Guidelines and Techniques for Designing high performance Web sites

HyPerformix, Inc.
4301 Westbank Drive
Bldg. A, Suite 300
Austin, TX 78746
www.hyperformix.com
1.800.759.6333

Abstract: *This document presents a discussion on tips, tricks and techniques for designing high performance web sites. All of the information contained within this document is available in the public domain.*

Table of Contents

1. INTRODUCTION 3
2. GENERAL CONSIDERATIONS 3
3. GRAPHICS 3
4. CACHING 5
5. HTTP COMPRESSION 6
6. PREFERRED CUSTOMERS 6
7. COOKIES 6
8. FRAMES 7
9. BROWSER PORTS 7
10. COMMUNICATIONS 7
11. URL REFERENCES 8
12. COMMENTS 8
13. HTML 9
14. DHMTL 9
15. DYNAMIC PAGES 9
16. JAVA 9
17. SCRIPTING LANGUAGE FOR WEB APPLICATIONS 10
18. JAVASCRIPT 10
19. ACTIVE SERVER PAGES 10
20. VISUAL BASIC 11
21. C AND C++ 11
22. FASTCGI 12
23. SERVER 12
24. WEB AVAILABILITY MANAGEMENT (WAM) 13
25. MISCELLANEOUS 13
26. CONCLUSION 14

APPENDIX A REFERENCES 15
APPENDIX B IMPROVING ASP APPLICATION PERFORMANCE 17
APPENDIX C TIPS TO IMPROVE ASP APPLICATION PERFORMANCE 28
APPENDIX D SERVER PERFORMANCE AND SCALABILITY KILLERS 34

1. Introduction

The purpose of this document is to present a set of tips, tricks and techniques for designing high performance web sites. These tips, tricks and techniques have been gleaned from a multitude of sources (books, articles, web sites, magazines, etc.).

The focus of this document is on performance issues. Both Internet and Intranet performance issues are presented; however, Internet performance is the primary focus. The source of each set of data is presented in Appendix A and noted where used by a bolded number within brackets **[00]**.

Appendices B, C and D contain articles that have significant content, it was decided to present them in their entirety rather than attempt to split them into the various topic sections.

2. General Considerations

- There are three ways to increase your web site performance **[47]**:
  1. Decrease the generation time of the page (server-related).
  2. Increase the speed that the information reaches your customers (transport-related).
  3. Lower the amount of time it takes to render your web site (browser-related).
- Fast downloads are the single most important usability consideration in Web design **[39]**. Sun Microsystems recently surveyed 1854 users and found that speed was more than three times as important as looks**.**
- An end-to-end Performance Study is necessary to determine the factors affecting response time.
- A good rule of thumb is that the size of each page, including graphic files, should not take longer than ten seconds to load on the slowest modem accessing your Web site **[47]**. For example, if the slowest users have 56K modems or better, the total size of your graphics and page should be 56 kilobytes or less. (10 bits per byte, 5.6Kbytes per second, times 10 seconds). More typically the actual transfer rate for a 56K modem would be closer to 30Kbps which would limit the total size of each page to only 30 kilobytes or less.
- Experiment with scrolling, i.e., sending more data than will fit on a page. The user would then scroll or hypertext to the information that is not showing. This way screens may be combined and during the time the user is reading and filling out the first part of the page the rest of page is being sent to the browser.

3. Graphics

- It is good design practice to provide the user the ability to choose Graphics or Text only. Include alternate text for each selectable image, for users that have images turned off.
- Avoid using large graphical fonts and images.
- Where possible use images in the GIF format rather than the JPG format.
- Use graphics only when they are critical to the information content of your page, not just for visual appeal.
- All images should have height and width parameters. Providing the height and width allows the browser to reserve that space for the graphic and continue loading other parts of the page while the image is loading **[31]**.
- Use GIF transparency only when needed. It makes the file larger and takes longer to load.

- Minimize the number of colors being used in a single image. Design your pages so that they look good on 256-color screens. 50 colors are a reasonable limit **[1]**. Force all graphics to use the same color palette **[49]**, especially those that appear on a single page. It is also worth considering using the Netscape color cube **[31]** and **[35]**. There are applications that can reduce the number of colors and the size of images **[31]**.
- Use graphical bullets and divider bars when they serve a purpose, not because they look "neat."
- Since each bullet and accent graphic is another image that needs to be loaded from the host computer, use a small set of bullets or accent graphics repeatedly, rather than using a large number only once each. Once the image has been loaded it is cached (stored) on the client computer and is available with minimal load time.
- Experiment with consolidating graphic objects. Even though you pass the same number of bytes, every object potentially requires a check to see if the version in the browser cache is out of date. This check is a message send-receive round trip from the browser to the server. For static images (ones that rarely change) this check is all overhead. Consolidating N images into one reduces this overhead.
- Use client-side image maps to consolidate smaller images that are hyperlinks. Hotspots on the image map can be used to link the various parts of the image to different URLs.
- Experiment with pre-loading graphics via the "single-pixel" technique. If, for example, the web site has a page with a form that the user must fill out, and the page after the form submission has a large graphic, then it might be possible to transparently pre-load the large graphic by including the following HTML code at the bottom of the page that has the form:
  <IMG SRC="large_image_for_next_page.jpg" WIDTH=1 HEIGHT=1>. The large image will be loaded while the user is filling out the form. Regardless of the original size of the image, it will appear as a single pixel (virtually invisible) on the page with the form. The benefit is that it will appear instantaneously when the following page is accessed.
- From Sun Microsystems: Guide to Web Style **[1]**
  1. If you're going to use many images close together on a page, loading each image requires a new connection between the user's browser and a web server. For users on low-bandwidth connections, or those connecting to overloaded servers, one large image may load faster than several smaller ones, because of the elimination of multiple browser/server connections. Users on higher-bandwidth connections going to fast servers may benefit from having multiple smaller images, as some browsers will make multiple simultaneous connections to a server to retrieve images. Understand your audience and their typical bandwidth and browser usage, and choose appropriately.
  2. Supply interlaced GIF files in your pages to allow images to load in multiple passes (i.e., the "window shade" or multi-layer effect). If your browser supports interlaced images, the interlaced picture will load in a series of passes, each one adding more detail. The non-interlaced image will appear in successive horizontal bands. The interlaced image shows more of the image sooner, with less detail in the early passes.
  3. Specify both low and high-resolution JPEG files to enable smart browsers to paint an entire page for your reader and then go back and fill in high-quality images after the page has loaded. This technique is worthwhile on large images, especially those over 100K.
- From Designing High Performance Web Pages **[31]**:
  1. The first and most critical is to keep images small (<25k)
  2. Where images are very large use a GIF thumbnail image and allow the user to click on it if they want to see the large image. Make it clear how big the large image is before they choose to see it.
  3. Background images they should be very plain and faded so that they don't interfere with the ability to read the pages. The actual gif to be loaded should be extremely small (<8k) as it will be tiled anyway. There is also no need to interlace it.

4. Image maps: Each click on an image map results in two files being opened – the image and the map. There are now new techniques to reduce this to one I/O by including the map portion in the web page, instead of as a separate file.

- Reduce the file size of graphics by making them smaller **[47]**. Reducing a 100x100 pixel graphic by 10% on either side (90x90 pixels) reduces the total pixel count by 20% in file size.
- Reduce the number of colors in the graphic or reduce the graphics DPI **[47]**. Gif files should be used for line art or text with only a few colors. Maximum file compression can be achieved with JPEG files by increasing the compression rate. Gif file size can be reducing by adjusting bit depth.
- Decreasing the number of graphics **[47]**. One example is a web page that has one hundred items listed with a small icon (8x8 pixel 200 bytes) bulleting every item. The graphic is small and it is reused so it doesn't take a long time to download or load into memory. However the operating system needs to render this graphic a hundred times.
- Reuse graphics on multiple pages to take advantage of caching **[47]** and **[62]**.
- Text should be in text form not graphic form.
- Experiment with compression tools with high/med/low resolution to determine the smallest number of bytes that look good **[49]**.
- Use 72-dpi resolution and experiment with compression tools with high/med/low resolution to determine the smallest number of bytes that look good **[49]**.
- Use the DeBabelizer **[43]** program **[49].**

4. Caching

- There are several objects and page types that cannot be cached **[31], [33]**. Redirected documents and server side includes (used by fake frames) and imagemaps are not cached. Client side imagemaps (USEMAP) can be cached. If the URL ends with a directory with no / at the end then the URL will not be cached. If it ends with a / or a page name (i.e. fred.html) then it can be cached. Other objects that cannot be cached include pages that are dynamically built or renamed, pages with counters or cookies, results pages from searches and most CGI programs. Any page that is secure (SSL) also cannot be cached. If there are pages on the web site that should not be cached then the never cache attribute should be coded in the page header to ensure that other sites do not cache this page.
- Some browsers, e.g., I.E. 4.0, do not allow ASP pages to be cached. To maximize client side caching, pull all the static HTML out of the ASP pages, place it in separate files, and then call the static HTML from the ASP page.
- Experiment with pre-loading graphics by putting the ones for the “next” page at the bottom of the “current” page. This would cause them to be loaded while the user is looking at the top of the current page, and they will be in cache when the user clicks to the next page. Of course, this presumes that you are dealing with static web pages. If they are dynamically generated then read-ahead will do you no good. When the browser requests the next page, the server will have to regenerate the page. There are three potential problems:
  1. The browser will not yet be "done" while loading the objects for the "next" page, and this sometimes causes input forms to act weird.
  2. The fact that the page is still loading may cause the user to delay filling out a form. A notice should be placed on the top of the page informing the user that it is ok to begin filling out the form while the bottom of the page is still loading.
  3. To keep the graphic for the next page from showing on the “current” page, simply specify a height=1 and width=1 to force the graphic into a single pixel.
- Read Appendix D **[53]**.
- Put a link on the home page that points the user to information about how to set up the user’s browser to maximize caching benefits.
- Investigate using predictive server caching software, for example, Fireclick Blueflame **[54]**.

- There are several good papers on the subject of caching at **[62].**

5. HTTP Compression

HTTP compression is a very powerful technique for increasing the data transfer rate from the web server to the client browser **[57].** HTTP compression reduces the size of the content of files transferred from the web server to the client. The web server must have a compression capability and the client browser must be HTTP 1.1 compliant.

The file content is compressed via either "Content-Encoding" or "Transfer-Encoding". "Content-Encoding" applies to methods of encoding and/or compression that have been already applied to documents *before* they are requested. This is also known as "pre-compressing" pages. "Transfer-Encoding" applies to methods of encoding and/or compression used DURING the actual transmission of the data itself.

Most modern browsers (Internet Explorer versions 4 and above, Netscape 4.5 and above, and Windows Explorer) are all HTTP 1.1 compliant clients. However, the browsers can be optionally set to be non-compliant, therefore it is important that the client user verify that his browser is operating in the HTTP 1.1 mode. The web site http://12.17.228.52:7000/ can automatically check your browser (if you are not on a proxy server). The Internet Explorer browser setting can also be manually checked in Internet Options, Advanced, HTTP 1.1 settings, Use HTTP 1.1.

A partial list of server compression programs and other compression information can be found at **[58]**, **[59]** and **[60].**

6. Preferred Customers

Content-delivery firms such as EpicRealm and CacheFlow offer a prioritization service that allows companies to serve Web pages to loyal customers faster than to other visitors **[61]** and **[62]**. PriorityRealm has been popular with early users, as traffic volumes on the Internet continue to rise and Web administrators look for ways to keep their best customers from becoming frustrated while waiting for pages to load. However, note that critics say prioritization could be considered a threat to online privacy and equality. The data that labels priority customers could be misused, warns Electronic Privacy Information Center policy analyst Andrew Shen.

7. Cookies

- By default, many web servers automatically set cookies on for every page **[31]**. This means that for every get that is processed the server checks the client machine’s cookies.txt file to see if there is a cookie set in there. This causes the use of a great deal more bandwidth, and affects response time. If the user has cookies turned off at their browser, they will be incessantly reminded with constant messages about the server wanting to set a cookie. This normally results in two things – (1) lots of abusive email to the webmaster and (2) users who never come back.
- The purpose for cookies was to remember users and to offer them customized services **[31]**. Most servers now use httpd 1.1 keepalive, which allows for persistent sessions and does away with the need for cookies. Cookies also do not work very well with cache servers. However, if cookies must be used, here are some tips:
  1. Never require them on the home page
  2. Never set the server to require cookies on every get
  3. Let people know before they click on a link that cookies are required to go there

8. Frames

For most people frames are very confusing **[31], [32]**. It becomes impossible to bookmark the page so that you can return to it later and, depending on your browser, printouts become difficult. A better solution is to use a fake frame (using the #include in your html) that is on every page and looks like a frame, but is really two tables. Since the result is a standard html page it can be book marked and printed. The only problem with a fake frame is that the data page cannot be scrolled independent of the frame data.

9. Browser Ports

- Performance impact of multiple open ports.
  Problem: When the browser encounters a tag that has a SRC attribute (e.g. <script language="javascript" src="HelpFunctionsParent.js">), the browser opens a new port to request the information for the tag, in addition to the port already open for the HTML page or ASP page that contains the tag. Since NT can process only one port at a time, throughput degrades because NT has to swap between the two open ports to retrieve data. Sniffer traces can easily identify this problem.

  Solution: Most of the time, this situation does not cause a performance problem, because the kinds of tags that have a SRC attribute are associated with files that can be cached on the workstation. So, at most, there would be a negative performance impact only the first time the web page is accessed. The scenario that could cause repeated performance problems is when the SRC attribute references an ASP page. An example of this would be an ASP page that contains an IFRAME, and this IFRAME loads another ASP page. If the IFRAME has a SRC attribute specified, throughput would be restricted because the operating system would be swapping between two ports to download the data needed for both ASP pages. A better approach is to wait until the first ASP is finished loading, and then from its "window_onload" method you should set the SRC attribute for the IFRAME. Loading the second ASP page this way would allow the port for the first ASP page to close before a new one is opened for the second ASP page.
- Use HTTP Round-trip time and Keep-Alives **[46 page 53], [47 page 74-76]**. HTTP 1.1 allows multiple HTTP transactions across the same connection, or "persistent connection". This also enables "pipelining" of new client requests before previous requests are resolved, and simplifies authentication.
- Do enable Keep-Alives. HTTP 1.1 allows multiple transactions across the same connection. This transmits fewer network packets and reduces overall latency **[27], [46 page 53]**.
- Do enable pipelined requests across HTTP 1.1 **[27], [47 page 76]**. Sending multiple client requests before a reply is received improves both throughput and overall response time.

10. Communications

- All 56K modems do not have the same performance. The modems have significant performance differences in regard to latency **[4].** Latency is that time from when the modem receives a command until the desired action begins, e.g., the time from when a transmit command is received by the modem and when the data actually begins transmitting. It is important to select a modem with low latency especially when there are many small messages being transmitted.
- Improve the quality of the telephone line **[6]**.
- Increase TCP/IP listen intervals **[27], [46 page 139]**. Much of TCP/IP's resilience and congestion control relies on the assumption that communications sessions are persistent. Increasing TCP listen intervals and queue lengths permits more pending requests to hang around, rather than be denied service.

- Disable “slow-start” sessions for web-based applications **[27], [47 page 139]**. Disabling "slow start" sessions means that small IP windows don't prevail because there's no time in an average HTTP transmission to renegotiate larger windows. The relatively small data volume for web-based applications does not justify time lost determining the optimum transfer window. TCP/IP slow-start was designed for Internet services such as FTP or remote login that involve long connections and/or massive data transfer **[46 page 139]**. Effectively, the web server first sends a small number of packets and then ramps up to an optimum. This occurs for each new connection. Most web transactions transfer such small amounts of data that slow-start is a nuisance and wastes time.
- Use TCP/IP for your network library for performance and scalability **[51]**.
- Disable reverse DNS lookups **[27]**. Although checking on the identity of those who visit your site is important where security is paramount, other Web sites can greatly speed up their Web services by omitting this time-consuming step (it effectively doubles the round-trip time for HTTP requests). You can always use third-party security monitors, like Haystack Labs's WebStalker, to perform this kind of service for you if you suspect a security break or a break-in attempt.

11. URL References

- Make sure all links are valid URL’s.
- Reduce the number of characters in each URL. The URL http://x.com/a.htm will load faster than the URL http://x.com/web/code/first_stuff/intro/first_page_code.htm.
- There are 3 ways of coding a URL reference to get a page **[31]**. They each have slightly different performance. They are, in highest performance to slowest performance order:
  a. page_name/index.html
  b. page_name/
  c. page_name
- If “c” is coded, it causes a hit to the web site, then a “/” gets added and another hit occurs and then the site must find the default page. If “b” is coded, the site must find the default page. If the complete file name is provided as in “a”, the page is directly accessed. On a heavily hit site it can make a large difference. So, it is important that all URLs include the full file name.
- Additional URL information is in Section 4, Caching.

12. Comments

Maximize your comments in the development copy of each html file in order to assist in the maintenance of the file. However, strip all comments and whitespace from the html file that is loaded onto the web server **[49]** and **[52]**.

13. HTML

- Read Appendix B **[51]** and Appendix C **[52]**.
- Three methods of requesting a file (HTTP GET, CGI in C and CGI in Perl) showed the standard GET was equivalent or superior to the other methods in all cases **[28]**. In addition, compiled C outperformed interpreted Perl.
- Remove embedded tables and avoid using the COLSPAN attribute **[47]**. Netscape, more then Internet Explorer, slows down with nested tables.
- Avoid verbosity in defining the names of the html directory tree. For sites with many links, the time to download these extra characters may add a half-second or more to the response times. Use short names like g/ for graphics, etc. long names are just more data to transmit (twice, once from the server then back to the server) **[49]**.

14. DHMTL

DHTML provides the ability to modify an HTML page after it has been rendered by the browser using code that is written and packaged with the HMTL. This code can be VBScript, Javascript, Java, and/or Visual Basic. You should refrain from using Visual Basic and Java to drive a DHTML page at the present time. Netscape Navigator and Internet Explorer 3.02 do provide some client side scripting capabilities but they don't support the full DHTML model. Care needs to be taken in developing a web application using Javascript, which is targeted for these Browser platforms.

15. Dynamic Pages

- Dynamic pages (written in CGI, Perl, ASP, Cold Fusion, etc.) have a high response time because the machine must execute some code to get the page together so that it can be streamed out the network card **[47]**. The amount of response time that it can take can be up to twenty times greater then static pages that are just read from the hard disk.
- One way to reduce the response time of dynamic pages is to change them to static pages **[47]**. The XBuilder tool can convert dynamic pages to static pages. This works especially well if the pages do not change for every user. For example, if you are storing information in a database, then publishing using dynamic pages, your pages do not change per user, only when the database changes. Building these pages static with XBuilder **[48]** every time can increase your performance dramatically.
- XBuilder **[48]** can also reduce the size of the page by removing the tabs when it builds the pages. Tabs are not recognized by browsers and are only used for development clarity **[47]**. Removing them can decrease the page size on average by 10%.

16. Java

- Because of a couple of security scares (see CERT Vulnerability VB9806) many users have turned off Java and Javascript on their browsers **[31]**. When designing a web site it should be made clear whenever a link is going to require these.
- Make sure you have the latest Virtual Machine from http://www.microsoft.com/java/ **[51].**
- Read Appendix B **[51]** and Appendix C **[52]**.

17. Scripting Language for Web Applications

- Do not use interpretive scripts if the scripting language permits compilation **[27], [28]**. Rewrite non-compiled scripts in a compiled language. Compiled code usually runs an order of magnitude faster and reduces process overhead.
- Interpretive script is costly compared to compiled C **[46 page 70]**. Interpretive programs are generally slower than compiled code because of the line-at-a-time execution overhead. The CGI scripts execute in its own process space, therefore it does not benefit from multithreading **[46 page 148]**.

18. JavaScript

- Because of a couple of security scares (see CERT Vulnerability VB9806) many users have turned off Java and JavaScript on their browsers **[31]**. When designing a web site it should be made clear whenever a link is going to require these.
- From Site Optimization Tutorial by Paul Boutin **[50]:**
  1. Junk the JavaScript. We use to have a JavaScript function in HotBot that set the browser's keyboard focus to the text-input box. Just that one function caused a noticeable delay in the loading of the page.
  2. If you're using JavaScript to control plug-ins or DHTML, read the manuals for those components. You may find that your 20-line JavaScript program can be replaced by a built-in function that both loads and runs faster. We wrote a "NextTen" function in JavaScript to change the contents of a table loaded into MSIE4. Then we learned that IE's built-in nextPage function was more than 10 times faster and didn't mess up the page when it ran.

19. Active Server Pages

- Read Appendix B **[51]** and Appendix C **[52]**.
- Do use buffer=true <%response.buffer=true%> **[2].**
- Do not use response.flush **[2]**.
- Speed Tips by Charles Carroll **[6]**
  1. Any page that does not need session can benefit from this directive at the top **[6], [7]**:
     <%@ enablesessionstate=false %>
  2. All pages should be buffered see **[2], [6]**.
     - Add this to top:
       <%response.buffer=true%>
  3. Flush buffer and HTML tags to improve perceived speed **[8]**.
  4. Pages doing database access will improve performance if the following techniques are used:
     - Getrows **[9], [10]**
     - GetString **[11]**
     - Disconnected recordsets **[12]**
  5. Before just believing any article you read on the web or in an ASP book (many articles are not researched thoroughly) and implementing code changes TIME THE CODE BEFORE AND AFTER changes. Speedtimer **[13]** has the needed code that will time scripts to millisecond accuracy.
  6. Eliminate any code that reads com properties twice **[14]**.
  7. Just say no to VB components and/or recordsets stored at the session level **[15], [16], [17], [18], [19], [20].**
  8. Cache frequent query results/HTML generated from databases in application variables **[21].**

9. Open database connections as late as possible, Close database connections immediately when done. Placing your code a few lines later could make a huge difference as your scripts run round-robin with other scripts **[22]**.
10. Don't waste time running scripts if the user hit the "stop" button or went to different page/site **[25].** Normally scripts started by a user run to completion whether the user is there to read output or not running invisibly on server.
11. Bite the bullet and master **remote scripting** so your HTML and ASP scripts can communicate with ASP asynchronously and refresh portions of page without submitting page to server. Remote scripting works in both Netscape and IE; any Javascript capable browser. Some remote scripting links and a listserver to get help is at **[26].**

- From Web Workshop – "Improving ASP Application Performance" **[44]**
  1. Store all Visual Basic objects and the Scripting.Dictionary at page scope. Single-threaded apartment (STA) objects at Session or Application scope thwart concurrency through locking or serialization.
  2. Migrate massive ASP script code to custom components and to functions. The former is somewhat of a special case of the "compile don't interpret" advice offered from many sources. The latter, partition code into generic functions, also improves performance and lays the groundwork for future components.
  3. Avoid large #include files. There are two serious issues. First, a complete copy of each #include file referenced by each ASP page is loaded into memory, so redundant copies many exist and unused routines are not removed. Second, the time to search the namespace for routine and variable references expands.
  4. Global variables should be scarce. It is better to denormalize a large #include file into multiple files of greater specificity and granularity.
  5. Significant performance improvements are gained by keeping blocks of server-side script together rather than interspersing ASP and HTML **[45]**. This reduces the context switching overhead.
  6. Avoid Session state altogether. ASP sessions are Web-server specific and limit scalability across multiple machines. Plus the stateful environment consumes resources for each user. Try workarounds such as passing data in QueryStrings or hidden form fields. Also consider a database keyed to a user ID that persists across the "session".
  7. Test **Response.IsClientConnected** before long routines.
  8. Parse strings with regular expressions instead of loops.
  9. Investigate the effect of #include files and Single-thread Apartment versus Multi-thread Apartment **[46]**.

20. Visual Basic

Visual Basic performs well when you have the right version, use the right settings and follow good design guidelines: Appendix B **[51]** and Appendix C **[52]**.

21. C and C++

- It is important to pay attention to compiler options and operating system levels **[31]**. There are several options on the UNIX C compiler that should be tested – O, O2 and O3, Q and qstrict. At a minimum just using –O (basic optimization) will normally improve performance for a CPU intensive application 100%. It should also be noted that upgrading to a new version of the operating system can also change compiled program performance. Numbers as high as a 17% improvement with a recompile are not uncommon.
- There are a number of other programming tips that can improve C performance:
  1. Use ANSI standard library functions

2. Use abbreviated assignments e.g. c+=3
3. Don't code constants such as cat=1 in a loop
4. Avoid recursion – recursive calls use more memory and take more CPU
5. Inline functions (-Q) reduce execution time
6. Function calls add overhead so limit the number of functions to what is necessary to keep the program manageable
7. Call by reference for arrays reduces memory needs
8. Goto is more efficient than a case, however it is viewed as bad programming practice that leads to hard to maintain code
9. Global variables improve performance because data doesn't need to be passed around

22. Fastcgi

Every invocation of a CGI script or program spawns a new process or thread **[31]**. Because of the overhead associated with this, many server vendors now provide a separate API interface to their server where the main script is dynamically linked and executed. Unfortunately these scripts have to run under the server's identity and they thus have access to the server internals. Another alternative is to look into the new open interface for performance – Fastcgi. Fastcgi **[37]** has been developed at NCSA as a language independent interface that removes the need to use APIs to get performance.

23. Server

- Tune your server **[23], [24]**.
- Do use multithreading or a pre-allocated process pool as an alternative to forking a new process for each request **[27], [30]**. This reduces context-switching overhead. If it is necessary to spawn a task, do so within the run-time context of the Web server.
- Use server-side APIs **[27], [46 page 148]**. Most web servers include proprietary API's that create threads or sub-processes that run within the Web server process space. This avoids 95% of the usual context-switching overhead associated with CGI. Note that CGI scripts execute within their own process space, negating some of the benefits of multithreading.
- Reduce the use of CGIs and other inter-process (IPC) causes **[27]**. While CGIs and other forms of interprocess communication do indeed add to a Web server's overall capabilities, the overhead involved in calling across process boundaries can be a real drag on performance. If performance has priority over portability, use Web-server API and server-side includes.
- File Locking **[31]**. To write CGI programs it is often necessary to have code that needs to be executed in a serial fashion. Areas that require this may be updating files, generating counters and other similar functions. To do this, it is necessary to use file locking. During the time the lock is held, any other application using that lock will wait for the lock to be freed. For performance and response time reasons, it is important to lock the file as late as possible and to free the lock as early as possible. Obviously, locks should only be used when it is critical to do so.
- It is common in the UNIX world to write applications that use sleep loops while they wait on an event to occur **[31]**. This is not a good way to write code as resources such as memory are held during this time. This kind of spin loop also wastes CPU resources as it wakes up periodically to check whether anything has happened. Loops should be kept small and simple with any constant variables being initialized before the loop is entered. This avoids repeatedly executing instructions that only need to be executed once.
- DNS (domain name services) is used to resolve an IP address into a name so it can be written to the log file for the server, which makes it more readable for post processing of the logs **[31]**. This is the reverse of what normally happens at the client end where the server name needs to be translated into an IP address to find the server. If the IP address can't be resolved into a name, then a timeout occurs and,

depending on the options chosen, either the IP address is logged or the connection is refused. On a busy server, this should either be turned off or a DNS server should be loaded onto the web server to speed up the time for the lookups and to reduce network delays.

- CGI programs and httpd connections tend to run as subtasks or threads under the main server **[31]**. Because of this, the *maxuproc* parameter on the server (or maximum children/threads) should be reviewed to ensure it is high enough to cope with peak load. The default on many systems is 40 and web servers will typically need numbers in the range of 1000 to 4000. Web servers also benefit from running on an SMP server where multiple CPUs can be active at the same time servicing requests. Many servers also allow the dynamic creation (inetd controlled) of httpd processes whenever a request is received – it is normally better to configure the server to keep a minimum number of processes ready to go and to let that range up to a maximum. This reduces the overhead in creating address spaces.
- There are three security levels for the web pages – none, basic or uuencoded and SSL (certificates with encryption) **[31]**. The higher in the security hierarchy the pages are, the more of a performance hit that will be taken. Encryption takes CPU cycles and the impact of this should be carefully measured before production.
- There are also some good options available to reduce I/O and network delays **[31]**. For the network, look at *Mbufs*, transmit queue size, collisions, and TCP send and receive space. Recommendations on what to look at are described in **[34]**. For I/O look at placement of files and logs. Unless they are actively being used, turn off the referrer and agent logs. The referrer log tells you what web page the user came from while the agent log tells you the browser they are using. These are very I/O intensive and should be avoided. The logs that are of most use are the access log and the error log. At a minimum, these should be in their own filesystem and on a reserved disk. It is also possible to split out write only directories within the web space to their own disks or to split up the cgi programs and document directories onto separate disks. This will reduce I/O contention.
- Read Appendix D **[53]**.

24. Web Availability Management (WAM)

WAM is a concept that revolves around the idea of congestion-control **[38]**. Instead of giving everybody a bad experience, or simply rejecting users ('connection refused' type messages), you employ the Web-equivalent of Muzak (present a page that says 'Your call is important to us. Here's some nice pretty marketing fluff while you wait...'). By protecting the overloaded segments of the system from further congestion, you allow a much faster recovery time.

For example, CNN cut the graphics from their pages when the site is under load. It has also been proposed to drop functionality, e.g., an on-line mart may allow users to browse the catalog but not submit new orders, until the congestion has eased.

25. Miscellaneous

AUTOMATIC OPTIMIZATION OF WEB SITES

Utilize “site checkers” to determine performance metrics and “optimizer” programs to reduce the response time of web sites. A small sampling of available products:

- **SpaceAgent**. Optimizes each page of an entire web site.
- **Web Site Garage**. Offers several web-based products for performance diagnostics, image optimization, and traffic analysis **[41]**.
- **GIF Wizard**. Reduces GIFs, JPGs, and BMPs **[42]**. Shows the user a selection of possible sizes and color depths for each graphic or performs automatic reduction.

- **DeBabelizer**. A multimedia file conversion and optimization program **[43]**. Scripting, batching, and HTML parsing features allow graphics to be converted among formats and optimized for speed versus quality.

Browser Instances [55]
When new browser windows are launched from existing browser windows, new instances of Internet Explorer are not created. It is possible to have several browser windows associated with one instance of I.E. To quantify the memory savings of having multiple windows associated with one instance, a memory constrained environment was created on the desktop, and then two tests were performed. In one test, several instances of I.E. were launched, each with one browser window. In the other test, several browser windows were launched, all within the same instance of I.E. The tests revealed that launching additional browser windows within the same instance of I.E. consumed about ¼ MB of additional memory per browser window. In contrast, each additional instance of Internet Explorer consumed about 3 MB of additional memory. It was also noticed that when a new window was launched from an existing window, it came up much faster than when a new instance of I.E. was launched. As an added savings, the user is not prompted for a userid and password when the new window was launched from an existing window. However, the user is prompted for a userid and password each time a new instance of I.E. was launched.

To launch a new browser window within the same instance of I.E., select File -> New -> Window. The shortcut is Ctrl+N.

26.Conclusion

This paper has presented a sampling of guidelines and techniques for designing high performance web sites. Following these tips and hints will help you build a Web site that ensures optimal performance.

Appendix A References

[1] Sun on the net: Guide to Web Style. http://www.sun.com/styleguide/tables/Welcome.html
[2] Why Buffer http://www.learnasp.com/learn/whybuffer.asp
[3] TCP techniques:
http://support.microsoft.com/support/kb/articles/Q240/1/46.ASP?LN=EN-US&SD=gn&FR=0
[4] It's the Latency Stupid, http://rescomp.stanford.edu/cheshire/rants/Latency.html
[5] 3Com line test site: http://www.3com.com/56k/need4_56k/linetest.html
[6] Speed Tips by Charles Carroll, http://www.learnasp.com/learn/speedtips.asp
[7] ASP Fastcode, http://www.asplists.com/asplists/aspfastcode.asp)
[8] ASP flushing: http://www.learnasp.com//learn/speedtables.asp
[9] Getrows, http://www.learnasp.com/learn/dbtablegetrows.asp
[10] Getrows, http://www.learnasp.com/advice/whygetrows.asp
[11] GetString, http://learnasp.com/learn/dbtablegetstring.asp
[12] Disconnected recordsets: http://www.learnasp.com/learn/dbtabledisconnected.asp
[13] Speedtimer: http://www.learnasp.com/learn/speedtimer.asp
[14] Com properties, http://www.learnasp.com/learn/propertyexpense.asp
[15] VB components and/or recordsets stored at the session level:
http://www.learnasp.com/advice/dbsessionapp.asp
[16] VB components and/or recordsets stored at the session level:
http://www.learnasp.com/learn/nosessionobjects.asp
[17] VB components and/or recordsets stored at the session level:
http://www.learnasp.com/learn/sessionoverview.asp
[18] VB components and/or recordsets stored at the session level:
http://www.learnasp.com/learn/sessionoverview.asp
[19] VB components and/or recordsets stored at the session level:
http://www.learnasp.com/learn/buildvbthreads.asp
[20] VB components and/or recordsets stored at the session level:
http://www.learnasp.com/learn/globalproblems.asp
[21] Cache frequent query results. http://www.learnasp.com/learn/speedappdata.asp.
[22] Round-robin, /advice/roundrobin.asp .
[23] Tune your server, http://www.learnasp.com/advice/threads.asp).
[24] Tune your server, http://www.learnasp.com/learn/speedserver.asp
[25] Stop code, http://www.learnasp.com/learn/isclientconnected.asp
[26]Remote scripting: http://www.asplists.com/asplists/aspremotescripting.asp.
[27] Web Server Performance Guidelines, by Ed Tittel
http://sunsite.icm.edu.pl/sunworldonline/swol-09-1997/swol-09-webserver.html
[28] Web Server Technology, by Yeager and McGrath.
[29] Capacity Planning and Performance Modeling - From Mainframes to Client-Server Systems, by Menasce & Almeida:
[30] Keeping the 400lb. Gorilla at Bay Optimizing Web Performance**,** by James Rubarth-Lay.
http://eunuch.ddg.com/LIS/CyberHornsS96/j.rubarth-lay/PAPER.html
[31] Designing High Performance Web Pages, by Jaqui Lynch, Circle4 Computer Consultants
[32] 1996 and 1999 web surveys by Jakob Nielsen, www.useit.com/alertbox.
[33] CacheNow Campaign: http://vancouver-webpages.com/CacheNow
[34] J. Lynch – UNIX Performance Tuning – CMG96.
[35] Netscape palette: www.dsiegel.com/tips/wonk10/images.html.
[36] Gif tutorial: www.infohiway.com/way/faster/gif.html.
[37] Fastcgi: www.fastcgi.com.
[38] UKCMG paper by Adrian Johnson.

[39] Sun's New Web Design, by Jakob Nielsen.
[40] Fortune Magazine's list of top corporate Web sites:
www.fortune.com/fortune/buyersguide/websites/product1.html
[41] Web Site Garage: http://websitegarage.netscape.com/
[42] Gif Wizard: http://www.gifwizard.com/
[43] DeBabelizer: http://www.equilibrium.com/debab/index.html
[44] Improving ASP Application Performance, by J.D. Meier.
http://msdn.microsoft.com/workshop/server/asp/server03272000.asp
[45] Article by Wayne Plourde: http://www.asptoday.com/articles/20000113.htm
[46] Active Server Pages and COM Apartments: http://www.develop.com/dbox/aspapt.asp
[47] Improving Web Site Performance:
http://help.xbuilder.net/xbuilder/htmlhelp/general_whitepaper_improving_web_site_performance.htm
[48] XBuilder: http://www.15seconds.com/XBuilder.htm
[49] Kim Walker: kim@kimwalker.com
[50] Site Optimization Tutorial by Paul Boutin:
http://hotwired.lycos.com/webmonkey/98/26/index3a_page2.html?tw=design)
[51] Improving ASP Application Performance, by J.D. Meier, Microsoft Corporation, March 28, 2000, Appendix B: http://msdn.microsoft.com/workshop/server/asp/server03272000.asp
[52] Tips to Improve ASP Application Performance, by Srinivasa Sivakumar, Appendix C, http://www.15seconds.com/issue/000106.htm
[53] Server Performance and Scalability Killers, by George V. Reilly, Microsoft Corporation, February 22, 1999: http://msdn.microsoft.com/workshop/server/iis/tencom.asp
[54] Fireclick Blueflame: Accelerating the Web:
http://www.fireclick.com/products/whitepapers_1a.html
[55] Email from Craig Mason, August 16, 2000.
[56] SpaceAgent, http://www.insidersoftware.com/index.htm
[57] http://webreference.com/internet/software/http/compress.html
[58] http://webreference.com/internet/software/http/compression.html.
[59] http://www.microsoft.com/TechNet/iis/httpcomp.asp
[60] HTTP Performance Overview from W3 org http://www.w3.org/Protocols/HTTP/Performance/
[61] USA Today article: http://www.usatoday.com/life/cyber/tech/cti628.htm
[62] Jupiter Caching.com white papers: http://www.caching.com/whitepapers.htm

Appendix B Improving ASP Application Performance

From http://msdn.microsoft.com/workshop/server/asp/server03272000.asp **[51]**
J.D. Meier, Microsoft Corporation, March 28, 2000

Introduction
Developers often ask how to get more performance out of their ASP applications. Because I frequently review Active Server Pages (ASP) applications for performance, I figured I'd share the set of questions I ask to identify problems and recommend improvements. Some of these tips may seem like common sense, while others may be less obvious; however, all of the recommendations are backed by real world experience.

ASP Page Performance
Are you storing Visual Basic objects in Session or Application scope?
Visual Basic® and other Single-Threaded Apartment (STA) objects should be used only at page scope. Storing STAs in a Session variable locks the object down to the thread that created the object, defeating the purpose of a thread pool. Storing an STA in Application scope serializes access for all users. See the following articles for more information on this issue:

- Don Box's "ASP and COM Apartments"
- Q243543 "INFO: Do Not Store STA Objects in Session or Application"
- Q243548 "INFO: Design Guidelines for Visual Basic Components Under ASP"
- Q243815 "PRB: Storing STA COM Component in Session Locks to Single Thread"

Are you storing the Scripting.Dictionary in Session or Application scope?
The Scripting.Dictionary is Apartment threaded and should be used only at page scope, or your application will suffer serious serialization issues (see Q194803 "PRB: Scripting.Dictionary Object Fails in ASP Application Scope"). This limitation usually raises the question of what dictionaries can be used at Session or Application scope. Options are somewhat limited, but include the Commerce Dictionary and the LookupTable Object. See "Abridging the Dictionary Object: The ASP Team Creates a Lookup-Table Object."

Are your ASP scripts hundreds of lines long?
Script is interpreted line by line, so eliminating redundant script or creating more efficient script can improve performance. If you have hundreds of lines of ASP script in a single page, perhaps you can better partition your user, business, and data services.

ASP script is great for gathering input or formatting output, but when it comes to business and data services, components offer some additional benefits -- such as early binding and protection of intellectual property. In his article "Component vs. Component Part II," Jason Taylor discusses significant performance gains he found porting his ASP script to custom components, using Windows Script Components (WSC) as an intermediate step.

If you aren't using components, you can still partition your services with functions. For example, if your script is rendering several tables, make a generic function to generate your tables. You can then put these functions into includes, or lay the groundwork for a future port to components. The following sample illustrates using functions and includes for improved maintenance.

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

```
    Sub Main()
        WriteHeader
        WriteBody
        WriteFooter
    End Sub

    Sub WriteBody()
        ...
    End Sub

    Main   'call sub Main
</script>
```

Are your #include files too big?
There are no hard and fast rules for include-file sizes, but knowing how includes work can help you gauge whether you are using them efficiently. When ASP processes includes, it reads the entire file into memory. Because ASP will cache the entire expanded code (your page + the include), not just the functions you call, you may end up with large, inefficient namespaces that ASP must search when calling methods or looking up variables. Note that this process occurs for each page that uses the include. A good guide is to create fine-grained includes, so that you can be more selective about which pages will include them. For a more detailed explanation of how ASP processes includes, see "The Implications of ASP #include."

Are you using global variables?
Global variables increase the namespace used by ASP to retrieve values from memory. Variables declared within subroutines or functions are faster. For more information, see "25+ ASP Tips to Improve Performance and Style."

Does your script excessively intersperse ASP and HTML?
Keep blocks of ASP server-side script together, rather than switching back and forth between server-side and client-side code. This switching usually happens when concatenating HTML with simple values from ASP, as when you are writing out an HTML table:

```
<%   For iCtr = 1 to 10  %>
<TR><TD>Counting ... <%= iCtr %></TD></TR>
<%   Next  %>
```

You can improve the code's by making it a single script block:

```
<%   For iCtr = 1 to 10
Response.Write "<tr><td>Counting " & iCtr & "</td></tr>"
    Next
%>
```

This technique has shown measurable and significant performance improvements.

Are you buffering output?
Buffering is on by default in Windows 2000, but may be off if you've upgraded from Internet Information Server (IIS) 4.0. When buffering is on, ASP will wait until processing is complete before sending down the response, reducing network roundtrips and server-processing delays. When buffering is off, ASP waits for TCP acknowledgements from clients, which can really hurt performance, particularly over slow connections.

Note that while buffering may improve throughput, it may reduce perceived performance. If perceived performance is an issue, you can turn off buffering with **Response.Buffer = False** or you can call **Response.Flush**.

Are you using Session state?
ASP Sessions are a convenient facility, but they limit scalability. Sessions limit the scalability of a single box because they consume resources for each user. While Session size is largely determined by what you stuff into the Session variable, the real cost is resource contention. Sessions also limit your application's ability to scale out across multiple machines, because each Session is Web-server specific.

In many cases, you can avoid the use of Session objects by using hidden form fields or passing data in your QueryStrings (see Q175167 "HOWTO: Persisting Values Without Sessions". These approaches will allow you to scale out but may become more difficult to code as the structure of the data grows in complexity.

Another possibility is to use a database. This is suitable for more complex structures, such as a shopping cart. You can store your shopping cart in a database and look it up as required. Many large commerce sites take this approach by transmitting a unique ID to the user and looking up information from the database when needed. This approach will allow you to scale out transparently. It will also let your Web servers scale to more users, and it will make the carts persistent between requests. Many developers overlook this approach, because they believe the cost of reconnecting to the database will hurt performance, but that's where pooling comes into play (more on pooling below).

Have you disabled Session state if you aren't using it?
If the application doesn't rely on Sessions, disable Session state for the Web or virtual directory through the ISM (Internet Services Manager). Disabling Session state allows ASP to skip an extensive amount of source code, reducing overhead.

If you need Session state enabled for you site, you can prevent ASP from unnecessarily checking for Session information on specific pages by using the following page declarative:
@EnableSessionState = False
You might do this on pages that use frames. ASP serializes concurrent requests from the same session, causing ASP pages in frames to load sequentially. If you turn off Sessions with @EnableSessionState, frames will load concurrently.

Are you using third-party components designed for ASP?
Ask your third-party component vendor whether the component was both designed and tested for ASP. "Troubleshooting Components Under ASP Technology" provides some guidelines for ASP compliance.

Are you using some form of caching for data?
Caching is one of the most difficult aspects of Web application development, because it threatens the scalability of your application. Determining what to cache is based on data volatility and scope. Data that is static or used application-wide can be a good candidate for caching. Data that changes frequently or is user-specific would not be a good candidate.

Determining where in your application you should cache your data is based on your application goals, and you need to know the trade-offs. For example, caching an ActiveX® Data Objects (ADO) Recordset offers flexibility, because you can still grab an array or save the recordset as XML. However, if your application will be rendering the same HTML repeatedly, such as to display a list box of countries, you might cache the HTML string in Application scope, rather than just the data.

XML has certainly opened up new doors in terms of caching flexibility. For example, you might store data as XML strings and apply XSL transformation as required. For a more thorough investigation of this topic, see Paul Enfield's article "Using Server-Side XSL for Early Rendering: Generating Frequently Accessed Data-Driven Web Pages in Advance."

A good stress test early in your design can help you determine which caching technique to use.

Are you using CDO?
If you're using Windows NT® 4.0, use CDO-NTS and NT SP 5 or later (see Q214685 XFOR: CDONTS Not Thread-Safe, Crashes Under Stress). Collaboration Data Objects (CDO) was not tested for server-side use, except in the case of Outlook Web Access. In Windows 2000, use "CDO for Windows 2000, CDOSYS.dll, when possible, which is designed for server-side development. See the following articles:

- "Collaboration Data Objects Roadmap"
- Q195683 "INFO: Relationship between 1.x CDO Libraries and CDOSYS.DLL"
- Q177850 "INFO: What is the Difference Between CDO 1.2 and CDONTS?"

Are you redimensioning arrays?
It is generally more expensive to redimension arrays than it is to grab more than you need up front. Redimensioning arrays requires Visual Basic Scripting Edition (VBScript) to allocate space for the new array -- and, if you've used the Preserve modifier to preserve the contents of the array, to copy the data from the old array into the new. This means that not only are you spending extra processor cycles to redimension the array, but the process initially requires twice as much memory for the copy. However, if you allocate more space than you need initially (you need only five elements initially, but allocate space for 128), then adding more data to the array requires VBScript to insert only the new values into the existing array.

Are you using multiple languages on a page?
Multiple languages on a given page mean multiple script engines for that page. Script engines use Thread Local Storage (TLS), so multiple threads cannot use an instance of a script engine concurrently. Therefore, five simultaneous requests to the same page will cause ASP to instantiate five script engines. More engines means more overhead, so you may be able to gain some performance by limiting your number of languages used on a given page.

Are you checking Response.IsClientConnected before processing long routines?
By testing **Response.IsClientConnected** your application can avoid wasting CPU cycles by quitting methods if the client is no longer connected. Note that IIS 5.0 overcomes a limitation in IIS 4.0 (the need to send content to the browser before checking the property). For sample code, refer to Q182892 "HOWTO: Use IsClientConnected to Check If Browser is Connected."

Are you using Server.MapPath unnecessarily?
When you request **Server.MapPath**, you are generating an additional request for the server to process. To improve performance, replace **Server.MapPath** with a fully qualified path when deploying your Web site.

Are you parsing strings?
Use regular expressions in validation routines, in formatting functions, and instead of looping through strings. See "String Manipulation and Pattern Testing with Regular Expressions" for practical advise on using regular expressions.

Are you using the same object many times?
VBScript 5.0 provides the **With** statement. The **With** statement allows you to execute a series of statements on a specific object without requalifying the name of the object. For more information, see the Windows Script Technologies Web site .

Does the global.asa contain empty Session_OnStart or Session_OnEnd methods?
Stripping out empty Session events reduces the amount of source code that ASP must traverse, and improves performance.

Component Performance
Are you storing your objects in Session or Application scope?
Storing references to objects in ASP's Session or Application scope will cause many performance and scalability issues if those objects aren't designed to be shared across threads or activities. Only agile components -- or, in Windows 2000, components marked Neutral -- can be referenced in Session or Application variables with direct access by client threads. Components are considered agile if they are marked Both and aggregate the free-threaded marshaler (FTM). See "Agility in Server Components" for a discussion on how to aggregate the FTM. Components with other threading models impose restrictions when stored in Session or Application scope.

As mentioned earlier, Visual Basic or other STA components should only be used within page scope. Note that solidly written Visual Basic components can perform extremely well if you follow this rule. Single- and Free-threaded components are not recommended, because of security issues and expensive proxy/stubs that get created when marshaling across apartment boundaries.

See the following articles for additional information:
- "ASP Component Guidelines"
- Q243544 "INFO: Component Threading Model Summary Under ASP" Q150777 "INFO: Descriptions and Workings of OLE Threading Models"

Are you concatenating strings in components?
Use fixed-length strings in Visual Basic for string concatenation. Don't just keep adding to a string, or you'll reallocate it multiple times -- and reallocation is expensive.

```
    Bad:
        Public Function BadConcatenation() As String
          Dim intLoop As Integer
          Dim strTemp As String      'this will be expensive

          For intLoop = 1 To 1000
            strTemp = strTemp & "<tr><td>Counting "
            strTemp = strTemp & CStr(i)
            strTemp = strTemp & "</td></tr>"
          Next intLoop

          BadConcatenation = strTemp
        End Function
    Good:
        Public Function GoodConcatenation() As String
          Dim intCtr As Integer
          Dim intLoop As Integer
          Dim strTemp As String * 32000   'this improves performance

          intCtr = 1

          For intLoop = 1 To 1000

            Mid(strTemp, intCtr) = "<tr><td>Counting "
            intCtr = intCtr + 17

            Mid(strTemp, intCtr) = CStr(intLoop)
            intCtr = intCtr + Len(CStr(intLoop))

            Mid(strTemp, intCtr) = "</td></tr>"
            intCtr = intCtr + 10
          Next intLoop

          GoodConcatenation = strTemp
        End Function
```

Several large-scale sites found major performance gains from this tip alone. If you've got extensive string concatenation in your application, put this technique high on your priority list. Experiment with different string sizes, and test to see which size will work best for your particular routine.

Are you using SQL Server for your middle-tier cache?
If you need to cache data that is read frequently and seldom updated in the middle-tier, use SQL Server rather than a roll-your-own solution. SQL Server provides high-performance middle-tier caching. For more information, read "Middle-Tier High-Speed Data Caching Involving COM/MTS and COM+."

Are you using transactions when you don't need to?
Transactions provide a service -- and that service can add a performance hit. Evaluate whether methods actually need transactions. For example, if you're grabbing a recordset to hand off to a browser client for reading data, you don't need a transaction. By factoring out operations that read data into separate components from operations that perform updates, you have more flexibility in marking your components for transactions. The following articles provide scalable patterns that you can use as guidelines to help you factor out your own components:

- "Scalable Design Patterns"
- "Simplify MTS Apps with a Design Pattern"
- "FMStocks Application: Start Here"

Are you calling SetComplete/SetAbort in each method?
Calling **SetComplete** and **SetAbort** in each method of your Microsoft Transaction Server (MTS) components will release resources earlier, and will ensure that a component does not live outside the scope of its transaction. In Windows 2000, COM+ provides the setting 'Automatically deactivate this object when this method returns,' which performs the equivalent code. You can enable this setting on a per-method basis in the Component Services console.

This performance gain is more applicable to large-scale sites, but is also good form.

Are you creating child MTS components with CreateInstance?
In Windows NT 4.0, use **CreateInstance** to create child MTS components. Create non-MTS components using **CreateObject** or **New** (unless the components are in the same Visual Basic project). Other combinations cause code paths and checks that could otherwise be avoided. For more information, see Ted Pattison's article "Creating Objects Properly in an MTS App."

In Windows 2000, this distinction between creating objects with different techniques goes away, and you simply use **CreateObject** (see Q250865 "INFO: CreateObject and CreateInstance Have the Same Effect in COM+".

Are your components in Server packages or Library packages?
Library packages run in their caller's process, while components in a Server package run in a new process. As such, Library packages do not have the cross-process performance hit that Server packages have. A typical recommendation is to run the Web application out of process, but have the components in a Library package to provide both process isolation and high performance. In situations where you need to use a Server package, as when you are calling remote MTS/COM+ components, take steps to minimize cross process overhead.

Are you crossing processes effectively?
Minimizing marshaling overhead and reducing network calls are keys to improving performance of distributed applications. Use early binding in your components to minimize expensive network round trips by eliminating the extra call to **GetIdsOfNames** that late binding incurs. You can further reduce network trips by bundling your parameters into arguments for method calls, instead of setting a bunch of properties individually. Rather than pass parameters **ByRef**, pass **ByVal** where you can to minimize marshaling overhead.

Are you using remote components?
Many developers find themselves in situations where they need to call remote components. All of the rules for crossing processes effectively apply. An additional recommendation is to use an intermediary package (see Q159311 "Instantiating Remote Components in MTS and IIS". Using an intermediary package to call the remote component avoids some security complications and allows you to use early binding.

Do your client and server machines use the same protocols in DCOM?
A common cause of activation delays of remote components results from the client and server machines having different DCOM protocol lists. Use DCOMCNFG.EXE on each machine to match the protocol sequence. For example, if both machines are using TCP/IP, move TCP/IP to the top of the list on both machines. A few guidelines apply when modifying your DCOM protocol list:

- Move TCP/IP to the top when you can.
- Remove protocols that you don't need.
- Any changes to the protocol list require a reboot.

Are you using ASP built-in objects from a remote component?
Don't. Even if you can figure out how to marshal the ASP built-in objects (Request, Response, and so forth) across machines, the performance cost outweighs any benefits.

Are you using Visual Basic?
Visual Basic performs extremely well when you have the right version, use the right settings and follow good design guidelines:

- Use Visual Basic 6.0 Service Pack 3.
- Set Unattended Execution and Retain in Memory in your projects (See Figure below). See Q186273 "BUG: AV Running VB-Built Component in Multi-Threaded Environment" or additional information.
- Read Q243548 "INFO: Design Guidelines for VB Components Under ASP" to avoid many common development mistakes.

ARE YOU USING MFC?
Active Template Library (ATL) components are lighter than Microsoft Foundation Classes (MFC) and are preferred for server components. MFC code is heavy for the server and may bring about unexpected serialization for state management. Also, you cannot create components marked Both that aggregate the FTM. MFC only produces STA objects, which are limited to page scope.

Are you using Java?
Make sure you have the latest Virtual Machine from http://www.microsoft.com/java/ and read Q232368 "PRB: Java Threads Blocking when Accessing COM Objects."

Are you waiting for stuff that could be done asynchronously?
If your ASP application is making long database calls or calling components that are waiting on other components to raise events, consider making the call asynchronous through a queueing mechanism, such as MSMQ. For information on MSMQ, see the following:

- Q173339 "HOWTO: Use MSMQ from an ASP Page"
- Q181839 "Mqasp.exe MSMQ Basic Queue Operations Using IIS/ASP"
- Q243546 "PRB: ASP Does Not Support Events"

Are you looping through large datasets in the middle tier?

Push more of these large or complex data operations to stored procedures where you can.

Data Access Performance
Are you using indexes in your database?
Indexes provide immediate impact on your application's performance. Poor indexes will slow your application to a crawl, while good indexes will help optimize your application's performance. For information on tuning indexes, see "Top Ten Tips: Accessing SQL Through ADO and ASP." or SQL Books Online.

Are you calling stored procedures rather than dynamic SQL?
Using stored procedures prevents your database from having to recompile your SQL statements repeatedly. Use stored procedures or parameterized SQL strings.

Are you returning just the required data?
Check your SELECT statements to ensure that you're returning only the required columns and only the necessary rows. If you have queries that can potentially return a lot of records, consider paging through your recordsets. See the following articles for more information:

- "Recordset Paging with ADO 2.0"
- "Ad Hoc Web Reporting with ADO 2.0"
- "6 Ways to Boost ADO Application Performance"

Are you using DAO or RDO?
Remote Data Objects (RDO) and Data Access Objects (DAO) are intended for a single-client application process, and weren't tested for the Web. ADO is designed and tested for Web use.

Which version of MDAC are you using?
Updated versions of MDAC provide improved reliability and performance. You should be using MDAC version 2.1 Service Pack 2 or later. MDAC 2.5 is recommended, because it's the most stable and it's been tested extensively. If you don't know which version you have on your box, you can grab the Component Checker tool from http://www.microsoft.com/data/

Are you following MDAC best practices?
See "Improve MDAC Application Performance."

Are you pooling connections?
Pooling allows you to reuse the effort of connecting to a database. ODBC Connection pooling is on by default in MDAC 2.0. In MDAC 2.1 or later, OleDb Session pooling is the default. Remember that in order for pooling to work, the user name, password, and resource in the connection string need to match (it's a byte-by-byte comparison).

See the following articles for additional information on pooling:

- "Pooling in the Microsoft Data Access Components"
- Q169470 "INFO: Frequently Asked Questions About ODBC Connection Pooling"
- Q187874 "CnPool.exe Test Connection Pooling with Tempdb Objects"
- Q191572 "INFO: Connection Pool Management by ADO Objects Called From ASP"

Are you storing ADO connection in Session or Application scope?
This defeats the purpose of connection pooling and creates resource contention. Create connection at page scope or within the functions that need them, and set the connections to nothing to free the connection back to the pool.

Are you explicitly closing Recordset and Connection variables?
Recordsets need to be closed if they are going to be reused (but reusing recordsets is discouraged). Closing Connection variables as soon as you can releases them back to the pool, so that they can be pooled for reuse. It is always good practice to explicitly close your object variables.

Are you reusing Recordset and Command variables?
Create new Recordset and Command variables rather than reusing existing ones. This won't necessarily improve your application's performance but it will make your application more reliable and easier to maintain. See Q197449 "PRB: Problems Reusing ADO Command Object on Multiple Recordsets" or more information.

Are you disconnecting the recordsets?
Disconnecting recordsets frees the Connection object back to the pool, allowing the Connection to be closed and reused sooner.

Are you using the right cursor and lock-type for the job?
Use "Firehose" (forward-only, read-only) cursors when you need to make a single pass through the data. Firehose cursors, the default in ADO, provide the fastest performance and have the least amount of overhead. See ADO documentation for more information on cursor and lock types.

Are you using DSN-less connections?
In general, DSN-less connections are faster than System DSNs (data source names), which are faster than File DSNs.

Are you using Access?
Microsoft Access is a file-based database, so don't expect it to perform well with concurrent users under IIS.

Are you using SQL Server?
Use SQL Server 7.0. SQL Server 7.0 is superior to earlier versions of SQL Server, and provides row-level locking, as well as other performance benefits. You've heard SQL scales -- but for proof, see http://www.tpc.org/ and read Jim Gray's paper at http://research.microsoft.com/scalable/

Are you using TCP/IP for your network library?
Use TCP/IP for your network library for performance and scalability.

Are you using the OLEDB SQL provider?
The SQLOLEDB, the SQL provider, is recommended over MSDASQL, the OLEDB provider for ODBC for performance and reliability.

Are you using Oracle?
Oracle performance really depends on a combination of the right MDAC bits with the correct Oracle client patches. If you're using Oracle, take the time to read the following articles:

- "Microsoft OLE DB Provider for Oracle: Tips, Tricks, and Traps"
- "Fitch & Mather Stocks: Data Access Layer for Oracle 8"

If you're using Oracle and MTS, be sure to review the following articles:

- Q193893 "INFO: Using Oracle Databases with Microsoft Transaction Server"
- Q191168 "INFO: Failed to Enlist on Calling Object's Transaction"

Are you using the same database for reporting and transactions?
Many large Web sites maintain separate databases for read-only and transactional data as an effective way to boost performance. This has the added benefit of allowing you to design your database schema to be optimized for reporting.

IIS Settings
Is ASP debugging enabled?
Check the ISM. If ASP debugging is enabled, the application is locked down to a single thread of execution. See Q216580 "PRB: Blocking/Serialization When Using InProc Component (DLL) from ASP."

Is ASP configured to have enough threads/script engines?
Read "The Art and Science of Web Server Tuning with Internet Information Services 5.0" and "Navigating the Maze of Settings for Web Server Performance Optimization."

Are you using SSL?
Secure Sockets Layer (SSL) is expensive in terms of bandwidth and CPU usage. If you're using SSL, it's because of security needs. Your best bet is to restrict SSL usage to where you need it, and keep the pages simple.

Stress Testing
It's a common misconception that performance equals scalability. Performance for ASP means the rate at which pages can be served. Scalability is measured by how much performance degrades under additional load. To put these terms in perspective, your ASP application may perform well with 10 users, but does not scale to 1000 users, because performance becomes unacceptable. Use stress testing to measure the

performance and scalability of your ASP application. For more information on scalability of WinDNA applications, see "A Blueprint for Building Web Sites Using the Microsoft Windows DNA Platform."

What are your performance expectations?
Performance can be measured in terms of the number of ASP requests per second that your servers can handle. Using ASP's performance counter, ASP Requests Per Second, you can set benchmarks to measure against.

Are you trying to stress test with a browser?
Stress testing with a browser may be throwing off your results. As mentioned earlier, ASP will serialize concurrent requests from the same Session. For example, if Cookies are enabled and you're hitting the server from one machine, then those requests will serialize. Use the Web Application Stress tool (WAS -- formerly known as Homer). For an introduction to WAS, see "I Can't Stress It Enough -- Load Test Your ASP Application."

Have you performed end-to-end testing?
While it's important and feasible to stress test your database, components, and ASP layers separately, end to end testing is how you'll find your application's real bottlenecks.

Conclusion
Reviewing ASP applications for performance means taking a look at several things. By breaking the application down into its various layers, you can build a framework for analyzing performance issues. While there are many guidelines and recommendations, nothing replaces stress testing your application and using sound judgment.

Appendix C Tips to Improve ASP Application Performance

From http://www.15seconds.com/issue/000106.htm **[52]**
By Srinivasa Sivakumar

Introduction
Performance tuning can be tricky. It's especially tough in Internet-related projects with lots of components running around, like HTML client, HTTP network, Web server, middle-tier components, database components, resource-management components, TCP/IP networks, and database servers. Performance tuning depends on a lot of parameters and sometimes, by changing a single parameter, performance can increase drastically.

Performance Issues
ASP application performance involves two parts:

- HTML page performance
- Response time

Further, we can boil down response rime to:

- ASP page performance
- Network bandwidth
- Database issues

Let's see each of them in detail.

HTML Page Performance
HTML page performance is purely a client problem. This problem can be related to the client's hardware and the bandwidth. Apart from these, there are few other factors that can affect the page performance.

As we all know, the smaller the file size, the better the performance. Here are a few elements that can reduce the HTML file size, which will improve the page load performance in the browser.

- Avoid lot of images. When a browser requests a page, it makes N number of calls to the Web server to display N number of images. This will slow down the page load process. When you can't avoid multiple images, try to show the images as thumbnails.
- Frames are another elements that will slow down the load process. For frames also, there will be N number of requests to display N number of frames.
- If you can avoid tables, that's good news. But we can't always do that, so try to avoid nested tables. This will boost the load performance.
- If you know your users, you can design better pages. Well, this is not possible in the Internet. But, if you are developing an Intranet product for a corporation, then the corporation will have a standard browser. For example, if the corporation's standard browser is a 4.0 browser then you can avoid some complex tables and you can use Cascading Style Sheets to position your HTML elements.
- Avoid redundant tags. Let's take the following example:
  ```
  <Body><br>
  <P><font face="Verdana" size="4"><br>
  </font></P><br>
  <P><font face="Verdana" size="4"><br>
  </font></P><br>
  <P><font face="Verdana" size="4"><br>
  ```

```
</font></P><br>
</Body><br>
```

You can avoid the <font> tag
Source 2:

```
<Body><br>
<font face="Verdana" size="4"><br>
<P><br>
</P> <br>
<P> <br>
</P><br>
<P> <br>
</P><br>
<font> </Body><br>
```

- Avoid lot of jazzy comments. This will reduce the file size. I hate to leave comments in the HTML page. This will allow the others the view my page and understand my style.
- Avoid long file names and use relative path names to identify other files.
- Even you can reduce the formatting and indenting for the HTML page. This will minimize the page size.
- Some of the WYSIWYG HTML editors will put lot of unwanted HTML tags in the page. So, it's a good idea to clean up the unwanted tags, once you have designed your page with the WYSIWYG editor.
- Avoid Java Applets in the HTML pages, when they are not needed. For example, if you are only using the Java Applets for animations, then try to use animated gif files or a Flash animation. They are faster than Java Applets.

ASP Page Performance

1. Reading from the object variable is always slower than reading from the local variable. So, if you are going to read from the object variable often, then store it in a local variable and access it.

   Slower:

   ```
   if Myobj.Value = 0 then
   Do something
   elseif Myobj.Value > 0 then
   Do something
   elseif Myobj.Value < 0 then
   Do something
   end if
   ```

   Faster:

   ```
   MyVar = Myobj.Value
   if MyVar = 0 then
   Do something
   elseif MyVar > 0 then
   Do something
   elseif MyVar < 0 then
   Do something
   end if
   ```

2. If you are using VBScript 5.0 or later, then you can use the With ... End With statements.

   Slower:

   ```
   Myobj.FirstName = "Srinivasa"
   Myobj.LastName = "Sivakumar"
   Myobj.City = "Chicago"
   ```

   Faster:

   ```
   With Myobj
       .FirstName = "Srinivasa"
       .LastName = "Sivakumar"
       .City = "Chicago"
   End with
   ```

3. Before doing a bulky ASP code processing, you can check to make sure Response.IsClientConnected. If not, don't process the page. This will reduce the load on the server.
4. As always, avoid session variables because each ASP page runs in a different thread and session calls will be serialized one by one. So, this will slow down the application. Instead of session variables you can use the QueryString collection or hidden variables in the form which holds the values.
5. Still you think you can't avoid the session variables. And if you have lot of session variables, then it is better to use the dictionary object.
6. Disable the session access, if possible, with <%@ EnableSessionState=False %>.
7. Enabling buffering will improve the performance, like Response.Buffer=True.
8. Wrap all your data-access code inside a COM component. In this way you can take advantage of speed of the compiled and multithreaded. As you know, creating a database connection takes lots of system resources and time. You can avoid this time-lag by taking advantage of the connection pooling, when your component runs inside the Microsoft Transaction Server (MTS). MTS is a Windows NT-based technology that when used with Distributed COM (DCOM), lets you build COM objects that are more easily distributed across a network than when using DCOM alone
9. Avoid multiple calls to the COM component. For example, if you want to write 10 values to a COM component, you will do it with 10 calls to the component. However, if you can do it with one call, then that will improve the performance.

   Slower:

   ```
   Myobj.FirstName = "Srinivasa"
   Myobj.LastName = "Sivakumar"
   Myobj.City = "Chicago"
   ```

   Faster:

   Instead of exposing 3 properties from the COM component, you can expose a single property called "Value," and you can take advantage of the single call.

   Let's see how the code will look in the COM component:
   Declare the private variables to hold the values of the properties. Declare three more constants to hold the property value position in the array.

   ```
   Option Explicit

   'Local variables to hold property values
   Private mvarFirstName   As String
   Private mvarLastName    As String
   ```

```
    Private mvarCity        As String

    'Property Value Constants
    Private Const CN_FirstName = 0
    Private Const CN_LastName = 1
    Private Const CN_City = 2
```

Add the "Value" property procedure.

```
Public Property Let Value(ByVal vData As Variant)

        On Error GoTo ErrHand

        'Check if the parameter value is an array
        If IsArray(vData) = False Then
            Err.Raise 100, App.Title & " - Property Let: Value", "Invalid property value.
    An array is expected."
            GoTo CleanExit
        Else
            mvarFirstName = vData(CN_FirstName)
            mvarLastName = vData(CN_LastName)
            mvarCity = vData(CN_City)
        End If

    CleanExit:
        Exit Property
    ErrHand:
        Err.Raise Err.Number, Err.Source, Err.Description
        Exit Property
    End Property
```

Here is how you will call the method from the ASP page.

```
    Myobj.Value = Array("Srinivasa", "Sivakumar", "Chicago")
```

Even you can write the Get property procedure to pass the data from the component to the ASP application with a single call.

Note: Always have error handlers in all your functions, subs, properties, etc. This will avoid the whole system going down when an error occurs in the component. Even an error in a component could bring all the applications down in the same memory space. It could bring the Internet Information Server (IIS) down, if the component is running in the IIS memory space.

10. Never, ever declare a COM component such as an ADO connection object with the application or session scope. This will also hit the performance because of the threading each call has to be marshaled between threads.
11. The Apartment-Threaded COM component objects are only good for page scope. If you want to access the COM objects across the page, then consider the free-threaded or both threaded components.
    Note: You can only develop Apartment-Threaded components in Visual Basic. If you are serious about the free-threaded component, then you have to develop them in VC++ or in Java.
12. Do not use OnStartPage and OnEndPage methods to access the ASP intrinsic. These methods are provided for legacy support with IIS 3. Instead of this, use the ObjectContext with Implements keyword.

13. When you are accessing the component across the process or machine, pass the variables to the objects as By Val. This will reduce the Marshaling overhead.
14. When you have more than 100 lines of code in ASP, it is advisable to move that code to a COM component. ASP scripts are interpreted at run time and COM components are compiled.
    Note: There is a time-lag for the component to be initialized, up and running. This could also affect the application performance, so be careful before making the decision between the ASP script and COM component.
15. Never use components such as Microsoft Word or Excel to manipulate the data. They are out-of-process components and they are not optimized for performance.
16. The other advantage of the component over the script is early binding vs. late binding.
17. As a Web developer, we always like to create a large Include file that will include all the global code and constants. The problem with this approach is some of the code or the declaration will not be used in every page. So for each and every page these Include files will be processed and this will definitely slow down the page performance.
18. Avoid multiple Request.Write statements and group them in to few.

Slower:

```
’Add a default Select Word
Response.Write " <option value=""0"">(Select a Program Manager)</option>"

¨         Do While Not objRs.EOF
Response.Write " <option value=" & objRs!ProjManKey & ">" & objRs!ProjMan
& "</option>"
objRs.MoveNext
Loop
Response.Write "</font> </select>"
```

Faster:

```
Dim strHTML

'Add a default Select Word
strHTML =  " <option value=""0"">(Select a Program Manager)</option>"

Do While Not objRs.EOF
strHTML = strHTML & " <option value=" & objRs!ProjManKey & ">" &
objRs!ProjMan & "</option>"
objRs.MoveNext
Loop
strHTML = strHTML &  "</font> </select>"

Response.Write strHTML
```

Network Bandwidth Issues

1. In most cases network performance can ruin the ASP performance. Use 10/100 network cards for better performance.
2. If your Web server and the database server are running in the same server, then it is advisable to move them into different servers.
3. Even you can introduce a new middle-tier server to handle the COM components with MTS. This decision totally depends on the load on the Web and MTS servers.

Database Issues

Well, a good database design can contribute a lot to the performance. Database design issues are out of the scope of this article. Let's see few points that could improve the database performance.

1. Whenever you access the database, avoid the dynamic SQL and use stored procedures or database views. Before finalizing the SQL statement for the stored procedure or view, analyze the SQL query and make sure it is optimized, and also make sure that it uses the proper indices for record scanning.

    Note: If you are using SQL Server 7 or later. then you can take advantage of the visual output of the "Query Analyzer."

2. The join type used in the SQL statement will also increase or decrease the performance of the query.
3. When using the ADO Recordset objects, use the proper cursor type and lock type. For example, if you are going to fill a combo box, then you can use the cursor type adOpenForwardOnly and lock type adLockReadOnly.
4. Sometimes allocating appropriate database buffers will increase the performance. For example, if you are working with the Oracle database, for each connection you should have three sessions open. If you can fine-tune these things, you'll get drastic performance increase.
5. If your site has lot of hits and your server can't handle the entire load, then replicate the ASP application and the database.

Note: While designing the database, you have to make appropriate design considerations for database replication. When we are replicating the databases, few elements will not be replicated. For example, if you are using Oracle as the database server, Database Sequences will not be replicated. So, if you are rely on Database Sequences for your primary key, then you are in trouble.

Summary

When we talk about ASP performance, there are lots of factors and I have discussed each of them in detail. But don't think that if you can fix all these factors, your ASP performance will increase. All these tips will not suit each and every ASP application. We have to consider them in application by application basis.

Well, make some time to read the book "Microsoft Internet Information Server Resource Kit" from MS Press.

Try these following for more information:
Improving Web Site Performance
Server Performance
Microsoft Web Capacity Analysis Tool
Simulating Cookies with the Munger

Appendix D Server Performance and Scalability Killers

From http://msdn.microsoft.com/workshop/server/iis/tencom.asp **[53]**
George V. Reilly, Microsoft Corporation, February 22, 1999

The Ten Commandments of Killing Server Performance
Following each of these "commandments" will effectively hamper the performance and scalability of your code. **In other words, these commandments should be broken whenever possible!** I explain why and how to break them in order to improve performance and scalability.

1. Thou shalt allocate and free lots of objects.
You should avoid excessive allocation, because memory allocation can be costly. Freeing blocks of memory can be even more costly, as most allocators attempt to coalesce adjacent blocks of freed memory into larger blocks. Until the release of Windows NT® 4.0 service pack 4, the system heap often performed badly in multithreaded processes. The heap was protected by a single global lock and scaled negatively on multiprocessor systems.

2. Thou shalt not think about processor cache.
Most people know that hard page faults caused by the virtual memory subsystem are expensive and best avoided, but still believe that all other memory access patterns are equally good. This has not been true since the 80486 was introduced. Modern CPUs have become so much faster than RAM that they need at least two levels of memory cache. On Pentium-class machines, the fast L1 cache holds 8 KB of data and 8 KB of instructions, while the slower L2 cache holds several hundred KB of mixed code and data. A reference to a memory location found in the L1 cache costs one cycle, references to the L2 cache cost four to seven cycles, while references to main memory cost dozens of processor cycles. The latter figure will soon exceed 100 cycles. In many ways, the caches are like a small, fast, virtual memory system.

The fundamental unit of memory as far as the caches are concerned is not a byte, but a cache line. Pentium cache lines are 32 bytes wide, Alpha cache lines are 64 bytes wide. This means that there are only 512 slots for code and data in the L1 cache. If data that is used together (temporal locality) is not stored together (spatial locality), it can lead to poor performance. Arrays have excellent spatial locality, while linked lists and other pointer-based data structures tend to have poor locality.

Packing data into the same cache line usually helps performance, but it can also hurt performance on multiprocessor systems. The memory subsystem works hard to keep the caches coherent between processors. If a read-only datum, used by all of the processors, shares a cache line with a datum that is being updated frequently by one of the processors, the caches will spend a lot of time updating their copy of the cache line. This high-speed game of Ping-Pong is often referred to as "cache sloshing." If the read-only data were in a different cache line, this sloshing could be avoided.

Optimizing code for space is more effective than optimizing for speed. Smaller code fits in fewer pages, leading to a smaller working set and fewer page faults, and it fits in fewer cache lines. However, certain core functions should be optimized for speed. Use a profiler to identify these functions.

3. Thou shalt never cache frequently used data.
Software caching can be used by all kinds of applications. When a calculation is expensive, you store a copy of the result. This is the classic space-time tradeoff: Sacrifice some memory to save some time. If done well, it can be extremely effective.

You must cache judiciously. If you cache the wrong data, you're wasting memory. If you cache too much, you'll have less memory for other operations. If you cache too little, the cache will be ineffective because you'll have to recalculate the data missed by the cache. If you cache time-sensitive data for too long, it will become stale. Servers generally care more about speed than space, so they cache more aggressively than desktop systems. Be sure to invalidate and flush unused cache entries periodically; otherwise you'll have working set problems.

4. Thou shalt create lots of threads. The more, the merrier.
Tuning the number of threads active in a server is critical. If the threads are I/O-bound, they'll spend much of their time blocking, waiting for I/O to complete — and a blocked thread is a thread that's not doing useful work. Adding additional threads can increase throughput, but adding too many threads will decrease server performance, because context swapping will become a significant overhead. The rate of context switches should be kept low for three reasons: context switches are pure overhead and contribute nothing useful to the application's work; context switches use up valuable cycles; and worst of all, context switches leave the processor's caches full of stale data, which is expensive to replace.

Much depends upon your threading architecture. One thread per client is almost never appropriate, as it does not scale well to large numbers of clients. Context switching becomes intolerable and Windows NT runs out of resources. A thread pool model, where a pool of worker threads consumes a queue of requests, works better. In the future, you will no longer need to write your own thread pool, because Windows 2000 provides suitable APIs, such as **QueueUserWorkItem**.

5. Thou shalt use global locks for data structures.
The easiest way to make data thread-safe is to surround it with one big lock. For simplicity, make everything use the same lock. There's a problem with this approach: *serialization*. Every thread that needs to manipulate the data is going to have to wait in line to acquire the lock. If a thread is blocked on a lock, it's not doing useful work. When the server is under light load, this is seldom a problem, because only one thread is likely to want the lock at a time. Under heavy load, a high-contention lock can become a huge problem.

Consider an accident on a multilane freeway, where all the traffic has been diverted into one lane. If traffic is light, the diversion will have a negligible effect on the rate of traffic flow. When traffic is heavy, the traffic jam will stretch for miles as the cars slowly merge into the single lane.

Several techniques can reduce lock contention.
Don't be overprotective, that is, don't lock data when you don't have to. Hold locks for just as long as you need them, and no longer. It is important not to hold locks unnecessarily around large sections of code or in frequently executed sections of code.
Partition your data so that it's protected by a set of disjoint locks. For example, a symbol table could be partitioned on the first letter of the identifiers, so that modifying the value of a symbol whose name starts with *Q* does not interfere with reading the value of a symbol whose name starts with *H*.
Use the Interlocked family of APIs (**InterlockedIncrement**, **InterlockedCompareExchangePointer**, etc.) to atomically modify data without acquiring a lock.
Use multi-reader/single-writer locks when the data isn't modified often. You'll get better concurrency, though the lock operations will be more expensive and you may risk starving writers.
Use spincounts on critical sections. See the **SetCriticalSectionSpinCount** API introduced in the Windows NT 4.0 service pack 3.
Use TryEnterCriticalSection and do some other useful work if you can't acquire the lock.

High contention leads to serialization, which leads to low CPU utilization, which encourages users to add more threads, which makes matters even worse.

6. Thou shalt not pay attention to multiprocessor machines.
It can be a nasty surprise to find that your code performs worse on a multiprocessor system than it does on a uniprocessor system. The natural expectation is that it will perform *N* times better on an *N*-way system. The reason for the poor performance is contention: lock contention, bus contention, and/or cache line contention. The processors are fighting over the ownership of shared resources instead of doing productive work.

If you're at all serious about writing multithreaded applications, you should be stress-testing and performance-testing your application on multiprocessor boxes. Uniprocessor systems provide a coarse-grained illusion of concurrency by time-slicing the execution of threads. Multiprocessor boxes have true concurrency, and race conditions and contentions are far more likely to arise.

7. Thou shalt use blocking calls all the time; they are fun.
Synchronous blocking calls to perform I/O are appropriate for most desktop applications, but they're a poor way to exploit the CPU(s) on a server. I/O takes millions of cycles to complete; cycles that could be put to good use. You can achieve significantly higher rates of client request and I/O throughput by using asynchronous I/O, at a cost of additional complexity.

If you have blocking calls or I/O operations that can take a long time, you should think about how many resources you want to commit. Do you want all the threads to be used, or only a limited set? In general, it is better to use a limited set of threads. Build a small thread pool and a queue, and use the queue to schedule work for the threads to complete the blocking calls. This will enable your application to leave other threads available to pick up and process new, non-blocking requests.

8. Thou shalt not measure.
When you can measure what you are speaking about, and express it in numbers, you know something about it; but when you cannot express it in numbers, your knowledge is of a meager and unsatisfactory kind; it may be the beginning of knowledge, but you have scarcely in your thoughts advanced to the state of science.
Lord Kelvin (William Thomson)

Without measurements, you do not understand your application's performance. You are groping blindly, making half-informed guesses. You have not identified your performance problems and you cannot begin to make any improvements or do capacity planning.

Measurement encompasses black-box measurement and profiling. *Black-box measurement* means gathering the numbers exposed by performance counters (memory usage, context switches, CPU utilization, etc.) and external testing tools (throughput, response times, etc.). To *profile* your code, you compile an instrumented version of your code and run it through various scenarios, gathering statistics on execution time and procedure call frequency.

Measurement is of little use if it is not complemented by analysis. Measurement will tell you that you have problems and it may even help you isolate where those problems are, but it won't tell you *why* you have problems. Analyze the problems so that you can determine how to fix them properly. Address the underlying problems and not just the symptoms.

When you make your changes, measure again. You need to learn if your changes were effective. The changes may unmask other performance problems, and the cycle of measure-analyze-fix-measure will begin anew. You must also measure regularly to catch performance regressions.

9. Thou shalt use single-client, single-request testing.
A common mistake for people writing ASP and ISAPI applications is to test their applications by using a single browser. When they deploy their applications on the Internet, they discover that they cannot cope with high loads and that they have miserable throughput and response times.

Testing with a browser is necessary, but not sufficient. If the server does not respond quickly enough, you know you have a problem. But even if it is snappy when you're using a single browser, you don't know how well it copes with load. What happens if a dozen clients are making requests simultaneously? Or a hundred? What sort of throughput can your application sustain? What sort of response time does it provide? What are these numbers like when your application is under light load? Medium load? Heavy load? How well does your application scale if you run it on a multiprocessor machine? Stress testing your application is essential to shake out bugs and it's essential to discover performance problems.

Similar considerations about load testing apply to all server applications.

10. Thou shalt not use real-world scenarios.
It's easy to fall into the trap of tuning your application for a few specific, artificial scenarios (such as benchmarks). It's important to pick a broad spectrum of scenarios that correspond to real-world usage and optimize for a broad range of operations. If you don't find these scenarios, your users and reviewers certainly will, and they'll pan you for it.

Conclusion
We've learned a few things about killing server performance and scalability since we started developing IIS. Writing high-performance server applications is not easy. In addition to the traditional performance issues that arise when writing desktop applications, you must pay special attention to memory allocations, cache lines, caching data, thread proliferation, locking strategies, multiprocessor machines, blocking calls, measurement and analysis, multiclient testing, and real-world scenarios. These issues will make you or break you.