocking is occurring. If the Total Queue Length grows and the Processor Time grows to an unacceptable level, troubleshoot custom components or database connections."

AspScriptEngineCacheMax

In IIS 5.0, the default for the maximum number of script engines to cache in memory is now 125. In IIS 4.0, the default was 30. This setting allows each ASP thread to cache a script engine, which means that ASP scripts process more efficiently.

AspScriptFileCacheSize

IIS 5.0 now has a default limit of 250 files. For IIS 4.0, the default was -1 (no limit) and the cache would consume a lot of RAM. The new default in IIS 5.0 helps keep the cache size in check, conserving RAM availability.

AspRequestQueueMax

Now IIS 5.0 defaults this to 3,000. For IIS 4.0, the limit for requests in the queue was 500. This means that 3,000 requests can be waiting in the queue now before "Server Too Busy" is returned. All of the settings discussed here can be changed, but make sure you have a good reason to do so before going ahead with changes. Also, Be aware that IIS 5.0's "Application Protection" setting replaces IIS 4.0's "Run in a separate memory space".

Isolation Levels

In IIS 5.0, you now have the following choices for isolation levels:

- Low – The application runs in the same process space as `inetinfo`. You may gain some performance, but you might jeopardize the stability of your web services if the application misbehaves.
- Medium (Pooled) – This is the default setting. Pooling means that all web applications share the same instance of `dllhost.exe`. So, instead of running in the `inetinfo` process space, all applications share an instance of `dllhost.exe`. While doing this saves the `inetinfo` process from interference, any one application that fails can bring down all the other applications.
- High (Isolated) – This setting lets you run completely isolated applications. This is the only safe route to take if an application has not been thoroughly tested yet. If an application running in an isolated process space crashes, the rest of the web server does not crash along with it.

Although low isolation may buy you some performance (by running within the highly efficient `inetinfo` process space), you could probably tweak another part of your application rather

than put at risk inetinfo's process space. High isolation is where you should run web applications that you don't trust because they have not been completely debugged. Placing such an application in isolation prevents a wayward application from stopping the rest of the web services you provide. Medium isolation offers a good tradeoff between performance and reliability; stay with medium for most applications.

Tips for Both IIS 4.0 and 5.0

Disable ASP Debugging

Disable ASP debugging if you've enabled it on the production server. You shouldn't be developing/debugging on the production server anyway, but let's face it, sometimes we must. If ASP debugging is enabled, the application is locked to a single execution thread, causing extreme slowdowns.

So, check for debugging with the Internet Service Manager (ISM) and disable debugging!

Stress Test, Stress Test, Stress Test!

By now, you should all know about the Web Application Stress Tool. There isn't enough time available in this session to go into its details (see the "Web Stress Testing" talk by Matt Odhner instead), so just go to the following address, download it, and make sure you use it frequently – but, don't install it on the Web Server! Install it on another box on your network.

For a good introduction to the tool's usage, see <http://msdn.microsoft.com/workshop/server/asp/server092799.asp>.

What else can we do to maximize performance?

Optimize Your Scripting!

Install the latest Scripting Engine. If you don't have it, it's available at <http://www.microsoft.com/scripting/downloads/ws/x86/scripten.exe>.

In addition, use advanced features like `with` in VBScript.

```
With RS
    .CursorLocation = adUseClient
    .CursorType = adOpenKeyset
    .LockType = adLockOptimistic
    .CacheSize = 50
    .Source = "Select CustomerID, LastName, FirstName, from Clients"
    .ActiveConnection = cn
    .Open
End With
```

This will be faster because the scripting engine doesn't have to re-qualify the name of the object. Everything within the block already has a qualified object it belongs to, as opposed to the script engine having to qualify each property's "owner" if the property isn't set within a block.

Under NT 4.0, never use the Dictionary Object in global.asa

Use the Lookup Table object, instead. This avoids threading problems associated with Application-scoped or Session-scoped dictionaries. The Dictionary object is apartment-threaded – even though, when it was introduced, developers were told that it was safe to mark it as both-threaded. It is *not* safe to do so, and if the dictionary object *is* marked as both-threaded, data corruption will result. All requests for the Dictionary object are locked down to the thread on which the object was first called and every other request has to wait in line to be executed. This slows the server down quite a bit. For additional information on the threading model, see

the topic "Selecting a Threading Model" in the IIS documentation, found at default location <http://localhost/iishelp/>.

Download the Lookup Table Object at http://msdn.microsoft.com/workshop/server/downloads/lkuptbl_eula.asp

Or obtain basic information at <http://msdn.microsoft.com/msdn-online/MSDNchronicles2.asp>.

Use Functions to Write Your HTML

The inherent efficiency of script blocks will speed up your HTML rendering. Use:

```
<!-- #include file="header.asp" -->
<script language="vbscript" runat="server">

    Sub WriteBody()
        ' whatever your body contains
    End Sub

    WriteBody 'call the WriteBody function

</script>
<!-- #include file="footer.asp" -->
```

Not only will it speed up your rendering, but site maintenance will be easier as well. Whenever header or footer info changes, changes to the includes will update your whole website in one fell swoop.

Response.Write All Your HTML

When you alternate between HTML and script code, the server's response time is slowed down because two different DLLs write to the html stream being returned to the client.

Let me show you a simple example. The following:

```
<html>
<head>
<title>Test</title>
</head>
<body>
<%=date %>
</body>
</html>
```

will always be slower than:

```
<%
Response.Write "<html>" & VbCrLf
Response.Write "<head>" & VbCrLf
Response.Write "<title>Test</title>" & VbCrLf
Response.Write "</head>" & VbCrLf
Response.Write "<body>" & VbCrLf
Response.Write date & VbCrLf
Response.Write "</body>" & VbCrLf
Response.Write "</html>" & VbCrLf
%>
```

The more you alternate between script and HTML, the slower your server will perform.

Use Local Variables

Using local variables may save you a few server CPU cycles. Here's an example:

```
Set a.b.c = a.b.d
If a.b.e = a.b.f then
' ...rest of code
```

This is terribly inefficient. Let's see what actually happens when that code runs:

- a is resolved as a global object
- b is resolved as a property of a
- c is resolved as a property of a.b
- d is resolved as a property of a.b
- The value of a.b.c is set to the value of a.b.d

- a.b.e now needs to be resolved, which means that:
 - a has to be resolved *again* as a global object
 - b has to be resolved *again* as a property of a
 - e is resolved as a property of a.b

- Now we have to resolve a.b.f, so:
 - a has to be resolved *again* as a global object
 - b has to be resolved *again* as a property of a
 - f is resolved as a property of a.b
 - a.b.e is compared to a.b.f (finally!)

Compare that to the much faster:

```
Set Obj = a.b
' i.e., resolve "a.b" once
Set Obj.c = Obj.d
' the value is obtained instantly, since "a.b" is already resolved
If Obj.e = Obj.f then
' the value is obtained instantly, since "a.b" is already resolved
' ...rest of code
```

Neat, isn't it?

Store Server-Side Variables as Local Variables

This will significantly speed up your pages, and is particularly applicable to recordset values and anything retrieved from the Request object. So, instead of:

```
<%  
If rs ("name")="Juan" Then  
    ' do something  
End If  
If rs ("name")="Joe" Then  
    ' do something  
End If  
If rs("name")="Bill" Then  
    ' do something  
End If  
%>
```

use the following:

```
<%  
name= rs("name")  
If name="Juan" Then  
    ' do something  
ElseIf name="Joe" Then  
    ' do something  
ElseIf name="Bill" Then  
    ' do something  
Else  
End If  
%>
```

With values retrieved from the Request object, instead of the following:

```
<%  
If Request.Form("name")="Juan" Then  
    ' do something  
End If  
If Request.Form("name")="Joe" Then  
    ' do something  
End If  
If Request.Form("name")="Bill" Then  
    ' do something  
End If  
%>
```

use this code:

```
<%  
name=Request.Form("name")  
If name="Juan" Then  
    ' do something  
ElseIf name="Joe" Then  
    ' do something else  
ElseIf name="Bill" Then  
    ' do something else  
Else  
End If  
%>
```

The same applies to `Request.QueryString("somevalue")`.

Use Literal Paths

When you use `Server.MapPath`, the path is retrieved via a separate request. That takes time. Paths such as `c:\data\app\myfile.ext` will always be resolved faster than `Server.MapPath "/app/myfile.ext"`.

Validate Client-Side Whenever Possible

Server-side validation overloads your server. Think about what is happening when your server is validating 100 simultaneous requests. Now think of the processor time saved when each client does its own validation.

Use as Few Server Variables as Possible

The tendency when starting ASP programming is to store everything in server variables, so we can have fast access to the variable values should we need them. Don't! This is especially critical if you store data in server objects like recordset objects or arrays – which you should never do anyway because memory usage shoots up exponentially.

Don't Redim Arrays

It is quicker to just create a new, larger, array than to redim an old one.

Don't use multiple scripting languages in the same script

Multiple languages in the same script means that multiple script engines are instantiated for that script. This clearly means more memory overhead.

OK – we've run through a few scripting optimization tips. Some of them you're bound to have heard before, but it's always good to be reminded of them. *What else can we do?*

Optimizing Data Access Performance

Data access and retrieval is often the most treacherous performance area for the web application developer. Top on my list of "must do's" is to use the latest MDAC!

MDAC 2.5 is vastly superior and more efficient than any previous incarnation. It's available at http://www.microsoft.com/data/download_250rtm.htm

Be aware that there are different MDACs for different needs. Use the Component Checker to make sure you can identify your current MDAC installation, and so you can diagnose any installation issues. Get it at: <http://www.microsoft.com/data/download.htm>

NOTE: MDAC 2.6 should be coming soon. ADO 2.6 is included in SQL Server 2000.

Bind Type Libraries.

It's a fairly common practice with IIS 4.0 to include large constant files with an ASP `#include`. This approach has problems. The `include` file is usually large and only a few of the constants are used, which is wasteful.

IIS 5.0 allows you to bind your web applications to a type library, making all of the constants available to all of your ASP scripts. To bind the ADO 2.5 type library, for instance, add the following to your application's `global.asa`:

```
<!-- METADATA NAME="Microsoft ActiveX Data Objects 2.5 Library"
      TYPE="TypeLib" UUID="{00000205-0000-0010-8000-00AA006D2EA4}" -->
```

-->

Keep in mind that that code text is wrapped. It should all be in a single line.

Now you can use ADO's constants natively, in any script in the application, without incurring bulky `include` file overhead. Alternatively, pare down the constants file to as tiny a working set as you really need.

If you're using SQL Server (you all should be using SQL Server instead of Access, heh heh!), you can make ADO free-threaded. There's a batch file for this in the ADO directory on your hard drive:

```
X:\Program Files\Common Files\System\ado\makfre15.bat
```

or use .REG file `X:\Program Files\Common Files\System\ado\adofre15.reg`

Cache Results from Data Sources that are Stable

For instance, if you are currently using ADO to populate a list box that will contain the cities in which your company has offices, use the `FileSystemObject` in an admin-only file to stuff the data into a file snippet whenever the data changes. Then "include" *that* into your ASP script.

Now, requests for that list box information can be fulfilled without making an expensive ADO call to a data source. OK –that's not really "caching", but you get the gist. Interestingly, someone recently mentioned that you could also use a shared MTS component that does the same thing but has data in RAM. By setting a data refresh interval, you could, conceivably, update the data inexpensively. I have not looked into this in detail, but it certainly sounds feasible.

For an alternative way to save server effort, you could use a client-side cursor, and disassociate the recordset from the `Command` object by setting the ADO `ActiveConnection` property to `Nothing`.

Don't Cache COM Objects in the Session or Application Objects

Cached COM objects in Session objects tie the Session to a single thread. Cached COM objects in the Application object cause all calls to the object to have to be marshaled and serialized, adding substantial overhead. In both cases, contention issues cause severe decreases in performance.

Use OLE DB Connection Strings

Native OLE DB connection strings are faster than ODBC DNSs and "DSN-less" connections. The question is, how much faster? Take a look at this table:

Performance Comparison					
SQL			Access		
	OLEDB	DSN		OLEDB	DSN
Connection Times :	18	82	Connection Times :	62	99

Use Stored Procedures

How many times have you heard *this* tip? Guess what? It's still one of the most under-used optimization techniques! Queries executed through stored procedures are faster than queries passed through a SQL query string because they bypass object library overhead.

You can use either the `adCmdText` or the `adCmdStoredProc` property to tell ADO that you want to execute a stored procedure. Always use the `adCmdStoredProc` property when creating a

Command object to execute a stored procedure. This boosts performance in comparison with using the `adCmdText` property because it bypasses parameter translation.

Set the `adExecuteNoRecords` option if your stored procedure will not return any rows. This instructs the Command object to not request a returning rowset, which saves a little overhead and reduces memory usage.

If You're Not Using a Stored Procedure, Create Views Instead

Calling a view is not as efficient as using a stored procedure, but it's much more efficient than using SQL in your ASP scripts or COM components.

Use Stored Procedures to "Batch" Related SQL Statements

Instead of calling a separate stored procedure for every database task you want to perform, use stored procedures to "batch" a group of related SQL statements. This reduces network traffic and server overhead.

Choose the Cursor that Uses the Least Resources

When at all possible, use the "fire hose" (Forward-Only) cursor, which consumes the least overhead of the four cursor types available through ADO.

Index Your Databases Intelligently

An intelligently indexed database will perform dozens of times faster than an un-indexed one, or an over-indexed one for that matter. Just for fun and if you don't believe me, index *all* the fields on any database and run a benchmark on your data access scripts. Index the most-frequently used fields in your "Order By" and "Group By" SQL statements.

Do Not Use ADO Record Addition and Deletion Methods

Avoid methods such as `AddNew` and `Delete`. If your application adds and deletes records often, it will perform better if you use the SQL `INSERT` statement.

Increase the ADO CacheSize Property

Set the ADO `CacheSize` property to an appropriately larger number than the default (1), depending on the size of the recordset returned.

By making ADO retrieve multiple records in one transaction with the data source, you eliminate some of the overhead involved in database transactions. You will probably see benefits if you set `CacheSize` to equal the lowest of either the number of records expected, or 100.

And, for my final recommendation...

Componentize Your Applications!

Windows DNA works! Period. There's really no need for long sausage-style scripts, which tie up server resources for extended periods of time. Well-designed components, whether written in C++ or Visual Basic, will speed up your site a lot.

Think about this for a second. When a browser requests a script, the code in it has to be interpreted before a response can be sent by the server. Code in components lives in RAM and so has a much faster response rate, over and above it being in faster native code as opposed to P-code. It's amazing to see developers who have no time to develop components, but have plenty of time to invest in writing slow-performance spaghetti scripts.

On the other hand, instantiating a COM object also takes time. If your data access code is short and is not reused, then using an ASP script may be more efficient than using a COM object.

But, if you have fairly complex database calls and use them a lot in your applications, by all means, *componentize your applications!*

Summary

That's it for today. I hope you can use some of the tips I've presented to your advantage. I can't leave without giving credit to a few people. I am particularly indebted, for some of the material presented, and for the inspiration to pursue optimization and troubleshooting information, to Mark Anders, J.D. Meier, George Reilly, and Mike Culver, "softies" all. Thank you guys - without your help and inspiration I wouldn't be here. Also, thanks to Mom and Dad. Without **their** efforts I **really** wouldn't be here. Finally, I also want to thank Wrox Press. As a result of Wrox's efforts, putting together conferences like this one, we all have a chance to meet and exchange valuable information on what to some is a duty, but to most of us is a passion. See you in Las Vegas and Amsterdam!