Technical White Paper Series

MULTI-TIER PERFORMANCE TUNING

Matthew Baute
G. A. Sullivan

Don Benage, Series Editor

www.gasullivan.com

G. A. Sullivan

is a global Internet software development company focused on providing e-business solutions for middle market companies, e-business startups and Fortune 1000 companies. With nearly 400 professionals across seven U.S. offices and one European office, G. A. Sullivan offers complete strategic, creative and technical expertise to satisfy its customer's e-business requirements. G. A. Sullivan has been a leader in software development since 1982. The company's focus allows them to concentrate on quality and helps them to attract some of the most talented Internet software professionals in the industry.

G. A. Sullivan has formed teams of business and technical experts who focus on specific industries creating intelligent and robust solutions. Strategic planning, product positioning and project management services are applied along with a proven methodology for solving complex technology and e-business problems. This ensures predictable and repeatable results. The company's Line of Business expertise includes financial industries (banking, insurance and securities), manufacturing & distribution (ERP integration, logistics and supply chain management) and many others. The company has high level expertise building E-Business, Business Intelligence (Data Warehousing), Knowledge Management and Security solutions that apply to all industries.

G. A. Sullivan has received national recognition for fast growth and industry leadership over the past several years. Recent recognition includes selection as Microsoft's Great Lakes District Partner of the Year for 2000 and the St. Louis Business Journal Laclede Award for best Corporate Culture among St. Louis companies with more than 250 employees. In addition, Gregg Sullivan was named an Ernst & Young Entrepreneur of the Year 2000 award winner for the St. Louis region. In 1999, the U.S. Small Business Administration selected Gregg Sullivan as the National Small Business Person of the Year from a field of 53 state winners. For two consecutive years, G. A. Sullivan was named Microsoft's Mid America District Partner of the Year and ranked on Inc. Magazine's Inc. 500 list of "fastest-growing private companies in America". The company has also been recognized on the Deloitte & Touche Technology Fast 500 list of "fastest-growing technology companies in the U.S.", for the last three consecutive years. G. A. Sullivan office locations include St. Louis (headquarters), The Netherlands, Kansas City, Cincinnati, Nashville, Atlanta, San Antonio, and Fairview Heights, Illinois. For additional information visit www.gasullivan.com.

G.A. SULLIVAN *business*^{e}

Contents

Multi-Tier Performance Tuning

For information, contact:

G. A. Sullivan

55 West Port Plaza

Suite 100

St. Louis, MO 63146-3131

URL: www.gasullivan.com

e-mail: corporate@gasullivan.com

Phone: (314) 213-5600

Fax: (314) 213-5700

First Printing (September, 2000)

Multi-Tier Performance Tuning

Introduction

The Windows DNA platform is an excellent choice for businesses building technical solutions to business problems for a variety of reasons. The platform is founded on proven Windows NT operating system technology, is easily extended through a variety of Application Programming Interfaces (APIs), and is widely supported in the industry.

However, the technical power driving the Windows DNA platform is derived from its complex set of interlocking pieces, and these pieces must be individually and collectively well-tuned for high performance. A thorough understanding of the pieces and how to best tune each is required by the individual assigned with the responsibility of either making an existing system achieve desired performance levels, or with planning a scalable and high-performance design of a new system. Additionally, a solid understanding of the tuning tools at the disposal of developers and testers must also be gained in order to uncover potential bottlenecks before a system is deployed.

This document attempts to provide a comprehensive look at the Windows DNA components, tips on how to tune each piece to achieve optimal performance, and a look at the tools that are helpful for enhancing performance. Because this paper is targeted largely for developers, a significant portion of the document covers proper development techniques within the three logical layers in which developers spend most of their time. Although you might not expect so much discussion of specific development techniques within a paper on performance tuning, it's crucial to implement solid code to ensure a highly performing system. It's not at all difficult to kill performance with a single unnecessary loop, or by missing an index on a frequently accessed table.

Tuning Overview

Why do I need to do performance tuning?

Beyond the obvious reply of making applications and systems run fast, there are a few more tangible reasons for undertaking a tuning effort and for justifying the resources that will be needed.

First, applications are implemented to meet specific business goals. New systems are developed for a variety of reasons: to solve an existing business problem by making a process more efficient, to replace an outdated system with new functionality, to take advantage of a new medium such as the Internet, or to gain a competitive advantage by offering customers an enhanced or unique service. If the new application meets all requirement specifications and fulfills all use cases but performs slowly, the new process may not actually be more efficient, will be in large part as outdated as the system being replaced, and will not offer the desired competitive advantage. For this reason, close attention should be paid to performance in order to meet the business goals of the company.

Second, in this increasingly technological age, we've come to expect applications to perform faster and faster. With the speed of affordable, widely available processors now reaching 1GHertz and the cost of RAM and disk continuing to drop, everyday computer users expect screens to update and refresh quickly. Web users will only wait a few seconds for each page to load, or else it's on to the competitor's site. In order to meet the general expectations of today's users, applications must be tuned so that queries to databases are returned in a reasonable time, mid-tier COM components do their job quickly, and Web pages are processed and sent out within a few seconds. Paying attention to meeting end-users' expectations and needs has always been a key strategy of Microsoft. We can take a page from the book of the most successful software company in history by listening and learning from what users of our applications are saying.

Third, performance tuning of existing systems is preparation for future growth. It's not uncommon for many Internet "dot-com" start-up companies to experience double, triple, or even exponential rates of growth within very short time periods. In order to ensure that a quality user experience is provided as these high rates of growth occur, performance tuning helps answer the "what if" questions before such situations actually arise. It helps provide assurances that performance will be adequate as a system scales larger and larger. It gives the organization a chance to make sure that the proper hardware resources are put in place in advance of the big Super Bowl commercial.

For these reasons, performance tuning should be a recurring task on the project plan. Measurable, concrete goals must be provided or else tuning will be without end. How do you know if you've reached your goal unless you have a specific mark for which to shoot? Abstract business goal must be translated into concrete benchmarks such as number of pages served per second, number of transactions completed per minute, etc. When performance goals are met, tuning has been completed until the business starts planning a new growth cycle.

How do I go about tuning?

After a few measurable goals have been set, tuning can begin. A load is put on the system using a tool such as the Web Application Stress Tool (discussed in more detail later). Reports generated by the tool along with Performance Monitor counters are analyzed to determine where bottlenecks lie. Is there a problem with specific ASP pages? Is there enough RAM on the SQL Server box?

A solution to alleviate the current bottleneck is proposed, implemented, and the system is stressed again. Did the solution help? If so, take note and continue by stressing again, finding the next bottleneck, then testing another potential solution. This process is iterative, repeated over and over until performance goals are met.

The cost-effectiveness of each potential solution should be kept in mind. By nature, developers tend to lean towards rewriting ASP pages or middle tier COM components rather than simply adding more RAM to a server. However, if adding more RAM solves the problem in and of itself, this approach is a more cost-effective solution and should be implemented.

When do I tune?

1. Tune from the start!

A common misperception among developers and project managers alike is that tuning should take place once development has reached the alpha or beta stage. *Beginning your tuning at such a late stage in the life cycle can have dire effects.* Considering the performance requirements of an application should begin *from the start*.

The technical architect responsible for the overall design of the system should be thinking of performance when designing the data model, the object model, and when choosing the implementation technologies. How normalized should the database be? To what level of abstraction should the object model be designed? Is it necessary to write a COM component for a specific task in C++ for extra speed? All these questions must be carefully weighed and answered by the architect very early in the design phase. *Trying to correct these early decisions once a product is in beta release can be an arduous task.*

It's crucial to get feedback from colleagues about the choices being made early in the design phase. See if the design makes sense to others who've been down similar roads before. Most importantly, solicit feedback from an experienced DBA on the data model – it's hard to overemphasize how important a well designed data schema is to the success of a highly performing application.

2. Tune throughout development

Performance should also be kept in mind by the programmers implementing the system every day during the development phase. Algorithms must be implemented efficiently for the application to perform well. Poor choices or shallow comprehension can cause the overall performance of an application to suffer. If developers are trained to be mindful of how the code they're implementing has a direct effect on system performance, the burden of tuning further down the road will be lessened.

A helpful practice during the development cycle is to convene regular code reviews. In these meetings, small teams of developers examine code samples, discuss implementation approaches, and trade general tips and tricks. People new to a technology or language will benefit from this exercise by listening to more seasoned staff who've experienced the pain of real-world lessons. Poorly implemented code can be identified and fixed early in the development life cycle, making it less likely that expensive rework will need to be performed later.

3. Tune at Pre-release

It makes obvious sense to stress-test an application prior to release. Put a significant load on the system and see what happens. Does it meet the specific performance goals for the release? If not, tune the hardware or software configuration until it performs to specification. The goal is to eliminate problems before any users touch the system.

4. Tune Post-release

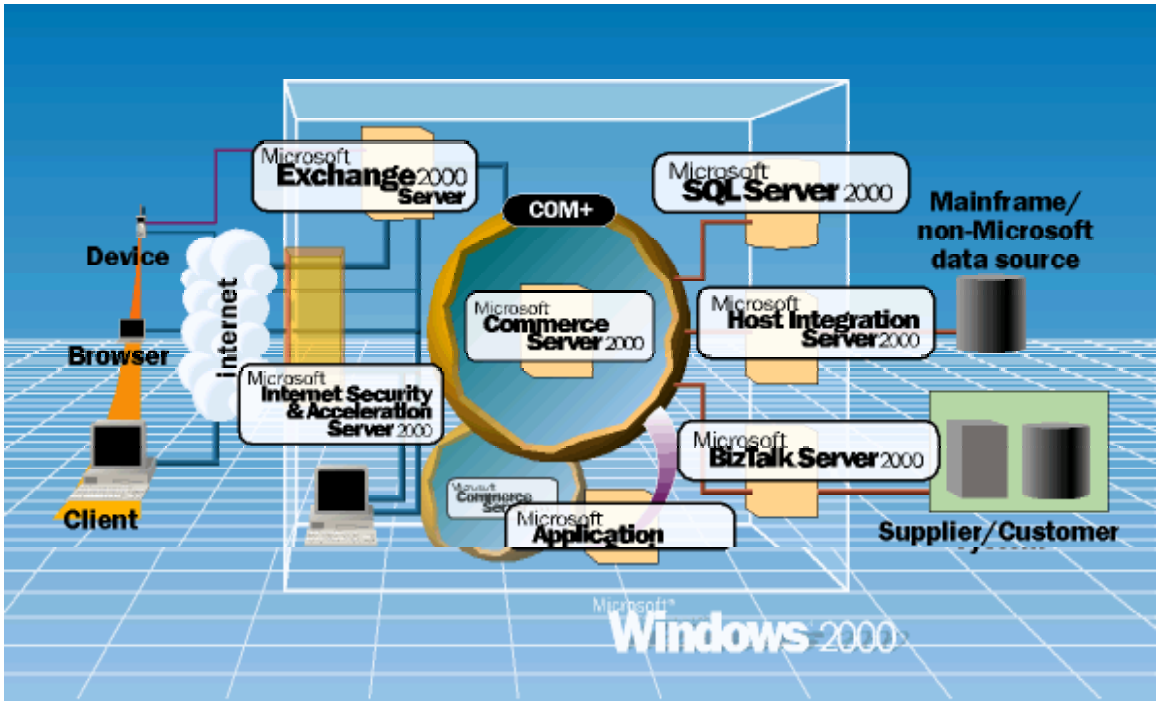
Once an application has been released to the user community, analysis should be performed on log files to determine real-world usage patterns. Perhaps users really like

a certain section of the web site that you didn't expect. In this case, it's beneficial to stress-test and tune that section to make sure it's performing well.

Is the application performing adequately now that it's live? If not, you may need to tune in emergency mode, monitoring production servers and making changes quickly to ensure that users are getting the quality experience that's intended.

Windows DNA

The Windows DNA architecture consists of three logical layers: presentation services, business services, and data services. Presentation services drive the user experience, capturing and validating inputs and displaying outputs. Business services house application rules and logic, compiled into fast-executing code modules. The data services persist the entities and their interrelationships, providing value-added features such as indexing for quick searching.



These three logical layers may be separated into many different physical configurations. For example, the presentation services could be a Windows desktop or an Internet Explorer browser. The business services could reside on the same physical machine as the data services, or on a completely different machine. In an application with very high-volume transaction requirements, the logical business tier may reside on multiple machines while the data services might be partitioned across multiple machines.

This document addresses the issues with making each logical layer perform well: presentation, business, and data. In the Microsoft Windows DNA platform, the presentation services consist of either a Windows desktop (rich client) or a WWW

browser (thin client). The business services are contained in compiled Visual Basic, Visual C++, or Visual J++ components running in the COM+ (MTS) application server environment. The data services are housed in Microsoft's enterprise database, SQL Server.

Additionally, Windows DNA provides system services underlying the three logical layers. These services consist of the Windows 2000 (or Windows NT 4) Server operating system, and components integrated into the O/S such as Internet Information Server. Tuning this System Service layer is also covered in this document.

Finally, the physical configuration of servers (CPUs, hard drives, RAM) and the network also plays a key role in delivering a highly tuned application. This paper will touch on important considerations that should be addressed relating to the physical implementation of applications.

Since a majority of middle-tier business logic is increasingly written in Microsoft Visual Basic, this language will be the focus of the Business Services section of this document. VB offers developers a very efficient programming environment, enabling the development of complex COM components in a relatively short time-frame. Comparatively, writing COM components in C++ or J++ takes additional time and expertise that's in short supply in our industry. As with most choices, trade-offs are involved. C++ offers many low-level speed advantages that may be a necessity in some applications. However, for most Windows DNA applications, VB will fill the requirements for business object development. Quite often mid-tier methods are simply passing parameters from the UI to a stored procedure in the database, and C++ offers little performance gains over VB in this case.

Microsoft is currently in the midst of releasing a new language called C#, promising most of the power of C++ combined with the Rapid Application Development (RAD) features of VB. The entire Microsoft.NET platform being rolled out over the next few quarters promises great productivity enhancements, but will certainly also present new challenges to building highly scalable, well-tuned applications.

I. Presentation Services

Presentation services in the Windows DNA platform are largely developed using the following technologies:

- VBScript / JavaScript
- Active Server Pages (ASP)
- HTML / XML

The web browser makes a request to the web server. The web server processes the ASP page, perhaps instantiating a COM component or reading data from the database. The ASP scripting engine parses <% %> tags, running the logic line by line and replacing strings with text, converting it to HTML. The constructed page is then sent back over the Internet to the browser.

VBScript / JavaScript

ASP pages are written with VBScript. It's important to remember that VBScript, like JavaScript, is an *interpreted* language. This means that it doesn't talk the native language of the processor, nor does it benefit from any optimizations made prior to run-time. Contrastingly, when the Visual Basic compiler is invoked on source code, many things happen to make the compiled code much more efficient than the VBA source code entered in the IDE. Code is converted to machine language, speaking directly to the CPU in its native tongue. Any performance enhancements to be made to VBScript code will focus on making the interpreter run more efficiently. These include the following:

1. Limit the size of your VBScript

If you have an ASP page with over a few hundred lines of nothing but VBScript (not including any HTML you may have further down the page), consider putting this VBScript code into a mid-tier COM component. The COM component is compiled and will usually run much quicker than the interpreted VBScript code in the ASP environment. You shouldn't have to do a lot of conversion from VBScript to VBA, so it's a fairly simple task. Consider this approach especially as a project continues to evolve and mature. It's not hard for chunks of VBScript code to start growing in length. If they get too large, pull them out.

2. Limit variable scope

Always make the scope of variables as limited as possible. For example, if you only need a variable within the context of a specific function, DIM the variable within that function, not at the beginning of the ASP page.

The hierarchy of variable scope within ASP is as follows (the lower the scope, the more efficiently memory is managed):

- Application
  - Session
    - Page
      - Function/Sub

3. Avoid the ReDim statement

The ReDim statement, especially with the Preserve keyword, is a very expensive function within VBScript (and also within Visual Basic). In many cases it's less expensive to declare the array beforehand with a "guesstimate" for the array bounds rather than executing a ReDim with the exact sizes.

4. Keep in mind JavaScript goes over the wire

Remember that client-side JavaScript is passed over the Internet along with the HTML in the page. If you have a lot of JavaScript going over the wire each and every page request, you may be taking up bandwidth resources that can be better utilized.

Consider putting commonly accessed JavaScript routines such as date/time functions or browser version checks in an external JavaScript file. Reference the file as an external

script from the HTML page rather than having the code embedded within each page. In addition to better modularity, the browser can cache such lines:

```
<SCRIPT Language=JavaScript src="../include/utilities.js"></SCRIPT>
```

Also remember that client-side JavaScript executes using the CPU, disk, and RAM of the client machine. If you have significant code to be run in JavaScript, the client machine should have adequate processing power. Like VBScript, JavaScript is an interpreted language, parsed line by line each time it's run. Make sure to test with older browsers on older machines if you're planning to deploy significant JavaScript in your application.

An efficient use of JavaScript is to perform form validations on the client-side, rather than making a round-trip back to the web server. It's very easy and lightweight to implement JavaScript alert windows that prompt the user for required fields. As much as possible, keep presentation-related validation in the presentation tier, on the client-side in the web browser.

5. Additional enhancements

Although the following VBScript tuning enhancements were found to have an insignificant impact on benchmarked applications, they are listed here for completeness:

- Remove debug comments from code
- Make necessary comments small
- Don't comment the obvious
- Remove extra blank lines and tabs

Active Server Pages

Microsoft, along with developers in the industry, have come to realize that limiting ASP to a proper role within the overall architecture of Windows DNA is a key to deploying scalable applications. ASP should be viewed as the "glue" that pieces together compiled COM components with presentation HTML. Using it improperly can significantly affect a system's scalability and performance.

On the horizon within the new Microsoft.NET platform is Active Server Pages+. This new release claims to offer significant performance improvements because ASP+ code will be compiled, not interpreted. Keep an eye out for this new technology to be released in the coming quarters. While still developing in the ASP environment, consider the following optimizations:

1. Minimize the number of calls from ASP to mid-tier objects

Rather than calling several methods to get data required to build an HTML page, mid-tier calls should be consolidated as much as possible. Consider the following code:

```
<%
Dim rsPersonData, rsAddressData, rsPhoneData, rsEmailData
Dim sPersonID
Dim oPerson, oAddress, oPhone, oEmail

sPersonID = Request.QueryString("PersonID")
```

```
Set oPerson = Server.CreateObject("Person.Person")
Set oAddress = Server.CreateObject("Address.Address")
Set oPhone = Server.CreateObject("Phone.Phone")
Set oEmail = Server.CreateObject("Email.Email")

rsPersonData = oPerson.GetPersonInfo(sPersonID)
rsAddressData = oAddress.GetStreetAddress(sPersonID)
rsPhoneData = oPhone.GetPhoneNumber(sPersonID)
rsEmailData = oEmail.GetEmailAccounts(sPersonID)
%>
```

This code snippet makes four calls to middle tier business objects to get data to populate a single HTML form on which the demographic data for a person in an application is maintained. This is a bad design for several reasons: more overhead is required for the multiple object instantiations, multiple calls are being made between IIS/ASP and the COM component services, and multiple calls are being made between the COM component services and the data services.

A better approach would be to make a single call to the mid-tier:

```
<%
Dim rsPersonData, rsAddressData, rsPhoneData, rsEmailData
Dim sPersonID
Dim oPerson

sPersonID = Request.QueryString("PersonID")
Set oPerson = Server.CreateObject("Person.Person")

oPerson.GetAllPersonInfo rsPersonData, _
                         rsAddressData, _
                         rsPhoneData, _
                         rsEmailData
%>
```

A single call is now made between IIS/ASP and the COM component. Additionally, the underlying data structures (Person table, separate Address, Phone, and Email tables) are now hidden from the ASP developer. Developers should only be concerned with the object model, not the intricacies of the data model.

Accomplishing optimizations of this type requires close communication between ASP and COM developers. It also highlights the importance of proper object model design.

2. Always Set objects = Nothing

Whenever instantiating middle-tier COM objects within VBScript, always explicitly destroy them before ending the ASP page. Don't rely on the IIS garbage collection mechanisms to do this work for you. In addition to being better style, objects are released more quickly into the COM+ resource pool.

3. Employ caching

The ASP environment offers several different caching mechanisms that can be used to speed up page processing. It's much quicker to get a piece of data from memory than it is to query a database or read from a disk drive.

The first caching mechanism is made available through the global.asa file. ASP provides an Application and a Session object where data can be stored for future reference. As a general rule, it's wise to avoid use of the Session object altogether. Heavy use of Sessions can limit scalability because memory must be allocated for each user that hits a site.

Use of the Application object, however, is strongly encouraged. Rather than calling a mid-tier COM component (and querying the database) each time a lookup value is required, call the component once in the global.asa's Application_OnStart event:

```
<SCRIPT LANGUAGE=VBScript RUNAT=Server>
Sub Application_OnStart
      Dim oLookup
      Set oLookup = Server.CreateObject("Lookup.Lookup")
      Application("varStates") = oLookup.GetStateList
      Application("varCountries") = oLookup.GetCountryList
      Set oLookup = Nothing
End Sub
</SCRIPT>
```

Rather than having to instantiate the oLookup object and make a method call on every page that needs one of the lookup variant arrays, the data is simply pulled from memory:

```
<%OPTION EXPLICIT

Dim varStates
varStates = Application("varStates")
%>
```

Better yet, if the HTML wrapped around the data will be the same throughout the site for any item stored in the Application object, transform the data into HTML text in the global.asa and store the HTML instead of the data itself. Then a Response.Write can be used with the Application variable, instead of transforming the data each time it's needed.

There are a few caveats for proper use of the Application object. Refrain from storing objects in Application space. Only objects whose threading model has been marked as free-threaded are safe. This eliminates storing any object written in VB, since VB components can only be apartment-threaded. Also, avoid use of the Application.Lock and Application.Unlock methods. Only one user at a time can be writing to the object if it's locked, and blocking can occur, greatly hampering scalability. Instead, only write to the Application object in the global.asa, and use it as read-only throughout the site.

A second type of caching available in ASP is at the page level. This is implemented by using the Response.Expires method with a value greater than 0:

```
<%OPTION EXPLICIT
Response.Expires = 30
…
%>
```

The line above instructs the user's browser to cache the page for 30 minutes. If the user returns to the page within the 30 minute timeframe, the browser will load the page from its cache, rather than requiring the web server to regenerate the page. This offloads

processing from the server to the client. You'll need to choose an appropriate timeout value for each page based on its functionality.

4. Avoid using ASP transactions

Although ASP provides developers with the ability to enlist transactions at the page level, this technique should be avoided for web sites that must scale and perform at a high level. The transactional features within COM+ and SQL Server provide a more robust transaction environment.

5. Put ADO code in the middle tier

If a web site must scale to any significant level, ADO data access code should be placed within compiled COM components. A significant advantage is the ability to create a compiled logical data access layer. This provides a single place where ADO settings can be tuned for optimum performance.

6. Avoid switching in and out of the <% %> delimiters

Whenever feasible, group together HTML and ASP output into Response.Write statements. This involves removing many of the ASP <% %> delimiter tags, relieving IIS of the burden of continually jumping in and out of the ASP scripting engine Consider the following code fragment:

```
<%
Dim i, a
For i = 1 to 1000 %>
      This is sample HTML.<BR>
      <% a = i %>
      The counter is at #<%=a%>.<BR>
      <%
Next
%>
```

This can be much more efficiently written as follows:

```
<%
Dim i, a
For i = 1 to 1000
      Response.Write "This is sample HTML.<BR>"
      a = i
      Response.Write "The counter is at #" & a & ".<BR>"
Next
%>
```

The optimized code has a single set of delimiters, enhancing performance because of the reduced switching. Users will notice absolutely no difference in the completed page.

7. Use Request.ServerVariables sparingly

The Request.ServerVariables collection has a total of 50 items. Any time you reference a single item in this collection, all 50 items are collected. This makes referencing a single item a relatively expensive operation. Therefore, avoid excessively referring to items within Request.ServerVariables.

8. Assign Request.xxx("VarName") to a local variable

For any of the Request collections (Form, QueryString, Cookies, ServerVariables) , if referring to an item in a collection more than once, dimension a local variable, assign the item to the variable a single time, and then refer to this variable multiple times.

For example, instead of doing this:

```
<%
if InStr(1, Request.ServerVariables("HTTP_USER_AGENT"), "Windows NT") >
1 then
      Response.Write "You are using a Windows NT PC."
elseif InStr(1, Request.ServerVariables("HTTP_USER_AGENT"), "Windows
98") > 1 then
      Response.Write "You are using a Windows 98 PC."
elseif InStr(1, Request.ServerVariables("HTTP_USER_AGENT"), "Windows
95") > 1 then
      Response.Write "You are using a Windows 95 PC."
elseif InStr(1, Request.ServerVariables("HTTP_USER_AGENT"), "Mac") > 1
then
      Response.Write "You are using a Macintosh."
else
      Response.Write "I'm not quite sure what you're using."
end if
%>
```

do this:

```
<%
Dim sBrowser
sBrowser = Request.ServerVariables("HTTP_USER_AGENT")

if InStr(1, sBrowser, "Windows NT") > 1 then
      Response.Write "You are using a Windows NT PC."
elseif InStr(1, sBrowser, "Windows 98") > 1 then
      Response.Write "You are using a Windows 98 PC."
elseif InStr(1, sBrowser, "Windows 95") > 1 then
      Response.Write "You are using a Windows 95 PC."
elseif InStr(1, sBrowser, "Mac") > 1 then
      Response.Write "You are using a Macintosh."
else
      Response.Write "I'm not quite sure what you're using."
end if
%>
```

9. Explicitly specify the Request collection (if known)

A handy feature of the Request collection is the ability to not have to explicitly specify the collection. For example, if the line Request("txtState") is written, ASP will look through all the Request collections until it finds an item with the name "txtState". However, if you know that the current page is a result of a form post, explicitly specify the Form collection so that ASP doesn't have to do the extra work of looping through the various collections: Request.Form("txtState").

10. Use <%OPTION EXPLICIT%>

Always specify the <%OPTION EXPLICIT%> tag at the top of all of your ASP pages. In addition to catching errors, this will force you to explicitly dimension your variables. Doing this prevents the scripting engine from having to check whether or not a variable has been allocated its memory every time it's referenced.

Additionally, omit the <%@ Language=VBScript%> statement that's automatically placed at the top of your file. The vast majority of ASP server-side scripting work is done in VBScript. This is a redundant line because IIS comes configured with VBScript as the server-side scripting language.

Although Visual InterDev doesn't provide developers with a way to automatically add the <%OPTION EXPLICIT%> tag like Visual Basic, the default template from which new ASP pages are copied can be modified to mimic this functionality. The template can also be configured so that the <%@ Language=VBScript%> statement is removed each time a new page is added. To configure the template, open the file called "New ASP Page.asp" in the C:\Program Files\Microsoft Visual Studio\VIntDev98\ Templates\Web Project Items directory. Also consider removing the gratuitous <META NAME="GENERATOR" Content="Microsoft Visual Studio 6.0"> tag. This is just additional overhead that will be sent over the wire.

HTML / XML

1. Reduce the data going over the wire

One simple way of increasing the performance of an Internet-based application is to reduce the size of the HTML pages that get sent from the Web server to the browser over the Internet. Several techniques can be used:

- HTML comment only where absolutely needed
- set attributes at the highest level
- use HTML tag defaults
- remove the <META "GENERATOR" …> tag
- employ Cascading Style Sheets (CSS)
- remove unneeded quotes

The following code snippet contains a lot of unnecessary text, especially within the For loop:

```
<%
Dim i
For i = 63000 to 63999 %>

<table cellpadding="0" cellspacing="0" border="0">
  <tr>
    <!-- Zip Code -->
    <td bgcolor="#cccccc" align="left" width="80"><font
face="arial" color="blue"
size="4"><strong><%=i%></strong></font></td>

    <!-- Location of the Post Office (city or town) -->
    <td bgcolor="#cccccc" align="right" width="200"><font
face="arial" color="blue" size="4"><strong>Zip Code
Location</strong></font></td>

    <!-- Text box for the user to enter a custom comment -->
    <td bgcolor="#cccccc" align="right" width="300"><input
type="text" name="txtComment<%=i%>" value="" size="30"></td>
  </tr>
</table>
```

```
<%Next%>
```

We can optimize this code in several ways, greatly reducing the amount of characters sent down to the browser:

```
<style type="text/css">
  .Zip {font-size: 18px; font-weight:bold; color:blue; font-
family:arial}
</style>

<%
Dim i
For i = 63000 to 63999 %>

<table cellpadding=0 cellspacing=0>
  <tr bgcolor=#cccccc align=right>
    <td class=Zip align=left width=80><%=i%></td>
    <td class=Zip width=200>Zip Code Location</td>
    <td width=300><input name=txtComment<%=i%> size=30></td>
  </tr>
</table>
<%Next%>
```

Notice that the border=0 attribute has been removed from the <table> tag. This is the default attribute for IE5, as is the type=text attribute for the <input> tag, which has also been removed. A custom CSS style has been defined and applied to each of the <td> tags. The bgcolor and align attributes have been moved from each of the <td> tags to the higher <tr> tag, and all HTML comments have been removed. Looping for 1000 zip codes, the size of this file has been reduced from 580MB to 250MB. This is a significant reduction in size which users surfing the Net over a 14.4KB modem will greatly appreciate.

2. Employ early rendering

Does your application have pages that are largely static in nature? If so, are you making a middle tier COM call and querying the database for the data to build the page each time? Why not make the call once and save the generated HTML, then present that to the user? It's obvious that much less processing is required to push a static HTML file down to the client than it is to generate an ASP with a database call.

The simplest method of early rendering is to browse to the page you wish to early render, select View\Source, then save the source as an .htm file. Import this file into the InterDev project and change the referencing links to point to this .htm file instead of the original .asp file. The page should now load much quicker. Repeat the process when the underlying data in the table changes.

Third party products are available which automate this process. One such product is XBuilder (http://www.xbuilder.net). In addition to providing a full-featured, configurable early-render development environment, XBuilder further compresses the generated .htm files by removing all white space characters such as tabs and carriage returns.

3. Explore image alternatives

Browsers today have enough imbedded functionality to provide snazzy looking web pages without having to download large image files to the client, taking up valuable bandwidth resources. Just experimenting with font sizes, table background colors, and single-pixel spacing .gif images for tables can provide a high-tech look and feel that's fast to render in the browser.

An additional area to explore is file type and compression rates for .gif and .jpg files. Knowing when to use a .gif file (best for images with large areas of solid colors and if a transparent background is needed) and when to use a .jpg file (best for photographs) is important. Also, experimenting with file compression (customized color palettes) using a tool such as Microsoft Image Composer or Adobe Photoshop also reduces file size without losing much quality.

4. Cache XSL files

IE5 provides an enhanced XML parser which can retrieve different views of data through XSL templates without having to retrieve the XML data from the server each time. Also, if there are a limited number of XSL files on the server, consider caching them as text strings in the Application object.

Microsoft has released the preview look at the new XML parser, called MSXML3. This parser will scale efficiently, whereas the current release of the XML parser had problems scaling. If a key component in your architecture will not scale, you have a performance bottleneck.

Microsoft is committing tremendous resources to integrate XML into all of its development, desktop, and server products. The .NET platform is based largely on XML, and performance improvements will continue to be integrated as this next-generation development platform is rolled out.

II. Business Services

Business services in the Windows DNA platform can be built using the following technologies:

- Visual Basic
- COM+ (MTS)
- ADO

Although other languages such as C++ or J++ can be used to build COM components and other data access technologies such as ODBC are available for data retrieval, the majority of requirements can be met by implementing solutions using VB and ADO.

Microsoft's new C# language along with the entire .NET framework bears close watching as it unfolds, promising significant performance enhancements along with other features such as cross-language inheritance for building middle tier components.

Additionally, the .NET platform will enable building Web Services, which can be consumed through the SOAP protocol by clients over the Internet.

Visual Basic

1. Design efficient classes

An efficient object model design comes close behind a solid data schema design in importance for a highly performing application. Consider which methods will be needed based on screen prototypes. When possible, try to limit the number of method calls per HTML page, preferably to a single call. Group methods appropriately, either by function or by object.

Publish your object model for ASP developers to reference as they implement HTML screens. Changes in the model further down the road can be expensive, so take time to implement a good design up front by soliciting feedback from team members.

2. Avoid maintaining state

A second tenet of good object model design in the Windows DNA architecture is being stateless. Web applications are by nature stateless – a request is made, a server fulfills the request, and then the connection between client and server is closed. Similarly, connections between browser and VB component should not be kept alive in Win DNA applications. Objects shouldn't be required to remember anything about clients between method calls. Call the object, do what's needed, release resources, and close the connection.

3. Early bind when possible

In addition to providing Intellisense through reading an object's type library, early binding lets the compiler resolve references, rather than references being resolved at run time. So instead of doing this:

```
Dim oBusinessObj as Object
Set oBusinessObj = CreateObject("BusinessObj.Customer")
```

do this if you know what class you want to instantiate when writing the code:

```
Dim oBusinessObj as BusinessObj.Customer
Set oBusinessObj = CreateObject("BusinessObj.Customer")
```

The majority of the time you'll know what class you're interested in, so make sure to early bind. You should notice the performance improvement immediately.

4. Avoid string concatenations

A string concatenation is an expensive function in Visual Basic because of the way VB allocates memory for each piece being concatenated. In the new .NET common language runtime, the string has been completely rewritten as an object. Hopefully Microsoft considered concatenation performance when designing the new string implementation.

For now, use the Len and Mid functions to optimize concatenations within VB. (Note that this technique does not work within ASP/VBScript). The key is to dimension a

variable as a string and make it a fixed length using the Space function. Next, determine the length of each piece to be concatenated and use the Mid function on the left hand side of the operator to insert the new string fragment into the large string:

```
Public Function TestWithStringReplacement() As String

    Dim sHTML As String
    sHTML = Space(256000)
    Dim lStartPos As Long
    Dim lLen As Long

    lStartPos = 1

    For i = 1 To 2500

        lLen = Len("This is code written in VB!!! " & i & "<br>")
        Mid(sHTML, lStartPos, lLen) = "Code from VB!!! " & i & "<br>"
        lStartPos = lStartPos + lLen
    Next i

    TestWithStringReplacement = RTrim(sHTML)
End Function
```

Instead of continually allocating memory each time a concatenation is performed using the "&" character, you're now simply dropping fragments one after the other into a very large string.

Note the call to RTrim as the return value of the function is assigned in the final statement in the function. This removes any leftover space hanging on at the end of the large 256000 size string defined at the beginning of the function. The 256000 number is a guesstimate – this number will never be exceeded for this function, so it's a safe upper bound.

5. Use ByRef only when needed

We calling COM components from ASP, only those parameters which will be changed by the method call should be defined as ByRef. Although ByRef is the default if omitted, ByVal should be used for two reasons. The first is that including this keyword prevents overwriting a variable that shouldn't be touched within the method. Secondly and of more importance to performance is that ByVal only needs to marshal the data to the object, not from it. ByRef must marshal the data both to and from the object. Passing data bi-directionally is a more expensive process.

6. Set Unattended Execution

When developing middle tier COM components with Visual Basic, go to Project Properties and check the Unattended Execution checkbox at the lower left. This will write any unhandled exceptions to the Event Log instead of raising them to the screen. This behavior is desired because no one will be sitting at each Web/COM server watching for message boxes to appear.

With this functionality enabled, administrators have the responsibility to regularly check for any problems in the Event Log on each mid-tier box. Developers can then be alerted

if any problems arise with the COM components, allowing them to tune and tweak for better stability. This enables more uptime and also a higher performing application.

7. Choose the appropriate data transport mechanism

When designing each method, the means by which data is remoted from VB/COM back to ASP/IIS must be carefully considered. There are several options to choose from:

- ADO Recordsets
- XML strings
- Variant arrays
- Function parameters/return values
- Property bags
- ByVal collections
- Delimited strings

The tradeoff important to performance is overhead vs. functionality. The more functionality provided by the transport mechanism, the more overhead and thus the slower performing it will be. ADO Recordsets have sophisticated functionality like sorting and filtering, but they are slower than variant arrays which have much less functionality.

Remember that you can specify the data transport mechanism for each method. Make the choice based on the business requirement and screen functionality that each method is serving.

8. Consider VB's Advanced Optimizations

If you need to get that last bit of performance from your VB COM components, there are a few extra settings you can flip. They're a bit hidden within the VB IDE, and for good reason (go to Project Properties, click the Compile tab, then click the Advanced Optimizations… button). By checking any of these settings, you're telling the compiler to disregard certain safeguards that are usually in place. Compile-time errors that would normally be detected and reported to you will not be displayed. Make sure you've tested and debugged the areas which these optimizations cover before you decide to turn them on in production.

9. Keep up to date on Visual Basic .NET

Microsoft has completely rewritten its underlying runtime to be shared across all development languages. Called the .NET Framework, this architecture promises significant performance improvement for middle tier components written in VB. Although Microsoft has not announced a timeline for the release of this next-generation development tool suite, beta testing has begun as of the writing of this paper (August 2000). Visit http://msdn.microsoft.com/vstudio/nextgen to learn more about the upcoming release of Visual Basic and the rest of the Visual Studio tools.

COM+ (MTS)

1. Design efficient COM+ applications

Once you've implemented your objects in Visual Basic, if they're designed to be configured they must be dropped into the COM+ run-time environment. Close attention should be paid to the grouping of DLLs into COM+ applications (MTS packages in Windows NT 4).

DLLs that call methods within one another should be grouped together within a single application. This will minimize the amount of cross-process marshaling which can be an expensive operation. Any time data is passed across a process boundary, a proxy/stub layer is created between the two processes. Passing data across process boundaries through a proxy/stub layer can slow down performance by 1000 times, so it should be avoided whenever possible.

Additionally, an appropriate level of component granularity should be implemented. If you have more than 20 or 30 DLLs within a single COM+ application, consider breaking these out into two or more applications.

2. Server vs. Library applications

Whenever possible, make your COM+ applications library applications. To do this, right-click on the application and select Properties, then on the Activation tab select Library application.

Library applications provide much better performance because they run in the process space of the web server. However, a very important tradeoff is that if they crash, they will also crash the web server. Because of this risk, DLLs should be thoroughly tested before deploying library applications to your production web servers.

In addition to stability benefits, server applications let you set security within the Component Services Explorer. Library applications can only run under the security context of the anonymous IIS user (IUSR_machinename).

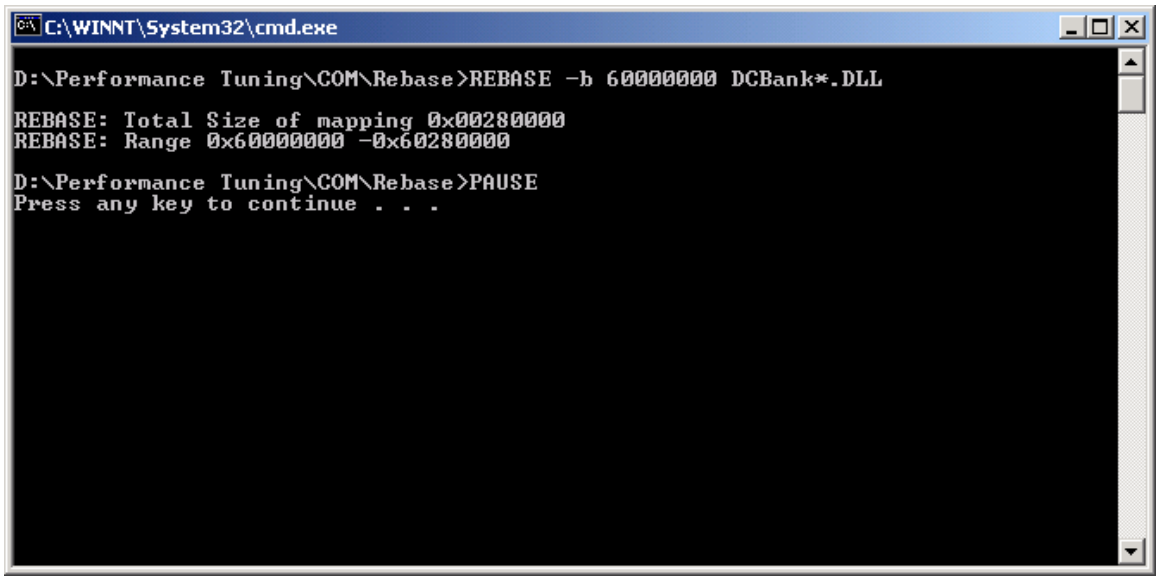
3. Explore the REBASE utility

A utility that C++ developers will be familiar with is the Rebase utility. This tool is available in the C:\Program Files\Microsoft Visual Studio\VC98\Bin directory if you've installed C++. It's helpful because it provides a simple way to create unique DLL base addresses for all the binaries in a project.

All COM components you develop in VB are compiled with the same default base address, #11000000. It's helpful to provide a unique base address (a virtual memory address from #00000000 to #FFFFFFFF) for each component so that the operating system loader does not have to perform the overhead of finding a unique address in memory for each DLL.

Although Visual Basic provides a mechanism to assign unique base addresses through the Project Properties Compile tab, if you're working on an enterprise-wide distributed application with many VB projects, the task of managing all the base addresses from within VB quickly becomes unwieldy. Simply tell Rebase the virtual memory address at

which to start (#60000000 and above), give it a list of all the DLLs you want to rebase, then hit enter:



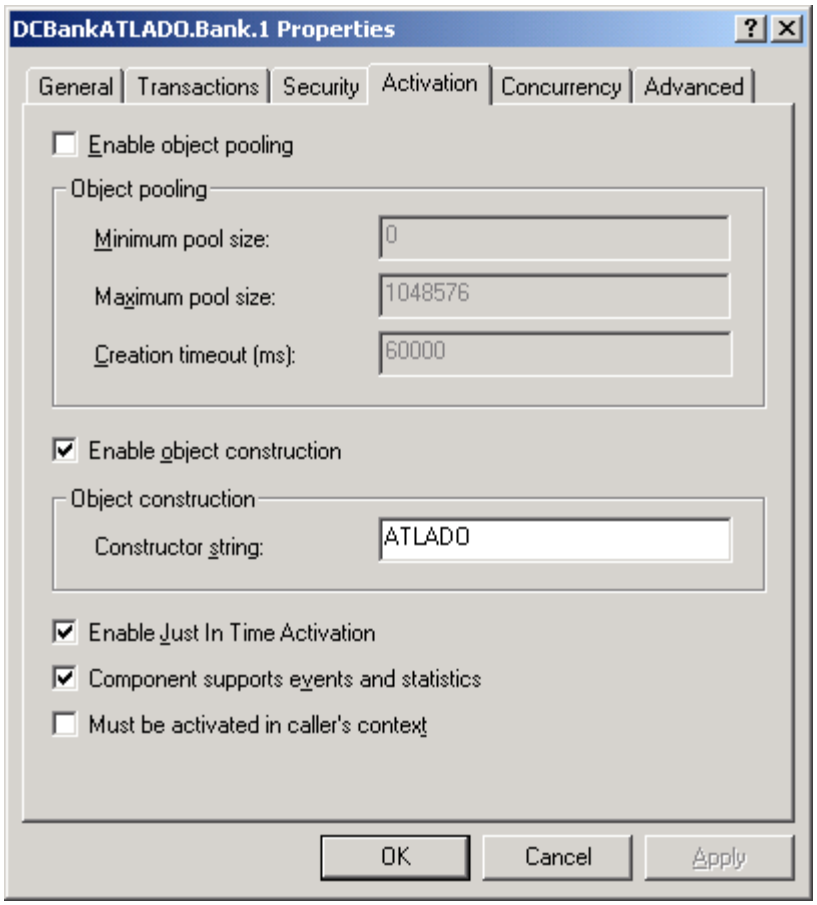
In the screen above a Rebase is performed of all the DLLs in the current directory that begin with DCBank (sample DLLs taken from the Windows DNA Performance Kit discussed later). Notice that Rebase reports the total size of the DLLs (#00280000 for about 25 physical DLLs in this case), along with the new memory addresses assigned as the base address for the DLLs. With this single command, unique base addresses have been assigned to 25 DLLs very quickly. Think how much longer it would have taken to do this within the VB development environment.

Note that Rebase actually changes the DLL binary images, so if you have DLLs under source code control, they must be checked out before the Rebase utility can be successfully run.

4. Employ the Shared Property Manager (SPM)

COM+ provides a storage area in memory called the Shared Property Manager where small amounts of state data can be stored. This is an ideal location for storing such things as a database connection string. Instead of reading the string from the registry every time the database is accessed, a check is performed to see if the string exists in the SPM. If it does, it will be read from the SPM, otherwise it can be read a single time from the registry and placed in the SPM. Reading from memory is always much faster than reading from disk.

COM+ also makes available a Constructor string for each component registered in the runtime:



Any string can be placed in this location at configuration time and can be read and written to from code. This is another good place to put a database connection string rather than reading from the registry.

5. Optimize the default DTC logging configuration

By default, the Distributed Transaction Coordinator (DTC) for each server on which COM components run performs its own logging. If you have a web farm and if you've configured your database to run on a different machine, you can optimize this default configuration by having only the database machine perform DTC logging. On each Web/COM server, stop the DTC service and configure the DTC to log to the remote SQL Server machine through Control Panel.

Having a single machine perform DTC logging is more efficient because there will be less overall disk access. (Additionally, if you're accessing a single database server, consider placing the transaction logic in stored procedures rather than in COM components).

ADO

1. Minimize calls from the mid-tier to the data tier

Similar to how calls from ASP to COM methods should be minimized, the number of calls from COM components into the database should be minimized. Usually the

database will be located on a different physical machine from the Web/COM servers, so all calls will carry the expense of crossing the network.
One way to reduce the number of calls is to return a recordset of recordsets. Instead of executing a stored procedure for each recordset needed, create a controller stored procedure that executes all the stored procedures using the T/SQL EXEC command. When the data is returned to the COM method, iterate through the recordsets using the ADO NextRecordset command.

Watch out for calls made to the database within a middle tier loop. Combine parameters into fewer stored procedure calls if possible. This may mean less flexibility if requirements change, but the performance gains can be significant.

2. Make ADO free-threaded

If you're only using SQL Server in your application, change ADO's threading model to be free instead of the default of apartment. This will give a nice performance gain. Do this by running the C:\Program Files\Common Files\System\ado\makfre15.bat batch file. It simply changes the threading model in the registry for the five main ADO objects (Connection, Command, Recordset, Parameter, Error).

3. Call New for ADO objects

In Visual Basic you should always avoid the New operator when calling other VB classes and methods to make sure COM context wrappers are placed around the objects. VB has an optimization that will bypass the COM layer if the method being instantiated is in the same project. Avoid this by using the CreateObject and CreateInstance functions.

ADO is an exception to this. Because the ADO components are not configured components (not dropped into the COM+ run-time), it's safe to use the New operator. This gives a performance gain because the ProgID doesn't have to be resolved:

```
Dim rs as ADODB.Recordset
Dim cn as ADODB.Connection

' Calling ADO components; OK to use New instead of CreateObject
Set rs = New ADODB.Recordset
Set cn = New ADODB.Recordset
Dim oBusinessObject as BusinessObject.Customer

' Calling a VB component; cannot use New
Set oBusinessObject = CreateObject("BusinessObject.Customer")
```

4. Choose the lightest cursor/locking combination

The fastest cursor/locking choice available in ADO is the "firehose" cursor – forward only/read only. This is the default ADO type and should be used whenever possible. If you need to scroll through a recordset, you'll have to use a static cursor. Very rarely should you need a dynamic cursor. It's by far the most expensive type since it continually refreshes data.

Make sure you implement the proper type for each COM method. Often, code is cut and paste from one method to another and the cursor/locking type may not be the right type for the new method, providing more functionality than is actually required.

5. Close Connection and Recordset objects

Always explicitly close any ADO Connection and Recordset objects you've instantiated. This will prevent "connection creep" from occurring on the database, keeping resources available for other processes.

6. Reuse the connection within a method

If you're making several database calls within a single method, instantiate an ADO Connection object and use that object for each of the calls. Don't get a new connection each time a call is made. Even if you're using connection pooling, there is still overhead associated with getting a connection from the pool.

7. Use parameterized Command objects

Rather than using inline SQL with Recordset objects, instantiate ADO Command and Parameter objects to call stored procedures. This should be done even for simple SELECT statements. SQL Server is able to compile parameterized commands a single time and use that compilation each time the command is executed, even if parameters change. Inline SQL calls must be compiled each time they're executed, making SQL Server do extra work that could be avoided.

8. Ordinal positions are fast

The fastest way to reference a field in a recordset is by its ordinal position:

```
sCustomerID = rsCustomers(0).value
sCustomerName = rsCustomers(1).value
```

Although this is the fastest way to access the data, it's obviously not the most readable. A good trade-off is to define a set of constants at the top of the class, or to include these in a type library:

```
Const CUSTOMER_ID = 0
Const CUSTOMER_NAME = 1
…
sCustomerID = rsCustomers(CUSTOMER_ID).value
sCustomerName = rsCustomers(CUSTOMER_NAME).value
```

Ordinal positions are faster because a lookup doesn't have to be made on the field name to determine where the field is located in the recordset.

9. Employ the Fields object

If you're referencing a field in a recordset multiple times, instantiate an ADO Fields object and point it to the field in the recordset. Then reference this local variable multiple times:

```
Dim fldCustomerName As ADODB.Field
Set fldCustomerName = New ADODB.Field
Set fldCustomerName = rsCustomers(CUSTOMER_NAME).value

if Left(fldCustomerName, 1) = "A" then
      …
elseif Left(fldCustomerName, 1) = "B" then
      …
elseif Left(fldCustomerName, 1) = "C" then
      …
…
```

```
end if
```

This prevents the overhead of determining where the field is located in the recordset each time it's referenced.

10. Connection.Execute

Consider using the Connection.Execute method for lighter calls rather than calling Recordset.Open or Command.Execute. The Connection object will have less functionality, but it will also have less overhead, improving performance.

11. Stored procedure with output parameters

If you have a stored procedure that you know will return a single row, consider changing the procedure to return OUTPUT values rather than a full recordset. Use the ADO Command and Parameter objects to capture the values returned from the procedure. Again, this method saves time because the overhead associated with a Recordset object is avoided.

12. GetRows, GetString, and Save

Three methods that can provide additional performance enhancements are the ADO Recordset object's GetRows, GetString, and Save methods. GetRows is helpful for converting a recordset into an array. GetString is designed to convert a recordset into a string. The GetString method is especially useful for doing simple HTML transformations on recordsets:

```
<%
Dim sCustomerData
sCustomerData = rsCustomers.GetString(,, "</td><td>", _
      "</td><tr></tr></td>", " ")
%>
…
<table>
      <tr>
            <td>
            <%=sCustomerData%>
            </td>
      </tr>
</table>
```

The Save method, available with MDAC 2.5 and above, allows persisting recordset data as an IStream or an XML DOM document. This saves the overhead of instantiating an XML DOM document and manually performing the transformation.

13. adExecuteNoRecords and CacheSize

When executing an ADO Command or Connection object that will not return any records (UPDATE or DELETE queries), use the adExecuteNoRecords option:

```
Dim cmd as ADODB. Command
Set cmd = New ADODB.Command
…
cmd.Execute ,, adExecuteNoRecords
```

Performance will be better because the overhead associated with a data set will be avoided.

Another parameter to tune is the CacheSize property. By default this parameter is set to 1, but if you're scrolling through a recordset with 50 rows, set this property to 50 (if greater than 100 rows, set it to 100), and you'll be scrolling through the memory cache instead of retrieving from disk.

14. Use the Native OLEDB Provider for SQL Server

The fastest way to access data in a SQL Server database is to use the OLEDB Provider for SQL Server. This is faster than the OLEDB Provider for ODBC, which adds an additional layer between the ADO objects and the database.

15. Employ ODBC Connection and OLEDB Resource pooling

Employ connection or resource pooling whenever possible. When pooling is enabled, the system keeps a set of resources cached, ready for use, rather than taking the time to perform an instantiation each time a resource is needed. If you're able to implement either, choose object pooling over connection pooling – it generally provides faster performance.

Explore the MDAC Pooling Toolkit, located at:

http://msdn.microsoft.com/library/techart/pooling2.htm.

This document contains detailed information on how to properly configure pooling in Windows DNA applications. It also discusses setting up detailed performance monitor counters so that you can make sure pooling is operating efficiently.

III. Data Services

Data services in the Windows DNA platform are largely built around Microsoft's enterprise-level database system, SQL Server 7. When planning, implementing, and tuning an application database in SQL Server, the following three areas should be given special consideration:

- Schema Design
- Database Operations
- Disk Configuration

Schema Design

1. Normalization / Performance tradeoff

It's very important to keep performance considerations in mind when designing the database schema. As a general rule, avoid highly generalized models. Flexibility often comes at the price of performance. If performance is a key design goal, you'll need to get very specific with tables and fields. A proper balance between "deep" tables with fewer columns and "wide" tables with fewer rows should be reached, based on how data will be queried.

An OLTP system generally requires a more normalized schema, while an OLAP application tends to be more denormalized. An appropriate level of normalization and

denormalization in each scenario must be reached, however. An OLTP schema that's far too normalized will not perform well. Designing the right level in each scenario is more of an art than a science. Share your proposed design with more experienced DBAs and solicit feedback.

2. Choose appropriate primary keys

Keep primary keys short and avoid GUIDs unless absolutely necessary. GUIDs are appropriate in highly distributed applications where new row entries will be made at remote sites that will be replicated into a central data store. Employing GUIDs as primary keys prevents one remote site from overwriting another's data. However, GUIDs come with a performance penalty since they are 128-bit integers. A CHAR or INT column as a primary key will perform much better if a GUID is not really necessary.

3. Choose appropriate foreign keys

When implementing foreign keys, refer to as few tables as possible. If updating or deleting rows in referenced tables, make sure the referencing columns are indexed. Otherwise the referencing table will be locked and a full table scan will be performed, an operation that can be very costly.

Database Operations

1. Tune indexes

Indexes speed up SELECT queries but slow down INSERT, UPDATE, and DELETE statements. The more selective an index is, the better. Composite indexes will only be used by queries if the leading column in the index is used in the query join or filter, so make sure to order the columns in indexes properly. It may be necessary to create additional indexes for a table with the same columns in a different order.

2. Use stored procedures

Stored procedures are pre-compiled and pre-optimized by the SQL Server engine, providing fast execution. Use them for performing database operations instead of inline SQL in COM methods.

Avoid dynamic SQL within stored procedures. This includes any statement within a stored procedure that's concatenated together:

```
CREATE PROCEDURE sp_GetAuthorsByNameDynamicSQL
      @LastName AS VARCHAR(50),
      @FirstName AS VARCHAR(50) = ''
AS

DECLARE @SQL NVARCHAR(2048)

SET @SQL =
      N'SELECT
            au_id,
            au_lname,
            au_fname
      FROM
            Authors
      WHERE
            au_lname = ''' + @LastName + ''''
```

```
IF @FirstName <> ''
      SET @SQL = @SQL + N' AND au_fname = ''' + @FirstName + ''''

EXEC sp_executesql @SQL
```

Instead, perform conditional checks within the stored procedure, placing the statement to be executed within each check:

```
CREATE PROCEDURE sp_GetAuthorsByName
      @LastName AS VARCHAR(50),
      @FirstName AS VARCHAR(50) = ''
AS

IF @FirstName <> ''
      SELECT
            au_id,
            au_lname,
            au_fname
      FROM
            Authors
      WHERE
            au_lname = @LastName
            AND au_fname = @FirstName
ELSE
      SELECT
            au_id,
            au_lname,
            au_fname
      FROM
            Authors
      WHERE
            au_lname = @LastName
```

Also avoid functions on columns in a WHERE clause or in a JOIN. Using a function in this context will prevent an index from being used that would normally provide fast access. The NOT IN keyword should also be avoided.

3. Minimize cursor use

Sometimes work performed by a cursor can be done instead with a complex SQL statement using a GROUP BY, HAVING, or other clause. Less experienced T/SQL programmers may especially be tempted to use a cursor because of how familiar it feels to a simple FOR loop in other programming languages such as VB. However, cursors can quickly limit performance and scalability and should only be used when the work they perform cannot be done another way.

For the fastest cursor type in SQL Server, use the LOCAL and FAST_FORWARD keywords in the cursor declaration:

```
DECLARE CustomerCursor CURSOR LOCAL FAST_FORWARD FOR
      SELECT …
```

This defines a forward-only, read-only "firehose" cursor. (Note that FAST_FORWARD cannot be used with the SCROLL or FOR_UPDATE keywords). The LOCAL keyword defines the cursor in a limited scope, reducing the amount of memory required.

4. Keep transactions short

The database locks certain resources during a transaction to protect data integrity. The longer the duration of the transaction, the longer any related resources are locked. This can cause other database operations to block, waiting for resources to be freed. In any system with a moderate to high level of activity, this can cause significant performance degradations. Keep the work performed within a single transaction to a minimum to reduce blocking.

5. Relieve contention and deadlocks

Locks are placed on rows in the database each time a query is performed to make sure they're not updated in the middle of the query. Without such locking, skewed results would be returned, with the first half of the result set representing one state of the data and the second half another state because an update was performed by another user mid-query.

Because locks are so integral to database integrity, they are necessary for operation and cannot altogether be avoided. However, too much locking in a system can cause contention for resources. Performance can be hampered because queries may spend a good deal of time waiting on locks to be released rather than returning result sets.

SQL Server provides several ways to monitor the state of locking in a database. The sysprocesses table in the Master database can help track down processes that are blocked. SQL also exposes the Locks object to the NT Performance Monitor, providing several counters that can help identify high contention levels in the DB. If you think excessive locking may be causing contention in your application, locate the root cause and test solutions.

Reduce deadlocks by making sure that all transactions access SQL tables in the same order. This will prevent one transaction from locking table A and attempting to write to table B while another process has first read table B and is attempting to write to table A.

6. Update statistics and rebuild indexes

Statistics must be updated and indexes rebuilt for SQL to use the optimal execution plan for queries. INSERT, UPDATE, and DELETE statements continually change the demographics of the data, and updating statistics will scan the table with the new data, helping SQL Server pick the appropriate execution plan when the next query is run.

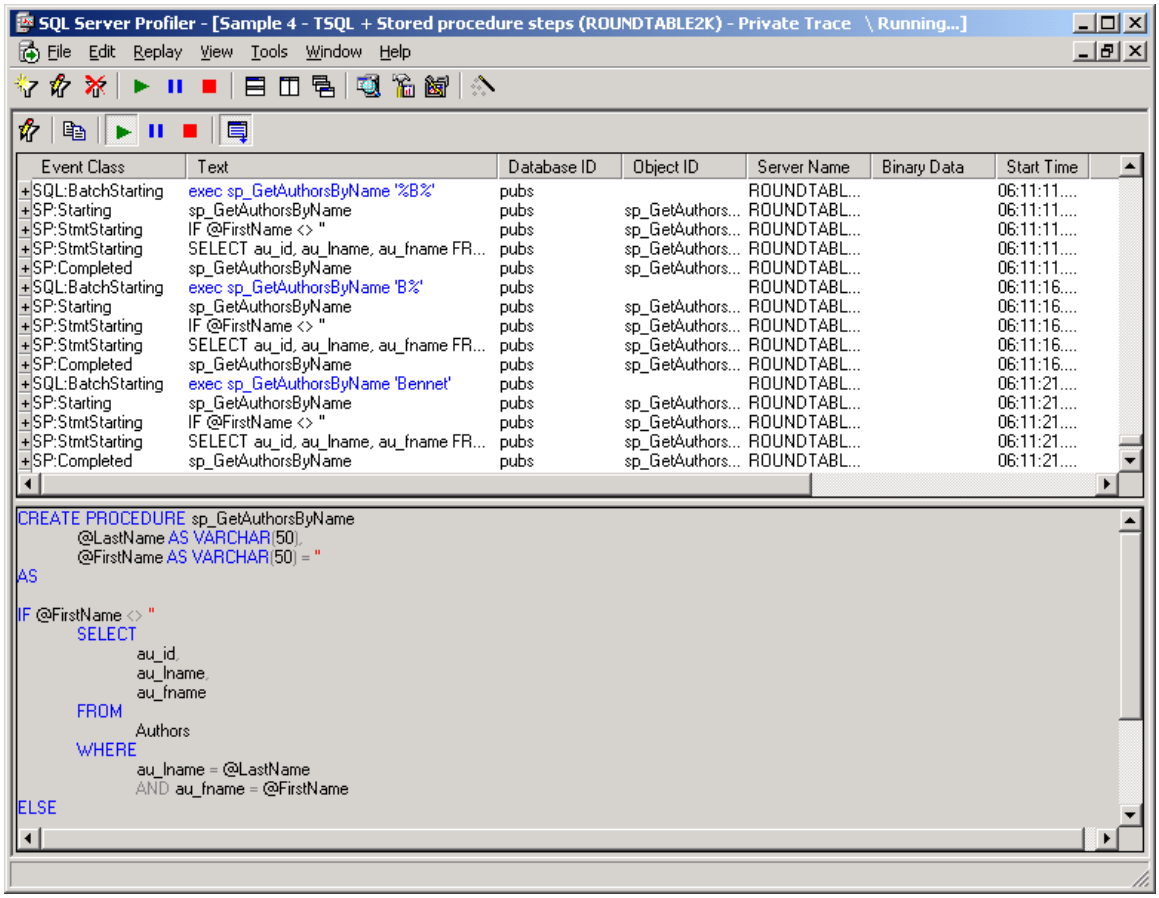
Any time significant data changes are made, the UPDATE STATISTICS and DBCC DBREINDEX statements should be run. If you have a highly dynamic application in which data is changing constantly, you may want schedule a job to run these commands during off-peak hours. Avoid running these statements during production hours, as they can take some time to execute on larger tables. Additionally, any cached execution plans will be invalidated, further slowing down database queries.

7. Identify poorly performing T/SQL

Use the SQL Profiler tool to determine which T/SQL statements are taking a long time to run. Profiler is a powerful tool, capturing events and filtering out unwanted data. It should primarily be run on a test system during load testing. However, depending on

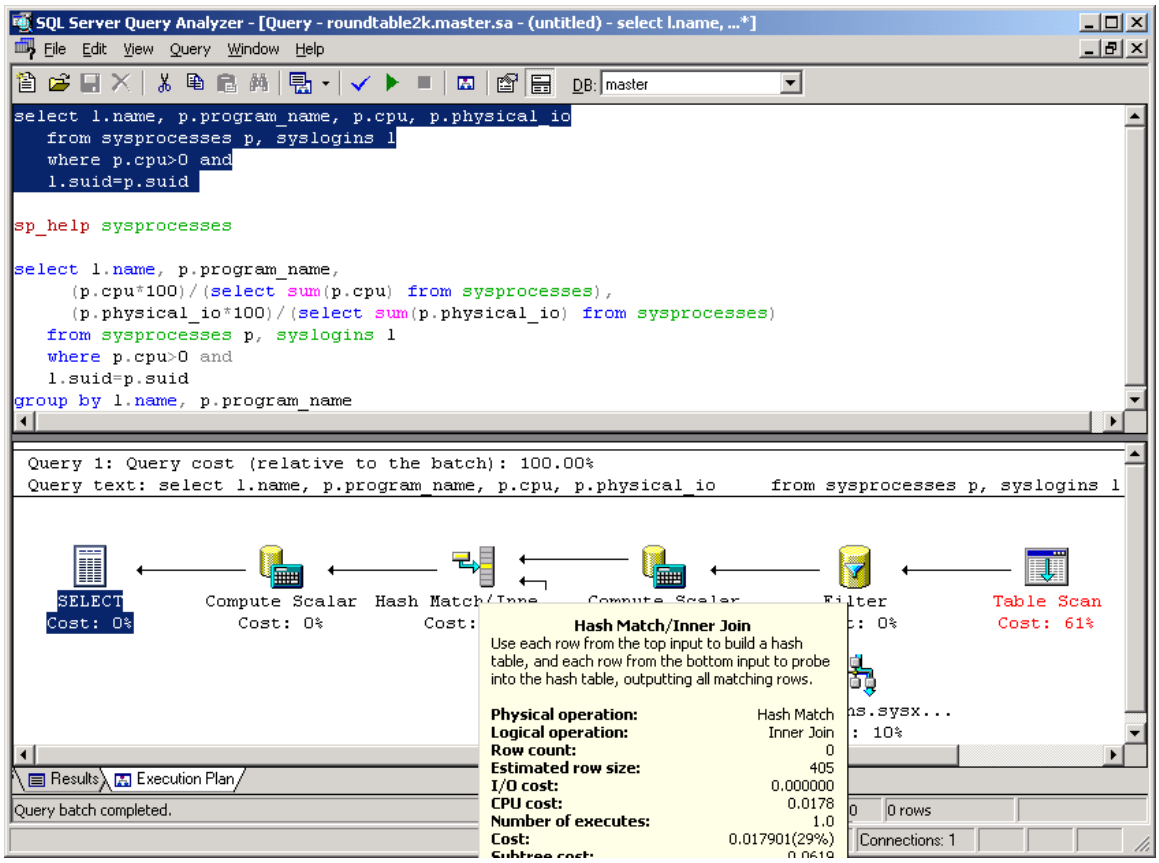
the loads experienced in production, you may choose to run Profiler on a limited basis on production SQL Server boxes to obtain more accurate performance metrics.

Profiler captures SQL statement execution by time slices, showing detailed timing metrics for each command:



Profiler also contains the Index Tuning Wizard. This tool can sometimes help optimize indexes on tables based on commands captured in the execution window. Although it may not always be selective enough for your needs, it can provide a good starting point.

Once you've identified the problem SQL statements using Profiler, SQL Query Analyzer includes a graphical tool called Show Execution Plan which provides details on how queries are executed:



Hovering over a step in the execution plan gives details such as a description of the logical operation performed and the argument of the SQL command that generated the step.

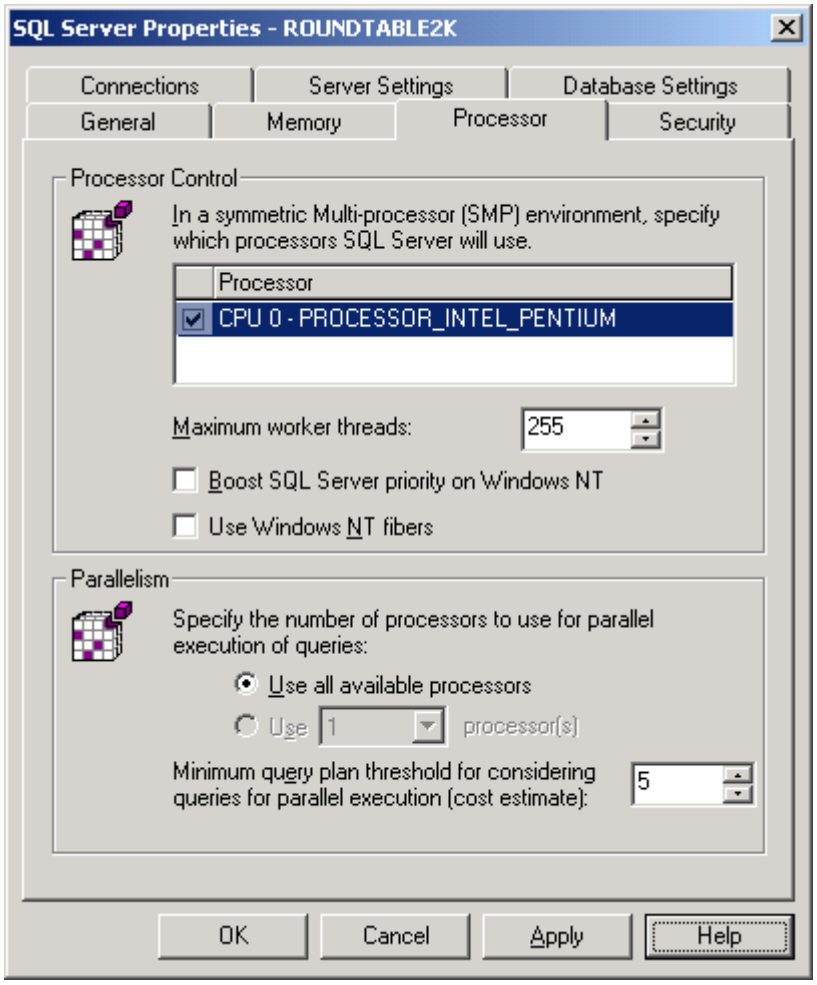
As you develop a feel for the why the problem T/SQL statements aren't performing as desired, rewrite the queries until performance is improved. Alternatively, tune the indexes on the tables that are being queried. As a last resort, optimizer hints such as locking hints, join hints, query hints, and table hints can be tested if you're still not getting the performance you desire.

8. Tweak SQL Server parameters

SQL Server is primarily a self-tuning database. However, certain self-tuning features can be problematic if they're triggered during production hours. The auto-update statistics and auto-grow DB file size features can cause negative consequences because they require significant resources and will slow down database operations.

Additionally, any time statistics are recalculated, each stored procedure must be recompiled, creating additional slow downs. For these reasons, consider turning off these self-tuning features on production systems. Schedule jobs to run during off-peak hours to prevent performance problems for your users.

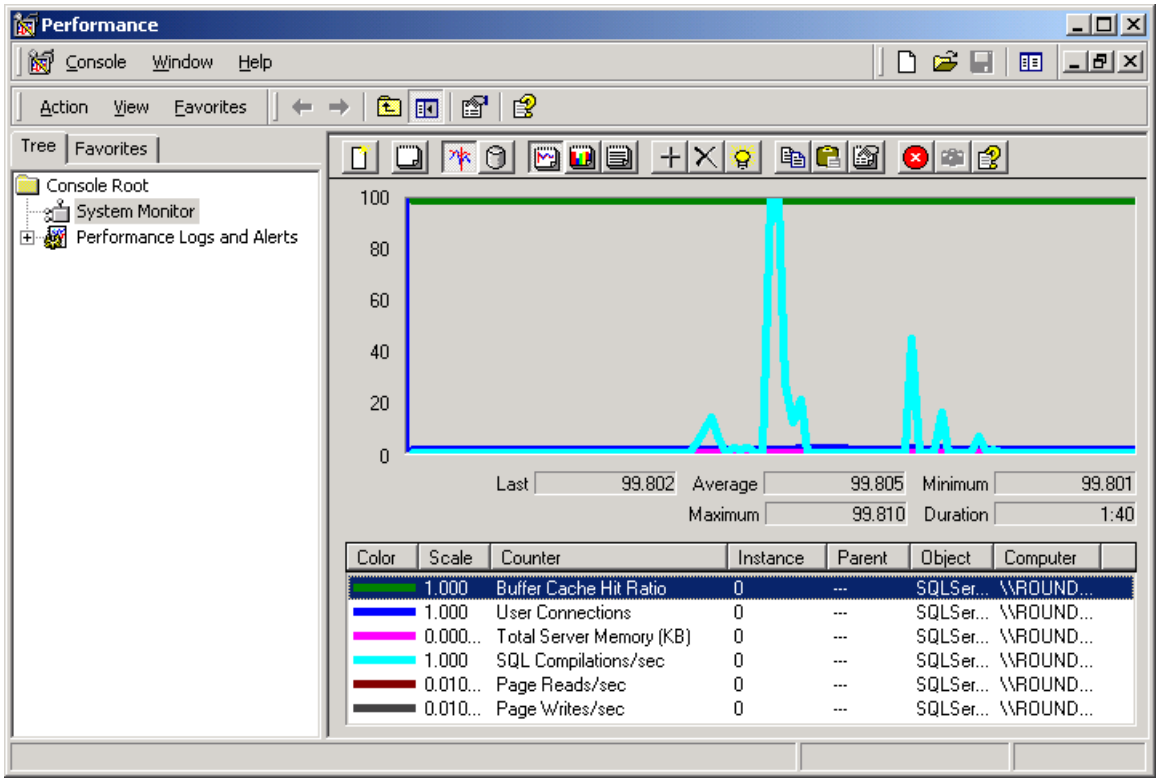
SQL allows configuration of a few other settings, accessed by selecting server properties:



Avoid checking the Boost SQL Server priority on Windows NT option. Although this will allow the SQL Server process to run at a higher priority in the operating system, if a stored procedure is coded improperly and enters an infinite loop, the box will have to be rebooted.

On a multi-CPU box, SQL can be restricted to run parallel queries on a specific number of processors in the Parallelism section. You may choose to leave one processor free solely for NT.

SQL Server also comes with a few Performance Monitor counters configured for frequently accessed monitoring. Selecting the Performance Monitor option from the Microsoft SQL Server 7.0 Program group opens PerfMon with the following counters:



Use these counters to monitor the general operation of your SQL databases. If you notice values exceeding normal ranges, use Profiler and Query Analyzer to track down problems T/SQL statements or configurations in order to maximize performance.

Disk Configuration

1. Identify I/O performance issues

Use Performance Monitor's physical disk counters to determine whether disk drives are a bottleneck in the system. If you see heavy writes and reads to and from a single disk, consider relocating tables and/or indexes to different drives. Separate out the different types of I/O (system swap file, transaction logs, data files, index files, tempDB) by placing them on different disk drives. This will help relieve "hot spots" by minimizing excessive disk activity on a single physical drive.

2. Cluster for availability, Data Partition for scalability

Clustering is an availability solution, NOT a scalability solution. A common misperception is that clustering provides scalability benefits. Once a SQL Server box has been scaled up to its maximum number of CPUs, data must be partitioned across multiple machines. Clustering should be implemented only for redundancy and risk management reasons, not for performance reasons.

IV. System Services

System services in the Windows DNA platform provide the underlying foundation on which developers build applications. Tunable pieces of this foundation include:

- Windows
- IIS

Windows

1. Application Server mode

For Web and SQL Server boxes, make sure to place Windows NT into application server mode. Application Server mode provides better SMP scalability, improves network throughput, and makes more memory available to applications.

Enable this mode by right-clicking on the My Computer icon, opening Properties, and clicking the Advanced tab. Click the Performance Options… button and change the radio button under the Application Response section to be optimized for background services. This gives priority to services such as the World Wide Web Publishing and MSSQLServer services.

In the same area, you can change the size of the paging file for virtual memory. Make enough space is allocated so that disk swapping is minimized.

Additionally, from the Start button, select Settings, Network and Dial-up Connections. Double-click the Local Area Connection, select the File and Printer Sharing for Microsoft Networks component, then click the Properties button. In the Optimization section, select the Maximize data throughput for network applications option.

2. Disable unused services

Unused Windows services can be disabled on application, Web, and SQL servers. These include Alerter, ClipBook, Plug and Play, and any others you may not need. These services take up memory and other resources that can better be used by your applications.

3. Monitor Event Logs

System administrators should check the Windows Event Log for problems at least every other day. This will help catch problems early, allowing developers to implement fixes before bugs become too damaging to the system and its users.

IIS

1. Internet Services Manager settings

The Internet Services Manager provides a graphical tool to change various IIS settings. On the Performance tab, move the slider bar to the far right to "More than 100,000" hits expected per day. On the ISAPI Filters tab, remove any ISAPI filters not in use.

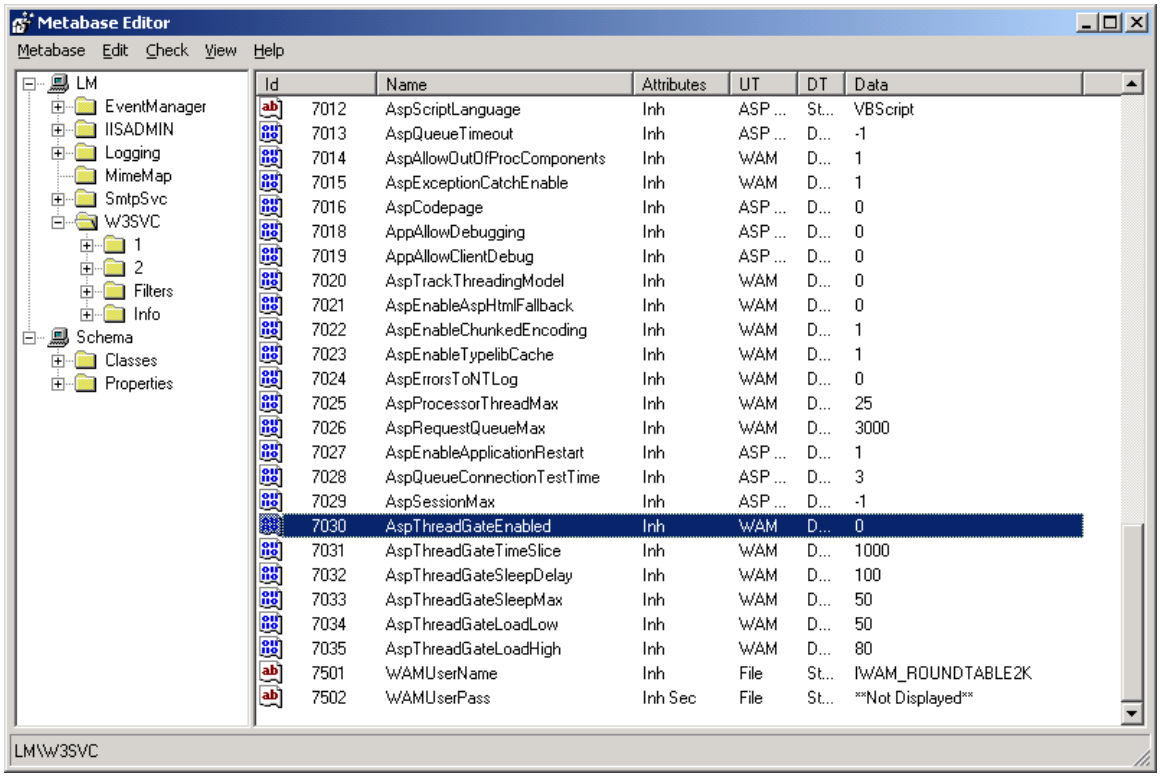
On the Home Directory tab, click the Configuration… button at the bottom right to access the Application Configuration dialog. Remove any unused application mappings from the list on the App Mappings tab (usually everything except .asa and .asp) . On the App Options tab, uncheck the Enable session state box and check the Enable buffering box. Buffering sends the entire ASP down in one stream after the dynamic pieces of the page have been built rather than sending down chunks a little at a time. Buffering HTML output can greatly improve overall performance of an application. On the App Debugging tab, uncheck the Enable ASP server-side script debugging. Debugging should never be enabled in a production environment.

On the Server Extension tab of the Web Site Properties dialog, set the appropriate performance setting in the Enable authoring section. On the Directory Security tab, click the Edit… button in the Anonymous access and authentication control section. Disable Integrated Windows authentication if your application does not require it, and select Basic Authentication instead. Remember only to use Secure Sockets Layer (SSL – https:// security) when needed, on a page-by-page basis. Don't use SSL on a global basis for an entire site.

Another important IIS setting is Process Isolation, with new options for IIS 5. On the Home Directory tab, a new drop-down box is available in the Application Settings section called Application Protection. You can set an IIS application to run either in-process with IIS (Low), in a pool of applications (Medium), or in its own process space (High). The default for all new applications is Medium. If you need optimum performance select Low, but keep in mind that in-process applications can crash the web server. If you need the utmost security and stability, select High.

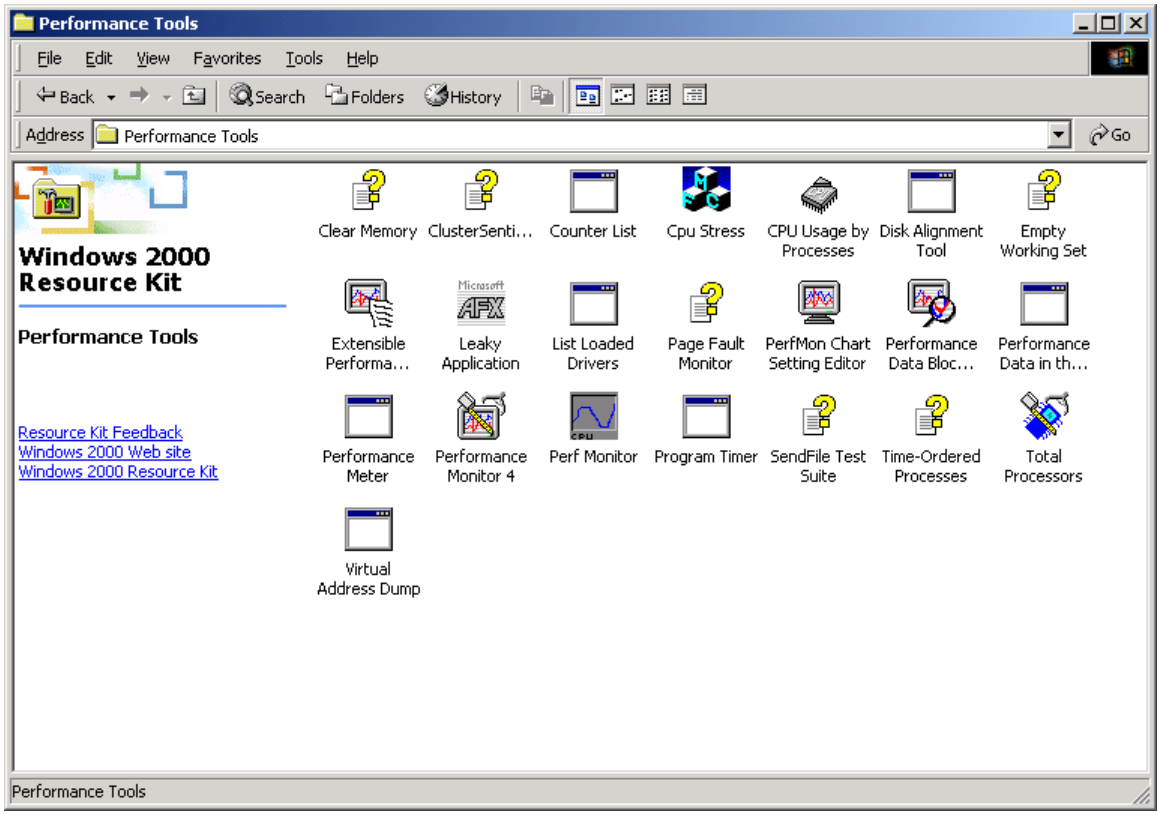
2. Additional Metabase settings

In addition to the settings which can be configured in Internet Services Manager, the Windows Resource Kit contains a tool called the Metabase Editor, very similar to the Registry Editor, which allows modification of even more IIS settings:



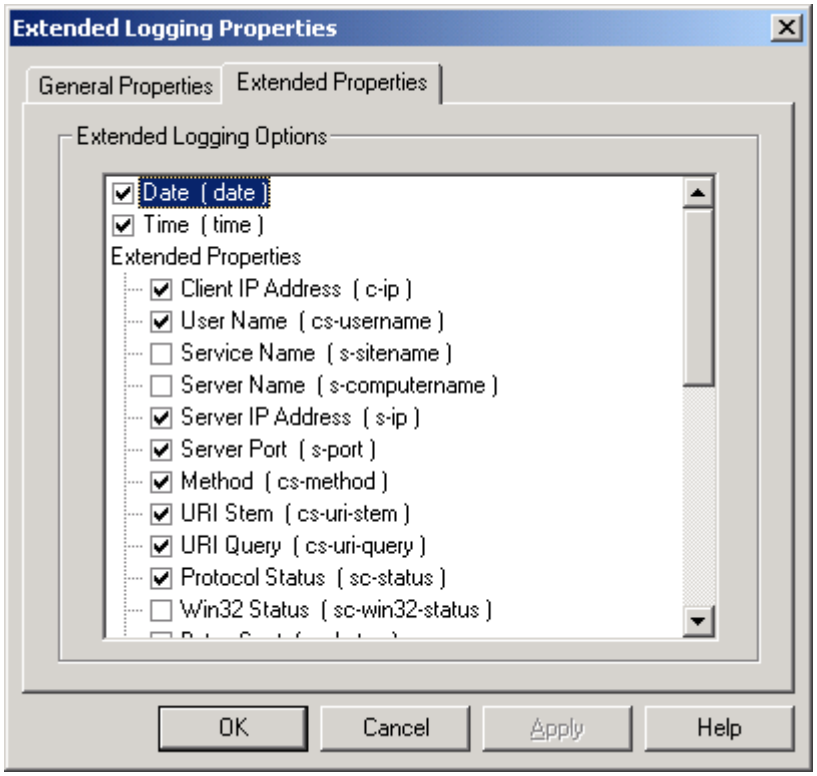
Use this tool to fine-tune IIS to provide even more performance gains.

The Windows Resource Kit contains several additional performance-related tools which are quite helpful for tuning Windows DNA applications:



3. Efficient logging

Log only what is needed from your web servers. The default configuration provided for logging in IIS may contain more fields than are needed, so be sure to uncheck the unneeded fields. Logging is configured on the Web Site tab of the Web Site Properties dialog. In the logging section at the bottom, click the Properties… button to tweak logging parameters. On the Extended Properties tab, uncheck any fields you don't need:



Remember that this data is being logged to disk, so the more fields that are checked, the more data that must be written to disk, taking up additional CPU time and disk space.

IIS 5 provides a new logging feature called Process Accounting. Turn this feature on to get details of what's happening on the web server at a machine/CPU level.

V. Physical Configuration

The proper physical configuration of a Windows DNA application is very important to its overall success. Proper attention must be given to each of the following physical components:

- RAM
- CPU
- Disk
- Network
- SMP

RAM

The more RAM, the better, especially for SQL Server machines. The more RAM a server has, the less swapping to hard disk the CPU has to do. Moving data in and out of RAM is much quicker than to and from disk. Since the cost of RAM continues to drop,

consider adding more memory as a first step in resolving bottlenecks. It's usually more cost-effective to install additional RAM than it is to have developers rewrite code.

CPU

Web servers can either be scaled "up" (adding more processors to the same box) or scaled "out" (adding more boxes). Opinions vary on which approach to take, but scaling out is usually more cost-effective. Four single-CPU machines are generally less expensive than a single machine with four CPUs. Plus, depending on the internal architecture of the hardware, if one processor goes down, the entire box may be unavailable. If you've scaled out and a box goes down, three others will still be online to handle requests. Although scaling out provides the additional benefit of reducing risk by improving redundancy, it requires either hardware or software in front of the web server farm to route requests to the various boxes.

Once a SQL Server has been scaled up to its CPU limit, data must be partitioned onto multiple machines. This is a much more complicated task than scaling out Web servers. You can either put certain databases on certain machines, or you can split database tables across machines, putting customers A-M on one machine, and N-Z on another. Either way, the middle-tier must become intelligent, knowing which SQL Server to query when a request is received. Only very large applications require data partitioning, so the added complexities that are involved can usually be avoided.

Remember that SQL Server Clustering technology is an availability solution, not a scalability solution. Clustering is implemented for failover to make sure that data is always accessible.

Heavy transaction processing systems and applications with significant ASP dynamic page generation often find CPU to be a bottleneck. In these types of applications, the processor must do a lot of computational work and should be monitored to make sure enough resources are in place.

Disk

Use multiple SCSI disk controllers to ensure maximum throughput. Also, ensure that the proper number of spindles are in place. This will enable handling large I/O volumes. Also monitor SQL Server closely, and consider putting the various database components (tempDB, indexes, tables, log files, etc.) on separate disks to minimize contention.

With Windows 2000, Microsoft has reintroduced the Disk Defragmenter utility. Use this tool to periodically defragment server hard drives to ensure maximum disk performance. The utility is located under Start, Programs, Administrative Tools, Computer Management.

Network

Use fast 100 Mbps network cards in Web and SQL Servers for fast throughput. Also consider having a dedicated connection between these two physical tiers so that application data doesn't have to contend with other network traffic.

When ODBC connection pooling between MTS and SQL, use TCP/IP as the network connection protocol. Additionally, remove any network protocols not being used, such as IPX/SPX and NetBEUI.

Even if you've done everything to ensure fast network throughput on the server side, not much can be done if users have slow Internet connections. You can make them aware of problems on their side, however, through a simple test that approximates their true connection speed. An example is located at http://computingcentral.msn.com/topics/bandwidth/speedtest.asp.

Network problems are a common bottleneck for sites with primarily static content. Since the CPU doesn't have to do a lot of work building pages, it spends most of its time sending pages out over the wire. If the network is congested, it will likely become a bottleneck.

SMP

Moving to a multi-CPU environment brings added complexities and challenges because it's an inherently more complicated system. Don't expect immediate linear scalability by simply adding more processors to your boxes. The operating system must perform more work context switching between threads on the different processors, so you won't get exactly double or quadruple the speed by adding one or three processors to a single-CPU box.

Pedro Silva's ongoing Duwamish Diary series is located at:

http://msdn.microsoft.com/voices

This is a good reference for SMP scalability issues. Microsoft has established a team tasked with building a fictional online book retailer called Duwamish. This team has been documenting challenges and solutions throughout its lifecycle. Recently the web servers hosting the Duwamish application were scaled up to four processors, and the following issues were noted:

- enable the ASP ThreadGate parameter (dynamically configures the number of threads allocated to IIS)
- cache frequently used XSL in the Application object (must use free-threaded XML DOM)
- avoid heavy use of the registry
- use the <OBJECT> tag instead of Server.CreateObject (the ProgID doesn't have to be resolved, saving a registry read)
- consider third-party Windows NT heap management tools for multi-CPU (SmartHeap for SMP from http://www.microquill.com/smp)

Windows DNA Performance Tuning Tools

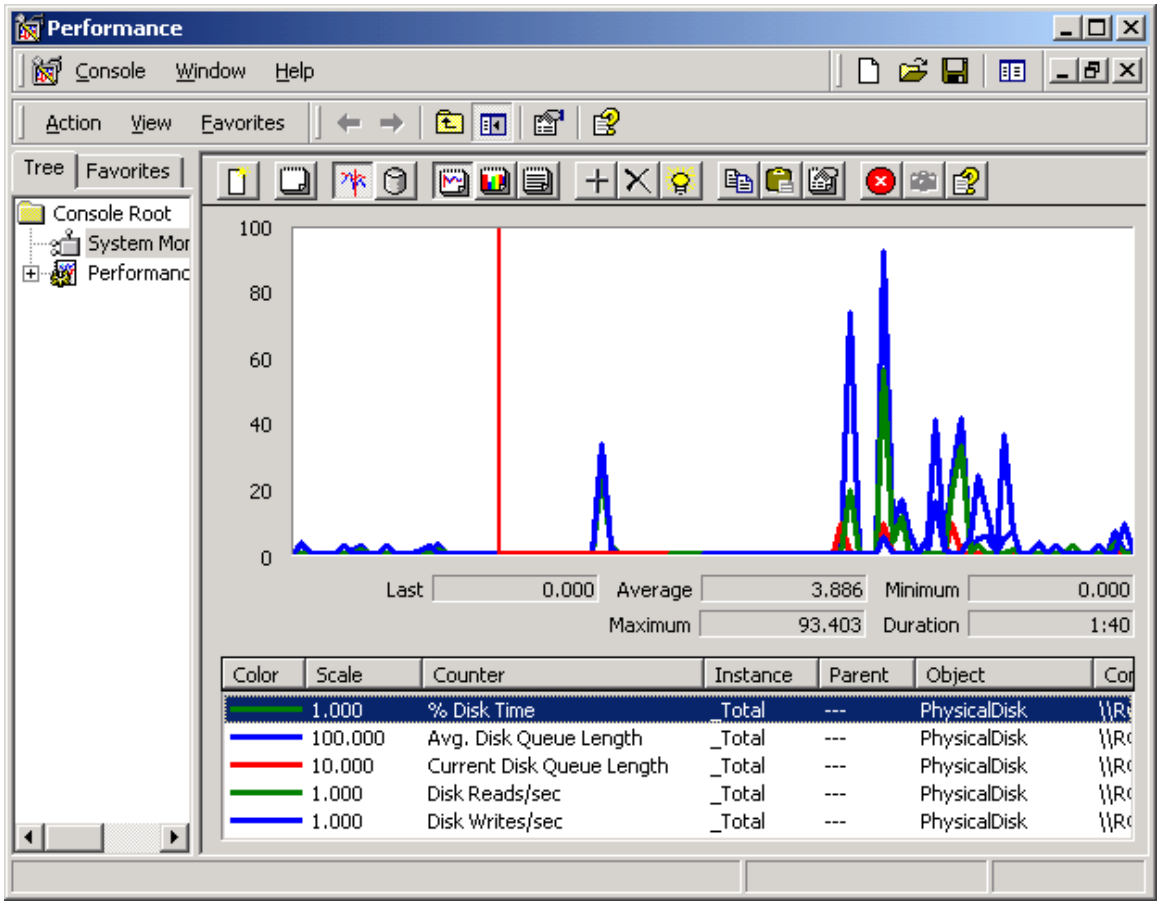
A variety of tools exist in the Windows DNA platform that provide developers and testers with the ability to get details about the internal workings of application code, system services, and physical configuration. These tools are designed to help identify

bottlenecks so that they can be alleviated, enhancing system performance. The most important tools in the tuning toolbox include:

- Performance Monitor
- Web Application Stress Tool
- DNA Performance Kit
- IIS Exception Monitor
- SQL Profiler & SQL Query Analyzer
- Visual Studio Analyzer
- Third Party Tools

Performance Monitor

The Windows NT Performance Monitor (PerfMon) gives you the capability of looking "under the covers" at the internal workings of the operating system and the various components of an application. You can view the data in real time, or configure PerfMon to write to a log file for later analysis:



A few of the key counters to track relating to tuning Windows DNA applications include:

- **Active Server Pages:** Requests/Sec, Req Rejected, Req Queued, Sessions Current
- **Inetinfo process:** Private Bytes, Virtual Bytes, Handle Count
- **Processor:** % Processor Time, Interrupts/sec
- **Memory:** Pages/sec, Page Faults/sec, Page Reads/sec, Page Writes/sec
- **PhysicalDisk:** % Disk Time, Current Disk Queue Length, Disk Reads/sec
- **Network:** % Utilization

Web Application Stress Tool

One of the most important tools available for tuning Windows DNA Internet applications is the Web Application Stress (WAS) Tool. This tool records test scripts and simulates user load, helping locate the bottlenecks in an application.

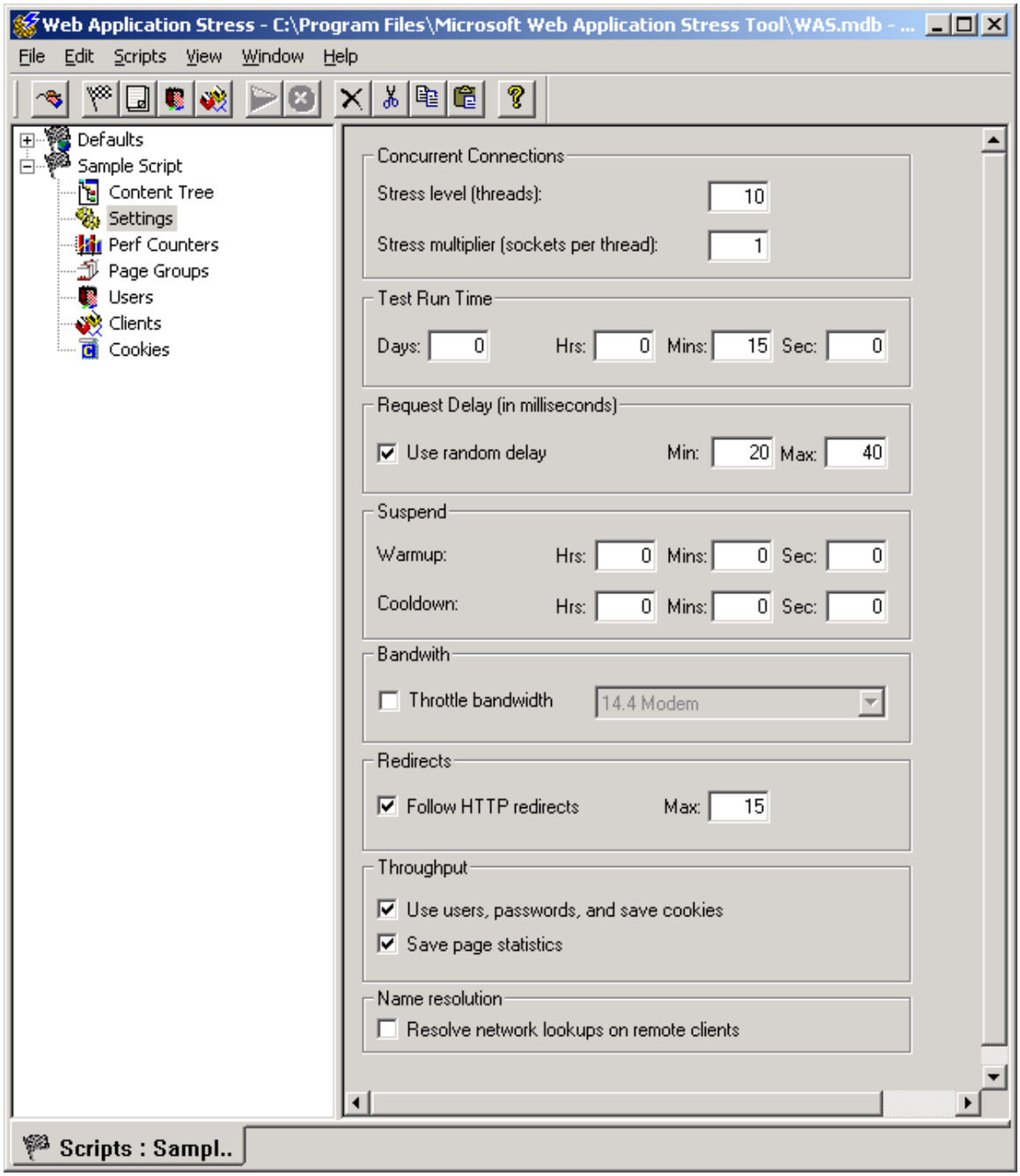
Stress testing should be done on a regular basis. Don't wait to put a load on an application just before you're ready to release to production. Putting a load on the application early in the development process will help you catch problems that may be too costly to fix later in the development life cycle.

Try to simulate real-world scenarios as much as possible. Use the WAS tool to record test scripts that emulate users clicks on your web site. Determine your application's peak load time and use the WAS tool to simulate this heavy load. Focus on being able to meet the HTTP requests that will occur during such heavy usage periods.

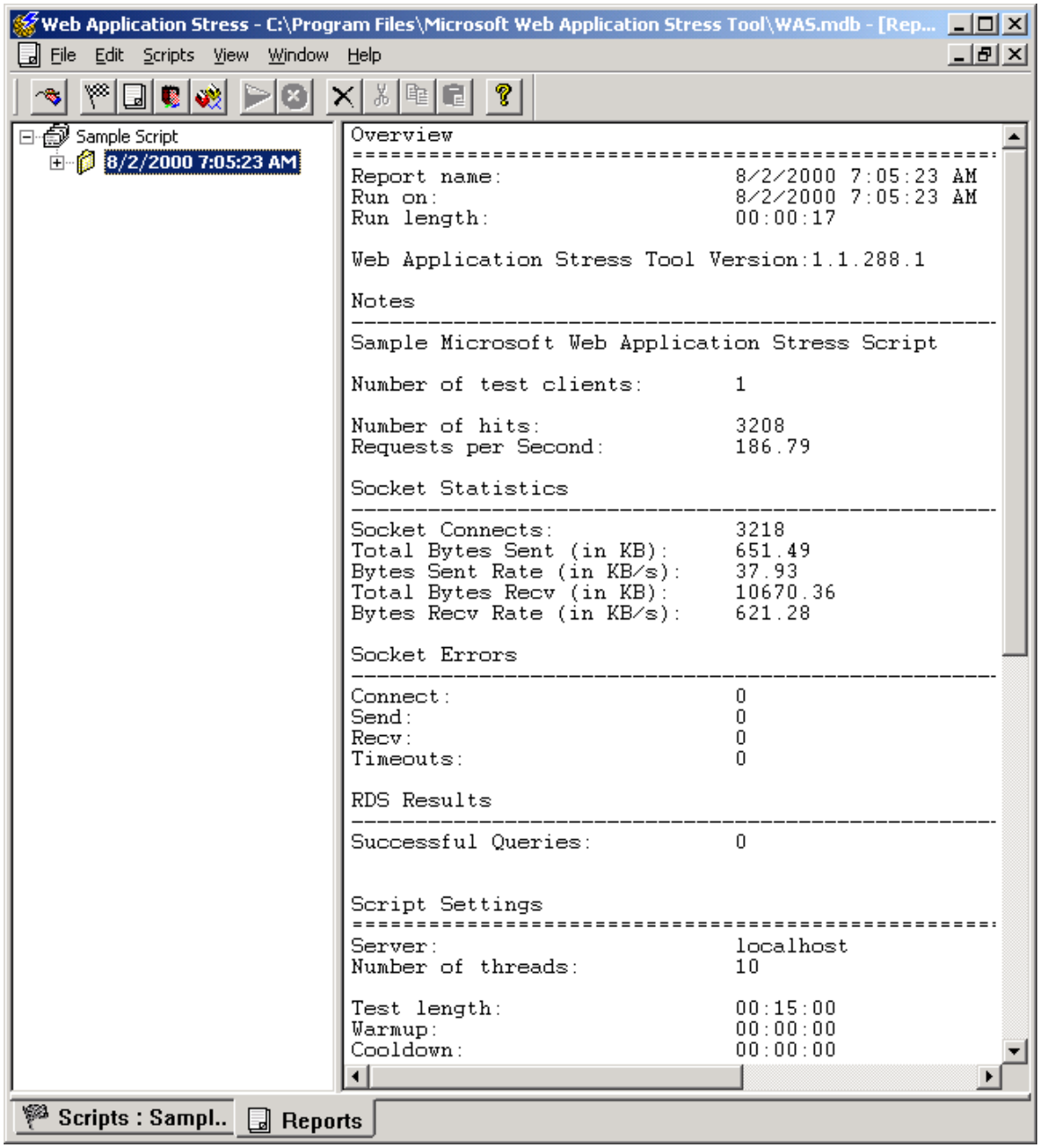
The test environment should match the production environment as closely as possible. Although this may not be financially feasible, it's important to get as close as possible. If you have a multi-CPU SQL box in production, you should be testing on a multi-CPU box. You'll encounter different problems with different hardware configurations, and it's of little use to solve problems in the test environment that are different from the problems that will be encountered in production.

Try to isolate the test environment from other network activity, placing the test box on its own network segment if possible. This will provide more accurate test results. If you have limited resources, try testing overnight using developer workstations as clients. It's easy to copy the WAS test scripts to multiple machines, and this is a convenient and cost-effective way to run more accurate tests. Also, remember to populate SQL Server tables with realistic data volumes. If you're testing against only a few rows of data while the production database holds millions of rows, test results will not be accurate.

The WAS tool allows you to configure load tests through a variety of settings:



After you've configured and run a test script, examine the report that WAS creates. Look for long-running pages which might require tuning:



The Web Application Stress Tool is a free download from Microsoft, available at http://webtool.rte.microsoft.com.

DNA Performance Kit

The DNA Performance Kit contains a wealth of tuning information, and the help file alone is worth the download. The kit is made freely available by Microsoft at http://www.microsoft.com/com/resources/WinDNAperf.asp. This too contains benchmarks for data access technologies, development languages, and hardware/software configurations. It contains a sample COM+ application with

different parameters that can be tweaked so that you can view results of various tests of your own.

Although the DNA PerfKit also allows you to wrap your own custom interface and subject it to load testing, this is not for the faint of heart. It's designed mainly for C++ interfaces, and wrapping a VB interface involves using OLE Viewer to create a typelib and MIDL to compile an IDL file.

Sample tuning scenarios provide the user with a glimpse of how an application might be tuned, step by step. Source code is provided for all COM components, including C++, VB, and J++, making it a great learning tool.

IIS Exception Monitor

The IIS Exception Monitor is a utility provided free of charge from Microsoft that is helpful for debugging COM+ library applications. Remember that library applications run in-process with IIS, so if a bug causes the application to crash, the web server will also crash. The IIS Exception Monitor will log which component and method caused the problem when a crash does occur so that it can be corrected. Without this utility you'll be lucky if anything helpful is written to the Event Log before a crash occurs, making debugging difficult. A log file analyzer utility is also included which helps make sense of the log files that the utility creates.

Use the IIS Exception Monitor during a project's development and testing phases to ensure application stability before deployment to production. This tool can be downloaded at http://msdn.microsoft.com/workshop/server/iis/ixcptmon.asp. The Windows and IIS debug symbols are required to use this tool. Since the debug files can be quite large, install them from the Windows Diagnostic CD instead of downloading them if you have the CD.

SQL Profiler & SQL Query Analyzer

Use these SQL Server tools to track down long-running and inefficient queries. See the Data Services section for a detailed description of their use.

Visual Studio Analyzer

Visual Studio Analyzer (VSA) is a tool that's designed to provide a picture of the overall functioning of a Windows DNA application. Unlike debugging and tuning within a single language such as Visual Basic, VSA shows interactions between components, machines, and processes. It captures data through events, and unwanted data can be filtered out. Graphical displays allow the architect to track down problems and bottlenecks.

VSA operates within the Visual InterDev IDE, and creating a new VSA project is similar to creating a Visual InterDev web project. VSA is available with the Enterprise Edition of Visual Studio.

Third Party Tools

Other third party tools can be helpful for tuning Windows DNA applications. Rational's Quantify and NuMega's TrueTime products are performance profiling tools. They construct reports that detail performance statistics captured during application tests. Even components for which you don't have source code are timed, providing a complete performance profile of the entire system.

Rational's PureCoverage and NuMega's TrueCoverage products are source code coverage tools, tracking the sections of code within components that have not been tested. This information provides testers with a chance to rewrite test scripts to cover all subroutines and methods before components are deployed to production. Although these are not strictly performance tools, writing test scripts that cover ALL the code in a component is important to the overall performance of a system – it only takes one untested, poorly performing method to degrade performance.

Visit Rational's web site at http://www.rational.com for more information on Quantify and PureCoverage. For more details on TrueTime and TrueCoverage, visit the NuMega web site at http://www.numega.com.

Intel's VTune Performance Analyzer product is another performance profiling tool, but unlike Quantify and TrueTime, this tool focuses more on the hardware architecture on which an application is running. It can suggest when certain compiler options (such as Favor Pentium Pro™ in Visual Basic) should be turned on. VTune gathers detailed statistical data during tests and generates graphical reports that help identify bottlenecks. Learn more about VTune at http://developer.intel.com/vtune/analyzer/index.htm.

Summary

Paying close attention to efficient coding practices in each of the three logical tiers of the Windows DNA architecture is an important part of writing a well-tuned application. Several parameters can also be set in the Windows operating system and in IIS to improve performance. The physical configuration of a system is extremely important to overall performance. Microsoft provides several tools designed to help identify and resolve performance bottlenecks that may develop throughout the various logical and physical tiers.

Although multi-tier performance tuning may take persistence and determination, it's well worth the effort to track down and resolve problems with a poorly running system. It's quite satisfying to help meet the business goals of a company and deliver a quality user experience through tuning. Plus, simply put, it's a great feeling to see a system you've worked hard on tuning running like lightning!

Appendices

A. Latest Upgrades

Often upgrading to the latest version or service pack of a software product will provide performance enhancements. Check the documentation before installing for details:

- Windows 2000 – contains IIS5 and COM+, both improved over earlier versions. IIS5 includes two new methods, Server.Transfer and Server.Execute, which supercede the Response.Redirect method, no longer necessitating a client round-trip.
- SQL Server 2000 (beta) – will return XML natively from the database engine, removing the transformation step now required on middle-tier boxes; also promises more efficient indexing.
- Internet Explorer 5.x – improved DHTML rendering engine and enhanced XML support: http://www.microsoft.com/windows/ie
- VBScript 5.5 (for IIS4): http://msdn.microsoft.com/scripting
- SQL Server 7 SP2: http://www.microsoft.com/sql/downloads
- Visual Studio 6 SP4: http://msdn.microsoft.com/vstudio
- MDAC 2.5: http://www.microsoft.com/data

B. Helpful Web sites

The following web sites contain helpful information related to performance, scalability, and efficient design and development using tools within the Windows DNA architecture. Content is updated often, new technologies are introduced, and service pack bulletins are announced. Check these sites often to stay current:

- http://msdn.microsoft.com
  - /workshop
  - /library
  - /voices
  - /code (Duwamish Books Win DNA sample – Phase 4)
  - /downloads
- http://www.microsoft.com
  - /seminar
  - /data
- http://www.asptoday.com
- http://www.4GuysFromRolla.com

C. Helpful Newsgroups

An important resource too often neglected by developers is Internet newsgroups. The Outlook Express Newsreader provides an effective user interface to these discussion group forums. Often a specific technology question will already be answered, and if not, submit a post and wait a few hours for people from all over the globe to weigh in and provide guidance:

Microsoft.Public

- .scripting.vbscript
- .scripting.jscript
- .inetserver.asp.general
- .xml
- .xsl
- .vb.com
- .vb.database.ado
- .vb.enterprise
- .mts.programming
- .data.ado
- .sqlserver.programming
- .sqlserver.xml

D. References

1. Windows DNA

- *Tech Ed 2000: Windows DNA Application Panel Discussion* by Steve Kirk and John Boylan at http://msdn.microsoft.com/msdn-online/start/features/panel.asp
- *Windows DNA Sample Projects – A Comparative Guide* by Steve Kirk at http://msdn.microsoft.com/msdn-online/start/features/guide.asp
- *A Blueprint for Building Web Sites Using the Microsoft Windows DNA Platform* at http://msdn.microsoft.com/library/techart/dnablueprint.htm
- *Introduction to Designing and Building Windows DNA Applications* by Frank E. Redmond III at http://msdn.microsoft.com/library/techart/windnadesign_intro.htm
- *Top Windows DNA Performance Mistakes and How to Prevent Them* by Gary Geiger and Jon Pulsipher at http://msdn.microsoft.com/msdn-online/start/features/windnamistakes.asp
- *Deploying Windows 2000 with IIS 5.0 for Dot Coms: Best Practices* at http://www.microsoft.com/windows2000/library/planning/incremental/iisdotcom.asp
- *Problem Isolation in Windows DNA Solutions* at http://support.microsoft.com/support/dna/bundles/probisol/default.asp
- *HOWTO: Isolate and Identify the Source of Inetinfo or Other Process Memory Leaks* at http://support.microsoft.com/support/kb/articles/Q253/7/06.asp

2. ASP (VBScript)

- Microsoft Internet Information Services 5.0 Resource Guide, Microsoft Press, 2000, Redmond, Washington.
- *Improving ASP Application Performance* by J.D. Meier at http://msdn.microsoft.com/voices/server03272000.asp

- *25+ ASP Tips to Improve Performance and Style* by Len Cardinal and George Reilly at http://msdn.microsoft.com/workshop/server/asp/asptips.asp
- *Maximizing the Performance of Your Active Server Pages* by Nancy Winnick Cluts at http://msdn.microsoft.com/workshop/server/asp/maxperf.asp
- *Optimizing ASP Performance in Site Server 3.0 Commerce Edition* at http://msdn.microsoft.com/library/winresource/ssreskit/rk_vcrocket_lcno.htm
- *Got Any Cache?* by Nancy Winnick Cluts at http://msdn.microsoft.com/workshop/server/feature/cache.asp
- *Server Performance and Scalability Killers* by George Reilly at http://msdn.microsoft.com/workshop/server/iis/tencom.asp
- *Enhancing Performance in ASP – Part 1* by Wayne Plourde at http://www.asptoday.com/articles/20000113.htm
- *Enhancing Performance in ASP – Part 2* by Wayne Plourde at http://www.asptoday.com/articles/20000426.htm
- *Don't Skimp on the Variables!* at http://www.4guysfromrolla.com/webtech/top10/int3.shtml
- *Explicitly Close and Destroy Object Instances!* at http://www.4guysfromrolla.com/webtech/top10/int1.shtml
- *Increasing the Speed of your ASP Scripts* at http://www.4guysfromrolla.com/webtech/121398-1.shtml

3. HTML / XML

- *Using Server-Side XSL for Early Rendering: Generating Frequently Accessed Data-Driven Web Pages in Advance* by Paul Enfield at http://msdn.microsoft.com/msdnmag/issues/0400/earlyrend/earlyrend.asp
- http://www.xbuilder.net (Third-party early-rendering software from Wayne Berry, the designer of the http://www.15seconds.com web site)
- *Solutions / Best Practices of www.microsoft.com* at http://www.microsoft.com/backstage/solutions.htm
- *Inside MSXML Performance* by Chris Lovett at http://msdn.microsoft.com/xml/articles/xml02212000.asp
- *Inside MSXML3 Performance* by Chris Lovett at http://msdn.microsoft.com/xml/articles/xml03202000.asp
- *ASP Technology and the XML DOM* by Alan McBee at http://msdn.microsoft.com/xml/articles/xml092099.asp
- *Creating and Optimizing Performance for XML Document/View Web Applications* by Dino Esposito at http://msdn.microsoft.com/msdnmag/issues/0600/cutting/cutting0600.asp
- *Simply Amazing: Caching in the Browser* by Robert Hess at http://msdn.microsoft.com/voices/hess06122000.asp

4. VB / COM+

- *Active Server Pages and COM Apartments* by Don Box at http://www.develop.com/dbox/aspapt.asp

- *INFO: Design Guidelines for VB Components Under ASP* at http://support.microsoft.com/support/kb/articles/Q243/5/48.asp
- *Developing a Visual Basic Component for IIS/MTS* by Troy Cambra at http://msdn.microsoft.com/workshop/server/components/vbmtsiis.asp
- *COM+ Application Guidelines for Visual Basic Development* by Edward A. Jezierski at http://msdn.microsoft.com/library/techart/complus.htm

5. ADO / MDAC

- *Pooling in the Microsoft Data Access Components* by Leland Ahlbeck and Don Willits at http://msdn.microsoft.com/library/techart/pooling2.htm
- *Improving the Performance of Data Access Components with IIS 4.0* at http://msdn.microsoft.com/workshop/server/components/daciisperf.asp
- *Improving MDAC Application Performance* by Suresh Kannan at http://msdn.microsoft.com/library/psdk/dasdk/impr8l2m.htm
- *Fitch & Mather Stocks: Data Access Layer* by Scott Stanfield at http://msdn.microsoft.com/library/techart/fmstocks_dal_sql.htm
- *Top Ten Tips: Accessing SQL Through ADO and ASP* by J.D. Meier at http://microsoft.com/mind/1198/ado/ado.htm
- *Benchmarking Different Approaches to Updating a Database* by Geoffrey Pennington at http://www.asptoday.com/articles/19990610.htm
- *Cursor & LockType Performance Issues* at http://www.4guysfromrolla.com/webtech/062799-3.shtml
- *Optimizing ADO Calls* by Mike Shaffer at http://www.4guysfromrolla.com/webtech/120299-1.shtml
- *Optimizing ADO Calls Using the Set Statement* by Mike Shaffer at http://www.4guysfromrolla.com/webtech/120899-1.shtml
- *Speeding up ASP by Using GetString* at http://www.4guysfromrolla.com/webtech/121598-1.shtml

6. SQL Server

- SQL Server Books Online
- *Microsoft SQL Server 7.0 Performance Tuning Guide* by Henry Lau at http://msdn.microsoft.com/library/techart/msdn_sql7perftune.htm
- *Tuning SQL Server 7 for High Performance* by Alex W. Chang at http://www.compaq.com/sqlserver

7. Network / Hardware Configuration and Tuning

- Internet Information Server Resource Kit, Chapter 4 – Performance Tuning and Optimization.
- Microsoft Windows 2000 Server Operations Guide, Part 2– Performance Monitoring, Microsoft Press, 2000, Redmond, Washington.
- *Duwamish Diary: Another Brick in the Wall* by Pedro Silva at http://msdn.microsoft.com/voices/sampleapp02032000.asp

- *Duwamish Diary: Contention… the Final Frontier* by Pedro Silva at http://msdn.microsoft.com/voices/sampleapp05042000.asp
- *Duwamish Diary: Beyond Contention* by Pedro Silva at http://msdn.microsoft.com/voices/sampleapp04202000.asp
- *Solutions for Poor Server Performance* by Gary Duthie at http://www.microsoft.com/technet/iis/sol.asp
- *Tuning the FMStocks Application* at http://msdn.microsoft.com/vstudio/centers/scale/tuning.asp
- *Performance Tuning Your Windows NT Web* at http://www.compaq.com/support/techpubs/whitepapers/457a1096.html

8. Stress Testing (Web Application Stress Tool)

- *Why Microsoft.com Believes in Testing the Web* at http://www.microsoft.com/backstage/bkst_column_21.htm
- *Stress Testing Data Access Components in Windows DNA Applications* by Mike Schelstrate at http://msdn.microsoft.com/msdn-online/start/features/dnastress.asp
- *I Can't Stress It Enough – Load Test Your ASP Application* by J.D. Meier at http://msdn.microsoft.com/workshop/server/asp/server092799.asp
- *Optimization and Testing Performance with the Web Application Stress Tool* by Scott Allen and Benoy Jose at http://www.asptoday.com/articles/20000420.htm
- *DNA testing: Put a load on early and often* by Giovanni Perrone at http://www.devx.com/upload/free/features/entdev/1999/08aug99/cs0899/cs0899.asp

9. DNA Performance Kit

- *Windows DNA Performance Kit Beta* at http://www.microsoft.com/com/resources/WinDNAperf.asp
- Windows DNA Performance Kit Beta Online Help

10. IIS Exception Monitor

- *Troubleshooting with the IIS Exception Monitor* by Aaron Barth at http://msdn.microsoft.com/workshop/server/iis/ixcptmon.asp
- *Analyzing Logs from IIS Exception Monitor* by Geoff Gray at http://msdn.microsoft.com/workshop/server/iis/readlogs.asp

11. Visual Studio Analyzer

- *Performance Analysis and Tuning with Visual Studio Analyzer (12/98)* by Chris Zeigler at http://www.gasullivan.com/1techlead.asp
- *Monitoring Events in Distributed Applications Using Visual Studio Analyzer* by Mai-lan Tomsen at http://www.microsoft.com/mind/0699/analyzer/analyzer.htm

E. Acknowledgements

Special thanks to Bill Wagner, Oracle Certified DBA, G.A. Sullivan, and to Eric Brown and John Pickett, Sr. Technical Architects, e-Business Practice, G.A. Sullivan. Thanks also to Rich Hepburn, MCSD, ASI Consulting, Toronto, Canada.

G. A. Sullivan
www.gasullivan.com
Email: corporate@gasullivan.com

Headquarters:
55 West Port Plaza, Suite 100
St. Louis MO 63146-3131
Phone: (314) 213-5600
Fax: (314) 213-5700

Atlanta Office:
5909 Peachtree Dunwoody Road, Suite 730
Atlanta, GA 30328-6106
Phone: (678) 781-5300
Fax: (678)781-5350

Cincinnati Office:
10200 Alliance Road, Suite 100
Cincinnati, OH 45242
Phone: (513) 745-2100
Fax: (513) 745-2150

Fairview Heights, IL Office:
141 Market Place, Suite 210
Fairview Heights, IL 62208-2013
Phone: (618) 394-7600
Fax: (618) 394-7660

Kansas City Office:
7400 West 110th Street, Suite 260
Overland Park, KS 66210-2341
Phone: (913) 696-3400
Fax: (913) 696-3500

Nashville Office:
Highwoods Plaza II
103 Powell Court, Suite 130
Nashville, TN 37027-5016
Phone: (615) 309-7000
Fax: (615) 309-7020

San Antonio Office:
8000 IH 10 West, Suite 950
San Antonio, TX 78230-3868
Phone: (210) 442-3600
Fax: (210) 442-3660

The Netherlands Office:
G. A. Sullivan
Statenlaan 63
5223 LA's - Hertogenbosch
The Netherlands
Phone: +31 (0)73 6223200
Fax: +31 (0)73 6223250