
Designing e-business solutions for Performance White Paper

Editors

Maggie Archibald - IBM Global e-business Integration Center, Global Industries - Dallas, Texas

Mike Schlosser - End to End Network Computing, NCSD - Austin, Texas

Contributors

Global e-business Integration Center - Dallas, Texas

Maggie Archibald
Simon Cheng
Alan Emery
Rod Fleming
Len Hand
Chris Li

Network Computer Division

Gary Huber, RTP
Lauren Kingman III, POK

Network Computing Software Division

Rajiv Arora, Austin
David Bachmann, Austin
Peter Bahrs, Austin
Robert F. Berry, Austin
Steve Burbeck, RTP
Nancy Burns, Austin
Jay Casler, RTP
Mike Collins, Austin
Mark E. Davis, Cupertino
Robert Dimpsey, Austin
Bryan Ellington, RTP
Beth Hutchison, Hursley
Michael Lentz, RTP
Jim Phelan, Austin
Dean Roddey, Cupertino
Norma Wolcott, RTP

Server Group

Judi Bank, POK
Eric Barsness, Rochester
Robert Dahle, POK

Software Solutions Division

Dov Bulka, RTP
Adrian Colyer, Hursley
Allan Dickson, RTP
Susan Hanis, RTP
Jim Knutson, Austin
Pat LiVecchi, RTP
Geoff Sharman, Hursley
Ruth Willenborg, RTP

Research

Peter Capek, Watson

Paul Dantzig, Hawthorne

Bilha Mendelson, Haifa

Frank Tip, Hawthorne

IBM Global Networks

David Bolthouse, Schaumburg, Il

IBM North America

David J. Johnson, Dallas, Texas

Version Note

The initial version of this paper was published as “Design for Performance White Paper” in June 1998 by the IBM Global e-business Integration (GEI) Center in Dallas, Texas. Version 2, published in March 1999, is a collaborative effort between the GEI Center and IBM Software Group. It builds on the foundation and methodology found in the first version and updates and significantly expands the guidelines provided.

Special Notice

The information contained in this document has not been submitted to any formal IBM test and is distributed AS IS. The use of this information or the implementation of any of these techniques is a customer responsibility and depends on the customer’s ability to evaluate and integrate them into the customer’s operational environment. While each item may have been reviewed by IBM for accuracy in a specific situation, there is no guarantee that the same or similar results will be obtained elsewhere. Customers attempting to adapt these techniques to their own environments do so at their own risk.

How You Can Help

The guidelines, hints and tips, and other information in this paper was provided by over 40 IBMers from many organizations and locations. The objective of this collaborative effort was to tap the vast reservoir of knowledge that exists within IBM and share it with others. You can help by documenting your experiences with designing e-business solutions for performance so we can include them in the next version of this paper, tentatively scheduled for 3Q99.

<p>Send your ideas, experiences, hints, tips, etc. for achieving good performance to: Maggie Archibald - marchiba@us.ibm.com Mike Schlosser - schloss@us.ibm.com</p>

Intended Audience	Page 6
Introduction	Page 7
End to End Performance	Page 9
Enterprise Solution Structure	Page 9
A Few General Guidelines	Page 10
The Budget	Page 12
The Client	Page 14
Introduction	Page 14
Performance Issues and Tradeoffs	Page 15
Recommendations	Page 19
The Network	Page 21
Introduction	Page 21
Performance Issues and Tradeoffs	Page 22
Recommendations	Page 27
Security	Page 29
Introduction	Page 29
Overview	Page 30
Performance Issues and Experiences	Page 32
Recommendations	Page 37
Web Servers	Page 38
Introduction	Page 38
Performance Issues and Experiences	Page 39
Recommendations	Page 43
Load Balancing	Page 45
Introduction	Page 45
Recommendations	Page 47
Integration Server - Object Request Brokers	Page 48
Introduction	Page 48

Performance Issues and Experiences	Page 49
Recommendations	Page 55
Integration Server - Message Servers	Page 56
Introduction	Page 56
Persistence	Page 56
Message Characteristics	Page 56
Recommendations	Page 59
Integration Server - Host Access	Page 60
Overview	Page 60
Host Publisher	Page 60
Recommendations	Page 61
Integration Server - CICS Transaction Gateway	Page 62
Recommendations	Page 64
Java	Page 65
Introduction	Page 65
Performance Issues and Experience	Page 66
Database	Page 88
Introduction	Page 88
Performance Issues and Tradeoffs	Page 89
Recommendations	Page 92
Appendix A - Dallas GEI	Page 93
Design with Performance Process	Page 93
Appendix B - Concept2 Client Framework	Page 97
Appendix C - Availability and scalability of web applications using session data and clustering	Page 102

Intended Audience

This paper should be read by software architects, designers, technical planners, etc. involved in the architecture, design, development and deployment of e-business solutions. While there is a focus on performance considerations of IBM hardware and software products and frameworks such as the IBM Application Framework for e-business, this paper also contains many application design topics that transcend specific products or technologies.

Introduction

This paper will discuss architecture, design, development and deployment issues that affect the performance of e-business solutions. Designing solutions with performance in mind is an old concept that can be applied to the new e-business environment. It's an old concept because IBM has been building solutions; and the hardware, software, and networks needed to run them, with performance in mind for decades. It has a new dimension because the computing model defined by the IBM Application Framework for e-business is based on several new and exciting technologies such as the Enterprise Java Bean (EJB) distributed object architecture. And, while value add networks and the web have been around for decades, and years, respectively; the explosion of e-business and e-commerce is fairly recent. New metrics, new bottlenecks, and new demands for performance abound.

It's time to apply designing for performance methodology and guidelines to this new technology and ensure we include performance considerations at the architecture and design phase for all e-business development projects.

Why is this so important? To be frank, we have encountered a few e-business solutions which were architected without a well understood and communicated requirement for performance. Without clear performance objectives during initial architecture definition and high level design, performance becomes an afterthought; something to be dealt with *after* hardware and software products have been selected and application code developed. We have enough data and experience now to know that these projects often fail when they are deployed - not because they cannot deliver the required business function, but because they cannot service the demand generated by their customer community in a satisfactory way.

We know that performance isn't the only thing to worry about and that there are other things to consider when designing e-business solutions. And we also know that the computing model and the infrastructure have evolved rapidly. The technology and tools may be too new for reliable performance and scaling estimates and projections to be made. How do you predict performance of Web based applications? How do you understand the potential performance and scalability issues of building your applications in an environment which has an effective life-span of a "web year"? And finally, how do you measure end-to-end performance anyway?

This paper will address these questions by sharing our real life experiences with projects and products across IBM; from experiments conducted by IBM Research to the stress testing of real Web based applications such as the Nagano Olympics Info'98 (intranet) application.

It is worth noting that this paper is not the only information you will need pertaining to performance in the e-business environment. This paper is complementary to other performance information you will need such as:

1. absolute performance and throughput of a specific product (e.g. web server X can serve up 3000 pages a second)
2. specific tuning information of a specific product (e.g. the XYZ parameter must be set to ON when using Java)
3. the relative performance among technologies (e.g. the same program written in Java will run 87.34% as fast as the same program written in ADA)
4. competitive performance (an entity bean will initialize 13% faster on an IBM processor compared to a SUN Microsystems Model QQQ).

In other words, even if the required products and technologies have world class performance, scaling, and other related capabilities you still must design your application with performance in mind.

Finally, the information in this paper may be all you need to design and develop a world class e-business solution with excellent response time, performance and scalability. Or, you may determine you need additional help with design considerations, with establishing the appropriate performance criteria or assistance with testing your e-business application. [Appendix A](#) describes the organization that can provide this service: the Dallas Global e-business Integration Center (GEI).

End to End Performance

Enterprise Solution Structure (ESS)

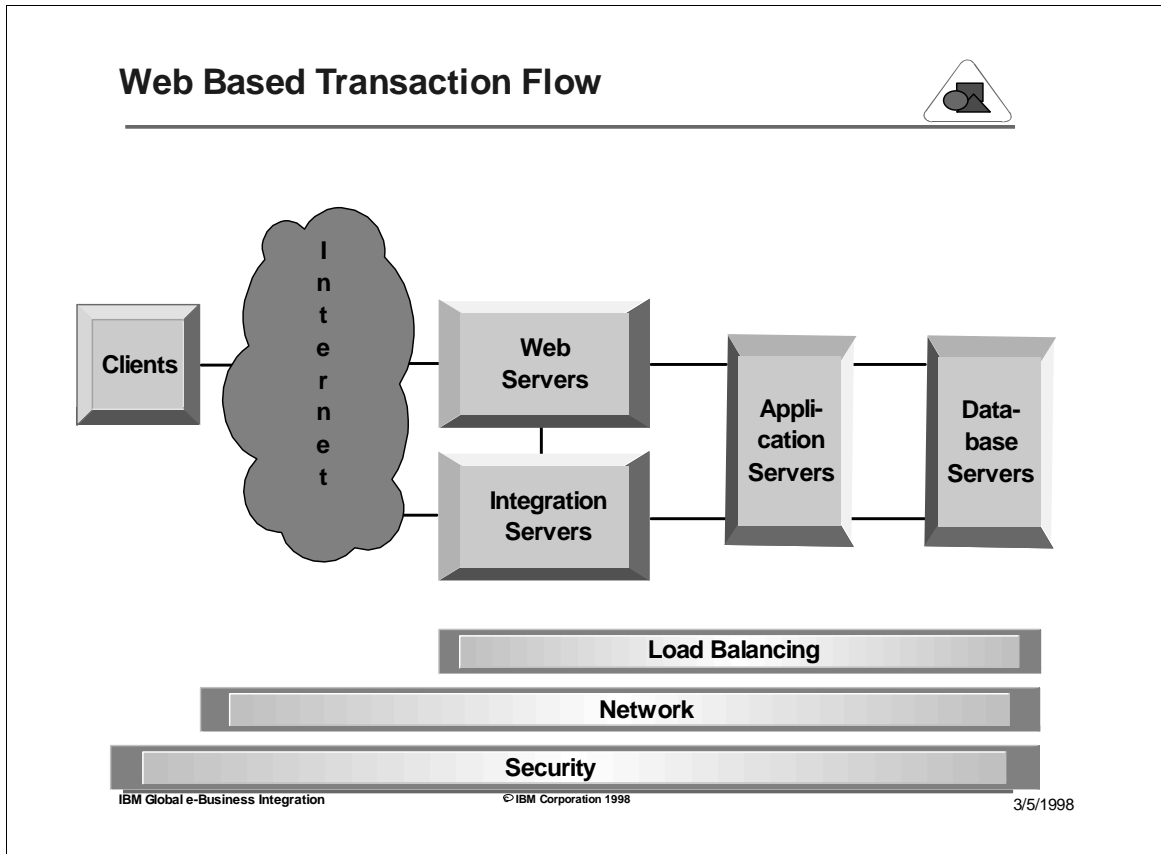
IBM has developed a comprehensive process for using well-understood design patterns to match design alternatives against requirements in order to design the application, select the appropriate products, etc. One key concept of ESS is using “nodes” or building blocks to define / delineate the solution. In the diagram below the highest level building blocks include:

- Client
- Internet network
- Web Servers
- Integration Servers
- Application Servers
- Database Servers

The “performance” metric of an e-business solution is typically defined as the response time for the end user. So, the “performance problem” is often defined as “slow” response time when the user takes an action or series of actions when using the application. This request may be for information or to have the application store or analyze information on their behalf. The user, quite rightly, has no concept of how much work and how many systems are actually involved.

Unfortunately, many application developers and systems support teams also lack a concept of what work and systems are involved in these transactions. A “simple” transaction may turn out to be a very complex series of data flows involving multiple systems in multiple locations. It may cause events to cascade through multiple systems before the summarized result is presented to the user.

The ESS building block approach allows us to move from the “response time is slow” description of performance to decomposing the solution into meaningful nodes and then analyzing each node in turn. This is the approach we will use in this paper. In addition, there are system-wide considerations such as security and load balancing that span nodes. These topics will also be discussed.



A Few General Guidelines

There are many specific performance guidelines, hints, tips, and other helpful information in each chapter. There are also a few macro-level guidelines for designing for performance that apply to all components of an end-to-end e-business solution. You should keep these in mind as you evaluate the architecture, design, and deployment.

1. **An e-business solution will have good performance if, and only if, attention is paid to each node; the client, the network, the servers, etc.**
2. **You must understand the detailed data flows and systems implications up front** in your initial design. This analysis, when combined with refinements in understanding during the applications development cycle will allow you to set expectations correctly and structure your applications and systems for success.
3. **Plan for growth in your initial design.** The Application Framework for e-business is based on technology that is open, industry standard, and available on many platforms including those available from IBM and other vendors. One of the benefits of Java is WORA (write once, run anywhere). A solution that is designed to be 100% platform agnostic and modular can, if required due to growth, be distributed

among additional network nodes and / or moved to a larger system. On the other hand, a solution that uses proprietary interfaces, system utilities, tools, extensions, etc. may end up “stuck” on a platform that doesn’t scale. Since developers often choose proprietary implementations in the belief that closed systems perform better than open systems, a plan for future growth requires a good understanding of the longer term performance and scalability needs vs the short term performance “optimizations” derived from proprietary technology.

4. **Use the latest release of infrastructure and system software.** Each release of e-business software and related infrastructure and system software provides faster performance than the prior one. In some cases, such as the IBM JDK 1.1.7 for Windows and for OS/2¹, dramatically better. Using the latest release of software, whether it is the Java JVM, web server, TCP/IP network stack, etc. is a key consideration for achieving good performance. You should pay particular attention to software prereqs and coreqs, especially if your solution includes existing systems in the enterprise.
5. **Cache.** Cache information anywhere and everywhere possible. We will discuss how caching is done in many of the specific sections below, but the macro-level guideline is cache, cache, cache!

Now that we’ve documented this short list of general guidelines, we’ll define a general methodology for assigning a performance “budget” to each logical node or segment of an e-business solution. After that we will consider each part of the schematic in turn and provide more specific hints, tips, guidelines, tools, and a discussion of some of the trade-offs that may be applicable.

¹ A report on the latest VolanoMark benchmarks and IBM’s leadership is at <http://www.javaworld.com/javaworld/jw-03-1999/jw-03-volanomark.html>

The Budget

How can you define the expected performance of an Internet application? The Internet is a vast interconnected network with over 5 million domain names!² Well, you can't define the performance of the Internet, but you can define the performance objectives for your solution. A common performance metric is interactive response time as perceived by the end user. You can divide your solution up into three buckets and assign a percentage to each bucket:

- time spent in the client - some %
- time spent in the network - some %
- time spent in servers - some %

For example, if your goal is 2 second interactive response time, you might assign the percentages as:

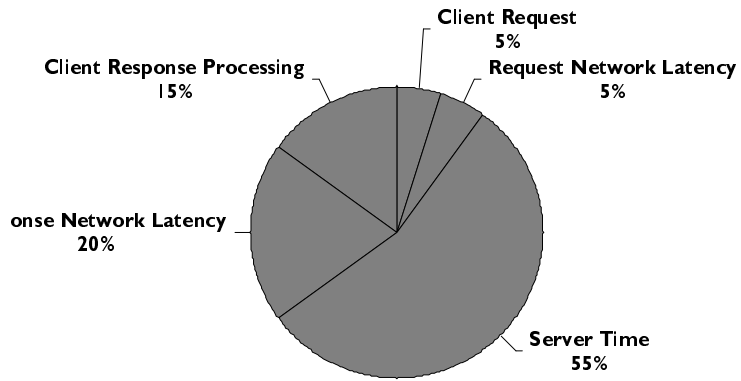
- client - 5% (100 msec)
- network - 30% (600 msec)
- servers - 65% (1300 msec)

You can then apply the target number (e.g. 1300 msec total for the servers) to the detailed data flows you defined during your initial design using the building blocks. Based on the total work required by each server, the inherent network latency of the enterprise network and the total number of trips between server systems, you can evaluate how close to your target you are. The response time goal, and the ratio of that goal assigned to each bucket can be used during initial design and data flow definition phase. The allocation of response time should continue through each step in the development process up to and including production deployment. The assignment of ratios among client, network, and server may require an iteration or two as well. The GEI performance team found that development teams initially assigned a very low percentage (about 5%) to the client, only to find this time to be in the 20-40% range of total time. This has usually been a surprise to the enterprise for which the investigation is being conducted.

The following chart provides an example budget for e-business solutions. The budget you develop for your application may be quite different based on user requirements, complexity of the application, the ability to define the network infrastructure (i.e. for an Intranet solution), etc.

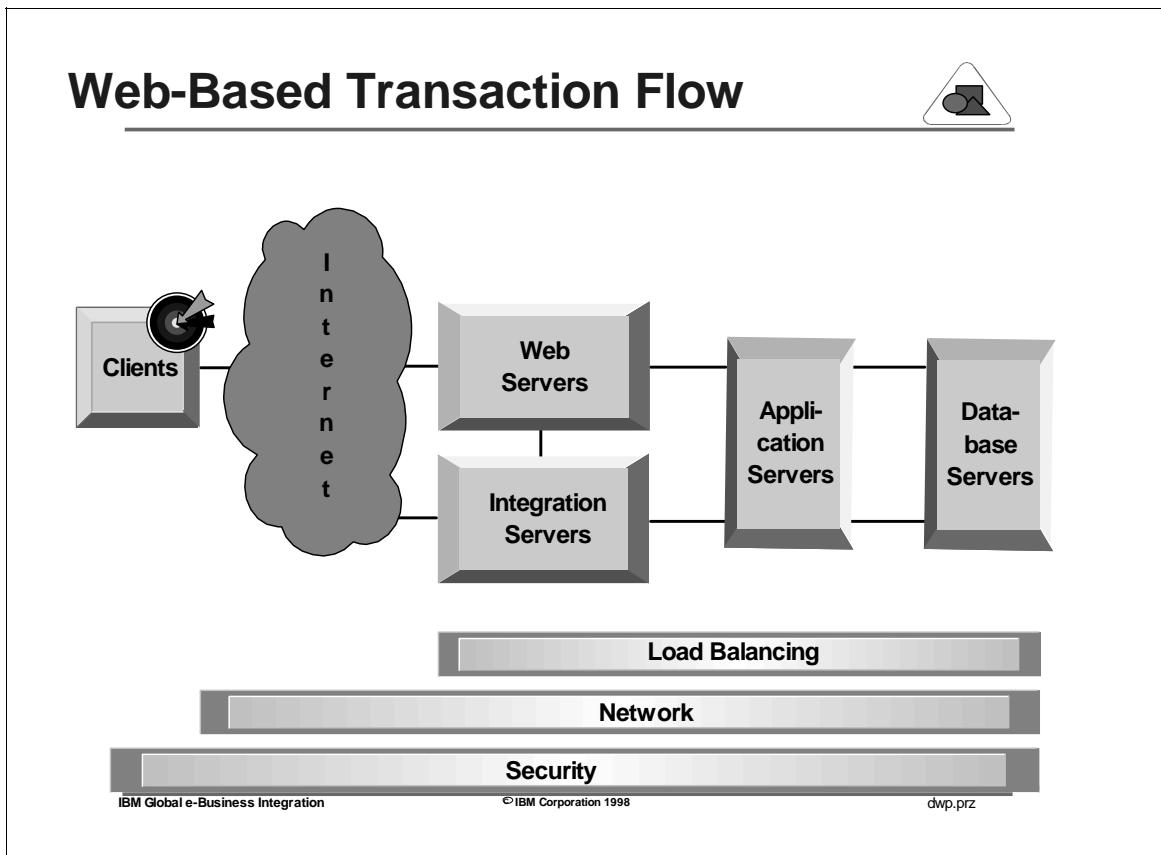
² actually 5,012,287 as of 02/01/99 as recorded at <http://www.domainstats.com/>

Success Tip - Response Time Budget



The Client

Introduction



The IBM Application Framework for e-business defines the programming model common to most e-business solutions.³ “The Application Framework for e-business clients span a diverse range of products from information appliances, such as digital telephones and PDAs, to network computers and PCs. While the individual capabilities of each of these clients vary significantly, they are unified to the Web application server by their reliance upon a set of Web-based technologies and protocols. This set includes Java, TCP/IP, HTTP, SSL, HTML, DHTML, XML, MIME, SMTP, IIOP, and X.509, among others.”⁴ The architecture and design of your e-business solution will depend on the technology you choose on both the client and server side. A useful guideline is “The less control the developer has in specifying the level or type of

³ “IBM Application Framework for e-business Web Application Client Programming Model” <http://www.software.ibm.com/ebusiness/clientwpab.html>

⁴ *ibid.*

browser required, the more basic the client code must become. If a Web application is to be accessed by users with varying browser capabilities, then to be conservative the client is often written in HTML.”⁵

In addition to the choice of base technology, (e.g. Java or no Java on the client, etc.), there are also development considerations that are directly related to performance. The development practices used to create client side code on small or constrained client systems are, more often than not, good practices for any client. Perhaps you have the requirement (and time) to develop multiple clients for your solution; one for robust, full function Intranet end user systems and another for Internet end users. If not, sticking to the conservative design point both for choosing the technology base and for developing the application code may be more appropriate. The trade-offs in architecture and design, as well as programming practices are discussed in the next section with additional information included in the sections on Object Request Broker and Java.

Performance Issues and Tradeoffs

The client / presentation side of e-business solutions can include a mix of pages, graphics, Java UI, etc. Some of the information presented is static, some is built dynamically, and will include / require end user interaction for selecting options, filling in forms, etc. It is beyond the scope of this paper to discuss general web page design or web site design other than to say that a well designed web site provides clear transversal paths through the pages and minimizes the presentation of superfluous information. A web site based on these and other usability guidelines will, as a secondary benefit, also exhibit better performance by reducing the number of pages that are displayed and/or generated by a confused end user.

One other “mind set” topic relating to web page design and performance is the schism that too often occurs between the artists that design the page graphics and other multimedia content and the programmers that develop the business logic, connectors, etc. Graphics people may not know that their beautiful images can cause poor performance, **and** the architects and programmers may not know how to request changes to improve performance. But we do know that 100 Kbytes of one image will download faster than 100 Kbytes contained in 10 images. And 30 Kbytes is faster than 100 Kbytes. Image compression and combining images are crucial to client side performance.

There are a couple of “tricks of the trade” that IBM used for the Olympics site (intranet, not the Internet site) to attain good performance. The home page JPEG images and all the images for the navigation frame were stored on each client PC. The images were preloaded on each PC, and if a change was required, we'd simply change the HTML from a `file:// URL` to an `http:// URL`. As it turned out, we didn't need to change

⁵ *ibid.*

anything. This saved a significant amount of network bandwidth and was one of many tuning changes which helped us to achieve our 90% of transactions in 2 seconds performance requirement. That technique might be handy for a PC-based kiosk where timeouts frequently cause the main screen to be displayed.

Another good idea uses frames. If you are able to require a frames-based browser and if you have pages that you know will take a long time to load, you might want to have a 'status frame'. This could be updated as events on the Web Server occur (e.g., getting account data, please stand by). That way the user doesn't have to sit looking a blank screen for minutes while their data is brought down. A status frame might be useful if you're a financial organization and the user asks you to pull down 90 days of account activity or something similarly long-running.

A technique similar to a status frame, but implemented using Java applets and inter-applet event flows, was developed as part of the Concept2 Framework used to provide a consistent user interface for administrators using IBM middleware products. A description of the Concept2 design and how to work with the anomalies of web browsers is in [Appendix B](#).

Transactions

The remainder of this section will focus on the client interactions that comprise "transactions" as defined by the ESS NC transaction pattern. These interactions include three types of tasks:

- formatting the request on the behalf of the user (or system)
- submitting the request
- formatting the responses for presentation to the user (or system)

While there may be some overlap among the steps in their actual execution, the time required (elapsed time) should be assigned to each category and used to build a generic profile for the application. The end user's expectation for a response begins when they believe they have asked the application to do something on their behalf. There can significant delay before any of the required data sources/sinks are even asked to begin their processes. These delays on the client (front) side of the transaction can be as high as 40% of the user's perceived response time.

We will refer to these starting delays as client startup delays. This portion of time is typically consumed by taking the user's request and reformatting it for compatibility with the target data sources/sinks. Client side scripts and applets are often used to validate the user's input prior to sending any requests to a web application server. Depending on the design and the need for additional data or components to be loaded, this validation process may be lengthy.

Formatting the request

One common source for the delay between the end user initiation of a request and its execution is the time spent reformatting a request. There may be one or multiple transformations. Another significant issue is the loading and initial startup of code in/on the client system. This is a particular concern for applet downloading from the network. Memory management on the clients can also contribute to significant delays at the client side. We have found that these client-side performance issues are often ignored in the design of the solution. When they are ignored, techniques to minimize their impact are not considered.

Encryption is one type of request/response conversion that impacts performance and response time. The use of SSL connections between browsers and servers and the growth of Virtual Private Networks is increasing the amount of encryption being used by applications. The use of encryption usually requires additional cycles on the client to execute a software algorithm for encrypting the data to be sent as well as overhead for session key management. This overhead adds time to the client side processing of the information. Through careful design you may be able to reduce the impact of encryption by making selective use of it. For example, you may not wish to encrypt selected graphics or images.

Another consideration of “poor performance” is when the user is ready to enter data but the system is not ready for them. This is usually caused by a request to download a script, applet or component as part of establishing the user’s input environment. Java applets downloaded must go through initialization. The same classes loaded from different servers are treated independently. These issues and others may mean that the application is not structured to quickly enable user input.

Submitting the request

One of the least understood characteristics of an application during the development cycle is how the client side of the application actually communicates with the other nodes during a transaction. Part of the reason for this confusion is that the high level interfaces and tools used to implement applications can mask the input/output patterns that will be generated by the application on the client and the other nodes.

We have seen several behaviors related to how the application requests the data. One of the most common behaviors contributing to slow response is the high number of requests the application makes to perform the required work. There are applications that move hundreds or thousands of records across the network to the client. Often these requests are done in a looping manner with a specific request for each data record that serializes these data requests. In one, worst case example, the application was reading the same data across the network multiple times only to use a different part of the same record. Data aggregation or coalescing work should not be done on the client, but should instead be performed by the web application server.

On the client-side, the use of connections by the Web Browser can impact performance. Web Browsers typically open a unique TCP/IP connection for the base page and each additional object retrieved on a page. The minimum time required to receive the first block of data is the two round trip times across the network per object. Many browsers will allow the configuration of multiple connections that can help mask the time required to retrieve objects by requesting them in parallel. But once the number of objects exceeds the number of connections, waiting for a new connection will still cause delay due to serialization on parallel flows. Depending on the browser and TCP/IP stack used, delays up to 250 milliseconds can occur between the closure of one connection and the retrieval of the next object.

One approach to browser connection delays is the ability to use persistent or Keep Alive connections. These persistent connections reduce the overhead of creating a connection for each object, class or picture on the page. It also allows a connection to better use TCP/IP's windowing mechanisms by reducing the number of new connections to which TCP Slow Start must be applied. Both the server and the browser must support the persistent capability and have it enabled before it will be used. Note that not all objects on a page are capable of using the persistent connection capability even when it is enabled on both ends. In addition, the object requested must come from a host with an existing connection from the browser. Finally, there is a tradeoff between performance for one user and access by other users. In some cases, the persistent connection capability is disabled on the server-side to avoid blocked connections for new user requests.

Java applets and client-side Java applications usually need some kind of network connection as well. Each of these connections will have overhead associated with them. Some of the server-side environments provide a pool of pre-established connections. This can reduce client startup time. When testing the scalability of an application attention should be paid to the pool and how it allocates and manages the use of existing connections among clients.

Formatting the Response

Once the response has arrived back at the end user's client device, it must be formatted and displayed. Many of the same considerations that apply to formatting the request apply here as well. Packets that arrive encrypted must be decrypted. A client side application may need to take actions on the data to derive a final result before being presented to the user. Responses to requests that were reformatted by middle ware may undergo another transformation back to the original format before presentation to the client side application.

Formatting the response, particularly in Web Browser based applications, can take significant amounts of time. Color management and remapping may also take place prior to the images being displayed.

Large table construction and formatting is very expensive. The use of tables containing images may also lead to perceptible delay during the construction process of a page for

a user. For example, with some browsers, if the dimensions of the images used in a table are not specified, the table will not be presented to the user until the image arrives and are decompressed. In other browsers, the table construction is visible during construction but is rebuilt as the images arrive. This technique has a negative impact on both performance and usability.

Another area where trade-offs must be understood is browser-side caching. Both pages and images can be cached locally at the user's client system. The user usually has the option of how much memory and disk space to allocate for a browser side caching. They can also control how often an object needs to be revalidated before being used from cache. The application can also define which objects are eligible for caching. In an Internet or Wide Area Network Intranet e-business solution retrieving objects from local user cache is usually significantly faster than their retrieval across the network.

There are usually three caches associated with the browser: disk, memory and image. Each of these caches has different performance characteristics. For example, images in the image cache are decompressed but images in the memory and disk cache are typically compressed. We have some experience that shows that the overhead of managing large caches can itself degrade performance. Also keep in mind that the same object retrieved from two different sources is treated as multiple unique objects by browser caches. The problem you face as a designer, particularly for e-business solutions, is that you have little control over when objects are cached or removed from the end user's cache.

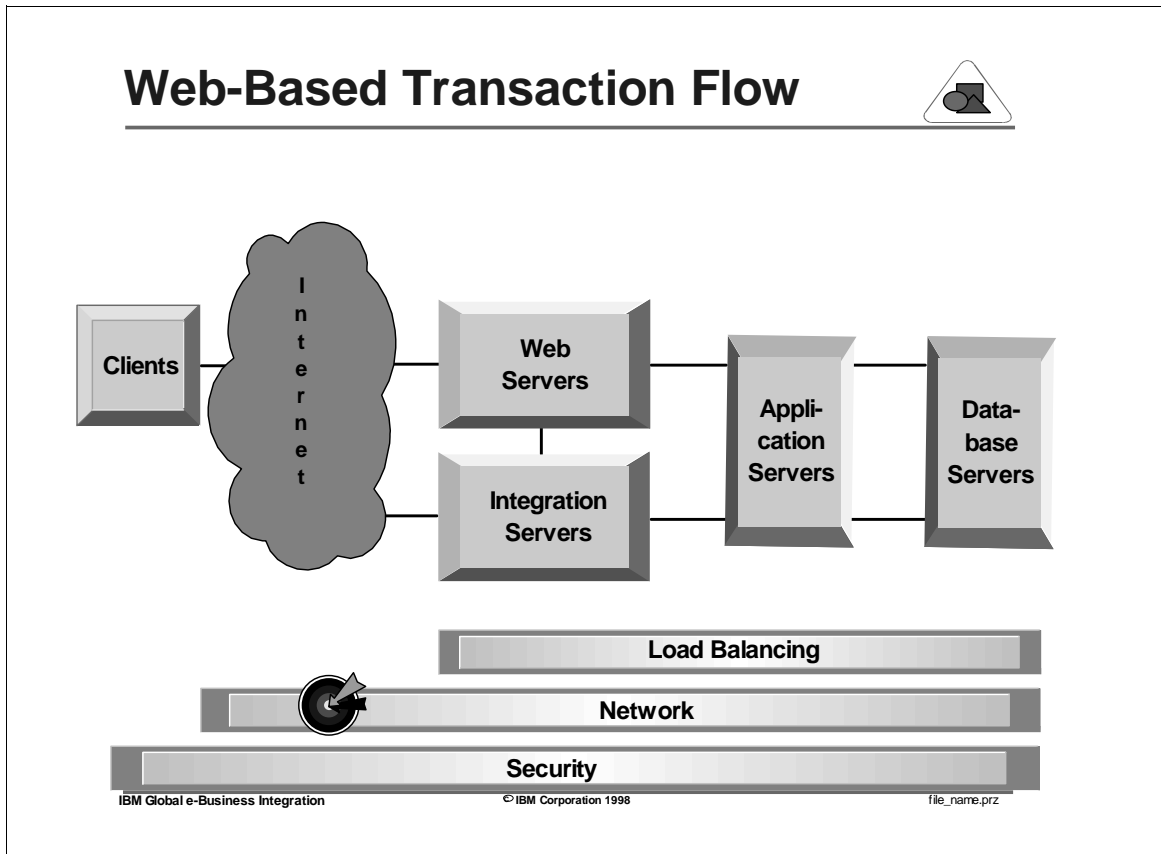
Recommendations

1. **Develop response time budgets for your design.** How much time should be allocated for time spent formatting the request and the response in the client?
2. **Define the client system configuration.** Many of the topics in this section indicate that the performance of the application is impacted by inefficiencies on the client side. In our studies, the speed of the client machine (processor, disk subsystem, video subsystem) and configuration (e.g. the amount of memory) made a perceptible difference (up to 35%) in user response times. Where possible (i.e. Intranet solutions) establish a "minimum" client configuration needed to run the application. In any case, avoid a mismatch, where the client-side of your e-business solution performs well on a 500 MHz Pentium II PC but must be deployed into a network where most end users have 486 PCs with no plan to upgrade. There is a tradeoff between advanced client-side function and achieving good performance, and it must be understood during design.
3. **Test the application on the full range of systems** it is to be deployed on, not just fast development machines. Conduct testing, during development, on similar systems and over a representative network configuration and load that matches the

deployment environment. This will help you understand the performance of the solution from the end user's perspective.

The Network

Introduction



This section focuses on the network and how it will affect performance of the solution. Rather than focus on network technology choices (an important subject but beyond the scope of the paper), we will focus on how network characteristics impact the application design.

Three network factors tend to influence application performance

1. the latency between systems
2. the packet loss rates
3. the budget

With a big enough budget, latency can be brought near to the current boundary at the speed of light. Once everything is in close proximity and moving at the speed of light without queuing and packet loss, your network is doing the best that it can for your applications. Unfortunately the budget will usually give out before this point is reached.

Performance Issues and Tradeoffs

In the “good ol’ days” application performance was much easier to predict. Applications, at least those with a user interface, provided the user with a limited amount of user input followed with a screen’s worth of textual output. This model worked well with 3270 terminal applications. The performance predictors focused on “host” response time combined with concurrent load at common points and latency in the network. Fixes to response issues time often focused on faster hosts and faster communication circuits.

The Application Framework for e-business incorporates some client side intelligence, middle tier servers and distributed data sources. This makes the modeling and estimation of the transaction more complex. There are some basic considerations about these new applications and their network input/output patterns when combined with network characteristics that can be helpful when designing with performance in mind. Fixing or avoiding performance problems requires looking at multiple systems, communication circuits and the applications interaction with each of these components.

Latency

Let’s examine each of the three network factors affecting application performance , beginning with latency. For purposes of this paper and design discussion, latency will be defined as the round trip time between two systems due to the sum of the delays in the network components traversed. Latency has two important measurements, the average time for these round trips and the variance observed across multiple round trips. Latency is bounded on the low end by the minimum times to transit the electronic components in the path plus the time required to serialize the data onto media in the path plus the propagation time required to cross the media.

As intelligence, processing and buffering are added to components in the network path, you begin to have sources of variance added to the latency. As the utilization of the media increases, queuing for access to the media grows, generating additional variance in latency as well. Variances in latency is measured by changes in response time for end users. Randomness in response time can be the source of user dissatisfaction just as much as consistently long response times.

The absolute value for latency is not as important to understand as it is to know how the application flows interact with latency. Application traffic can be simplified for study into

two categories; serialized flows and parallel flows. Serialized flows will fully expose the network's latency for each round trip required. Parallel flows, through overlapping requests, may mask or hide some of the network's latency for each request concurrently active beyond the first one.

Starting with the initial client request, most application flows are serialized; they make a request and get a response. A user enters or selects a Uniform Resource Locator (URL) and waits for the resulting page to appear. More advanced pages will solicit some form of user input and then return a page of information or results, but this model still resembles a request/response serialized flow.

Behind this serialized flow may be other serialized flows and parallel flows. In the simple example of a page returned with text and some limited number of graphics, there is a serialized flow to get and return the base HyperText Markup Language (HTML) page. As the page is returned and it is parsed by the user's browser, parallel flows may be used to request the additional objects and classes needed by the page for presentation to the user. For requesting the additional objects specified in the base HTML for the page, parallel sessions may be used by the browser. A common default is for a web browser to support four parallel sessions for retrieving objects. As more browsers and servers enable support for HTTP 1.1's Persistent Connection capability, the number of parallel connections seen is often two. Two is the number recommended in section 8.1.4 or RFC2068, Hypertext Transfer Protocol -- HTTP 1.1.

As the number of objects on a page increases, queuing for the available parallel connections can actually lead to a serialization of the flows again. If you have a base page and more than three additional objects on the page, you will serialize or block on the availability of the next connection. The overhead required for terminating the previous TCP/IP connection and setting up the new connection plus the actual transfer time for the previous object affects the response time of the user. Latency is magnified by the serialized points in the parallel data flows as well as in the elapsed time to move the object.

The minimum time required to open a new TCP/IP connection and obtain data from a source is two round trips across the network. Therefore the best possible response to the user is two times the network's round trip latency. In reality, objects are often large enough that they require additional round trips across the network, increasing the minimum time to present the object to the user. These additional round trips to bring the object back can be affected by latency when the TCP/IP window has not opened large enough for the a TCP Acknowledgment (ACK) to return before the current window of bytes for transmission is exhausted.

The number of objects on a page therefore can greatly impact the user's perception of response time because of the way they magnify the network latency during the retrieval of those objects. Improvements in web application performance have been achieved by combining adjacent visual objects on the page into one object. This reduced the

number of connections required for a page and in most cases reduced the actual number of bytes required to transfer to the page to the browser.

In the limited number of applications studied to date, the use of persistent connection capability has had only a limited impact on the user response time. Requests for objects still queue for the persistent connections. The main benefits were achieved because the TCP/IP transmit window was open for the connection. For the Intranet LAN based environment where these applications were studied, this led to a minor percentage improvement in response time. We expect that applications deployed in an Internet client dial-up connection based network will exhibit a greater improvement in response time because the higher latency requires a larger window to avoid waiting on ACKs.

The HTTP 1.1 standard allows for the "pipelining" of requests on a persistent connection. Pipelining allows for multiple requests to be outstanding on one connection. This can reduce the wait time caused by the delay between non-persistent connections by avoiding the startup time that each connection requires to open the transmit window to the optimal size. But queuing can still take place because a server must send responses back to the client in the order the requests were received. Small objects can still be queued behind larger objects for transmission, but the queuing is now relegated to the server end of the connection rather than the browser. The HTTP 1.1 standard also suggests that clients using persistent connections should have a maximum of two connections to each server. This cuts the amount of traffic that can flow in parallel in half compared to the common default of four connections per browser using non-persistent connections.

One particularly poor behavior, seen initially in client/server applications and more recently in e-business applications, is retrieving multiple records across relatively high latency network components in order to develop summary information. This poor behavior is usually not exposed in a development test bed and only appears during pilot or initial rollout into a production distributed network.

The multiple record retrieval and its resultant network latency symptom should not occur between the browser and the web server. It should be done on the web application server that is building the page to be presented to the user. The servlet or EJB program that is invoked at the web server acts as the collection point for the data to be returned to the user. We have seen multiple examples where the code written on the server has looped or iterated through the records required to build the page. In this case the number of trips multiplied by the round trip network latency between the web server and the data sources becomes important.

In summary, the technique used to access data and the amount of data requested are key design decisions due to network latency. Applications that read every record in a file to satisfy a search or develop a summary are particularly susceptible to the impact of network latency.

We can also apply what we have learned from other computing models such as client/server to e-business solution design. For example, we know that that the latency through a bridge between two Local Area Network segments can have a significant impact on end user response time. Bridge latency varies, but let's use the example of two milliseconds. One application studied read thousands of records, one by one, across the network to search for data. Ten thousand records were read to satisfy one request. In this case, network latency contributed a minimum of 20 seconds to the application response. And we know, from recent analysis of e-business applications, that the action required to construct a web page can cause a hundred or more serialized trips across the network due to loop constructs within the programs. With only 20 milliseconds of latency, this meant more than two seconds was added to the user response time for this portion of application data retrieval.

The volume of data returned is also impacted by latency. We have seen application that accessed large volumes of data over relatively slow links. In some cases the data was never used. We have also observed the padding of returned data streams due to mismatched buffer sizes. In a relatively simple LAN the padding added 30 milliseconds to each retrieval.

Packet size is another consideration. Smaller packets usually mean more protocol overhead to move the same volume of data. There may also be more ACKs required for transmitting the packets. Packets that are too large will either be discarded or broken up into smaller packets enroute. If TCP/IP is configured to send large packets, but the application is sending small pieces of information, you may encounter delays while TCP/IP waits trying to optimize the amount of data in a packet. Mismatches between large packets and small receive windows may slow down the rate at which data can be transmitted through the network as well. These issues are hard to estimate and require testing in and then tuning applications in an environment similar to that in which it will be deployed.

Often IP based applications require a special step prior to using a target system. They must resolve a host's name to a specific IP address. This usually involves using the Domain Name System (DNS) to resolve the address. Depending on the application's design, the IP setup, and the response returned by the DNS, this overhead may be incurred once or for every request. There are alternatives to this worse case scenario. For example, the Language Environment (LE) component of S/390 has a cache for DNS entries. This cache is searched first before sending a request to a DNS. The time to resolve the IP host name should be considered and added to your budget if it will occur frequently in your environment.

Similar to the name resolution process for IP, other protocols will have flows required to identify resources needed in the network. These flows will need to be identified as to their frequency and the time required to complete for input into your design process.

Packet Loss

Packet loss across a network is usually due to one of two causes: overloaded components in the path or packet corruption in transit. For TCP/IP networks, TCP will detect both packet loss and packet corruption. The most common problem is packet loss due to overloaded components. This loss may be real or perceived. It is real if a component could not buffer the packet. It is perceived if the network buffers it but the delay associated with the packet or its acknowledgment becomes so long the packet is believed to be lost by the endpoints.

A TCP packet received and determined to be invalid because the TCP checksum does not compute correctly at the destination will simply be discarded by TCP and will not be presented to the application. The originating host will time out waiting for the corrupted packet's acknowledgment and will generate a new packet. Recovery for lost packets and packets corrupted in transit is handled the same way, time out and retransmission by the originating host.

In order to understand the impact of packet loss on an e-business application, it is important to understand the recovery levels affecting the application's data flow. Detection and recovery of packets can be handled at multiple levels depending on application design and the protocols being used. Most web applications are built upon IP's reliable connection oriented TCP capabilities.

In simple terms, TCP detects the loss of a packet through the use of sequence numbers and acknowledgment timers. It is the acknowledgment timers that interact the most with network flows. The values used for TCP acknowledgment timers are dynamically calculated based on the prior round trip times of previous packets and their acknowledgments. Therefore the more network latency that has been observed, the longer the time before recovery begins. Once error recovery begins, it is not guaranteed the condition will have cleared by the time a retransmission starts. The general rule is that retransmissions for unacknowledged data will occur with the delta time between packets doubling until a maximum time is reached or the maximum number of retries has been attempted. These dynamic retry values and doubling of retries leads to potential variance in response time for the user. The larger the normal network latency value, the more variance that the user will be subject to when recovery is successful.

For web applications built on top of connection less protocols like IP's UDP, the application must provide packet loss detection and recovery. Performance studies conducted on client/server applications using UDP and similar protocols have shown that errors in network characteristics assumptions can dramatically affect performance. One problem involved a windowing scheme that falsely allowed the server to open its send window larger than the client's receive capability. When the server sent packets that were larger than the client's receive buffers, the application would temporarily pause while the server timed out waiting an application level acknowledgment. After the time out, measured in seconds, the server would retransmit the data with a smaller window and the client's application could handle the data and update the user's screen.

The problem only manifested itself when the data fields returned in the middle of an application session exceed the client's receive buffer.

In a second example, the application implemented its own packet sequence numbering for packet loss detection and recovery. If the packets lost were the last packets in a sequence of output data to the client, the client's application would hang. The feedback mechanism to acknowledge packets received only worked when the client sent new data to the server. The user's thought their systems were hung so they were not generating any actions to trigger the inbound traffic required for the acknowledgments to be carried back to the servers.

Different layers in the application may also have their own recovery timers and characteristics. It is important to understand possible interactions with network latency for these timers. In a high latency network, a timer may expire before an ACK has had a chance to return across the network, causing retransmissions which then add to the network load.

Budget

In a perfect world, all systems would be in close proximity to the users and infinite capacity would avoid queuing and packet loss problems. Liberally applied funds can be used to reduce the network latency and packet loss rates in the path between components. As this situation almost never occurs, entering the design process with reasonable expectations for the network latency that will be encountered and quantifying the number of turns across the network that are required for the user's interaction with the application is extremely important.

A response time budget can help set expectations for the application's performance. Start with the desired response time for each step in the application. Look at the portion of that response time you are willing to allocate to the client, network transit and the server side. If your goal is two seconds, you may wish to allocate .5 seconds to the client portion and .5 seconds to the server side. This leaves one second for network transit time between the client and the server and the server and additional tiers. If the expected round trip latency in the network is 20 milliseconds, you know your design requires less than 50 serialized round trips. As the application is designed and profiled, you can decide whether the allocations of time are reasonable for the technology chosen, whether the technology needs to be used differently, or if the response time budgets must be reallocated.

Recommendations

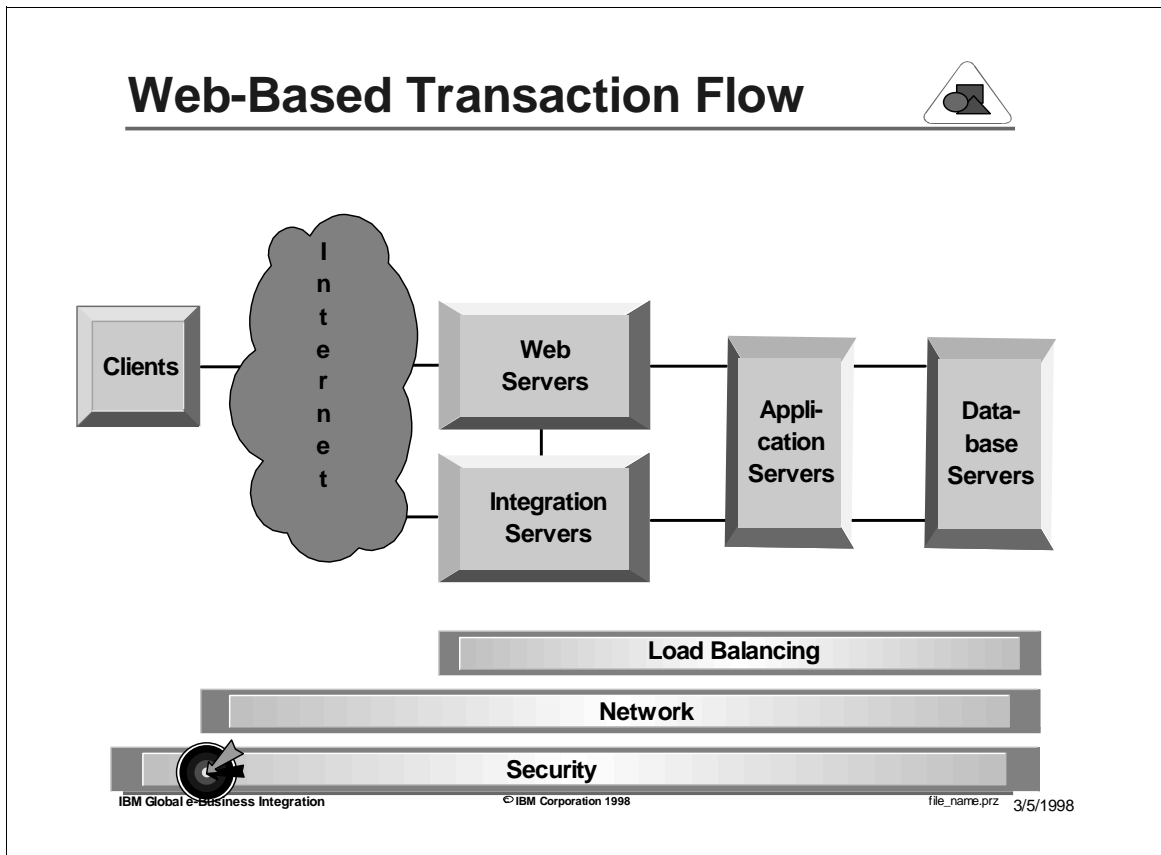
- 1. Use application profiling and technology testing during design and development to understand how your application uses the network.** Profiles

will help you quantify the number of trips required across different network components. They will also allow you to identify serialized input/output patterns and when large numbers of parallel flows are required. The profile will also help identify the traffic volumes and potential for mismatched packet flows and network packet sizes.

2. **Understand the deployment network** and the typical and peak round trip latency for the paths your application will be using. This information will allow you to manage the amount of the desired response time you must allocate for crossing the network.
3. **Gather data about application performance** by adding some limited logging to the application. Keep track of response times from the user application's viewpoint and report those back to a central repository. If a trend analysis shows response time growing or a variance from what is expected you can conduct performance analysis work to determine what is varying from your expected results and the criteria they are based on.

Security

Introduction



This section deals with the performance ramifications of various “security” technologies and implementations. Security is only one domain of a design, but can have a major impact on the overall end to end performance budget. Trade-offs must be understood between a very secure but poor performing solution and an application that exhibits excellent performance and response time but is not secure.

So what do we do? The first step in understanding the performance ramifications of a security architecture is to understand the technologies. It is beyond the scope of this paper to define all aspects of security. The discussion that follows will help to understand some of the important ramifications of security on an end-to-end budget.

This section will focus on :

- SSL (Secure Socket Layer)
- ACL (Access Control List)
- Directory Server
- Firewall

Overview

To make sure we all agree on the terminology, here are a few definitions:

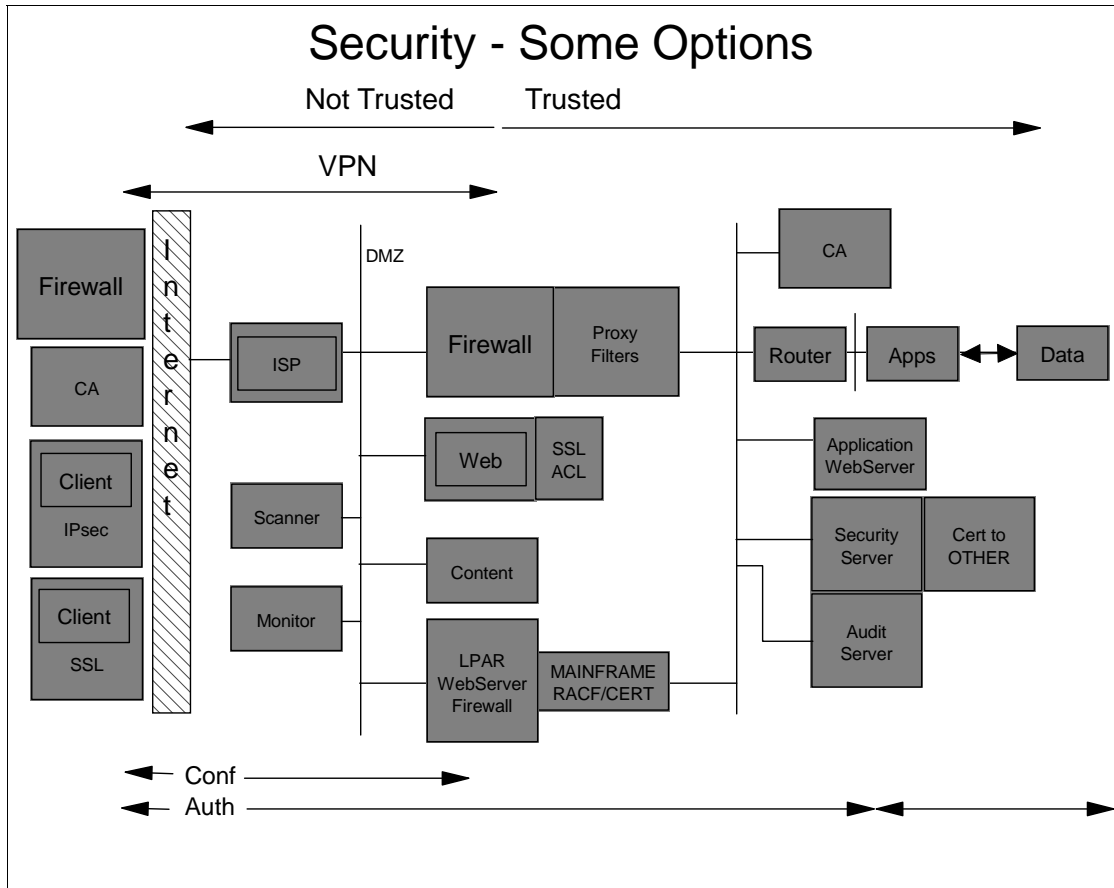
- **Cryptography** -- the "art" of hiding and securing information for storage and transmission.
- **Encryption** -- the process of converting readable information called plain text into unreadable information called cipher text
- **Decryption** -- the process of reverting encrypted information (cipher text) back to plain text
- **Algorithm** -- the computational procedures used to encrypt and decrypt information
- **Key** -- a value that causes a cryptographic algorithm to run in a specific way and to produce a specific cipher text (i.e., a 128-bit key)
- **Cryptanalysis** -- the science or "art" of analyzing a crypto system, either to verify its integrity or to break it for ulterior motives
- **Secret key symmetric encryption** -- a single key is used to encrypt and decrypt information
- **Public key asymmetric encryption** -- 2 keys are used, one for encryption, the other for decryption
- **Vulnerability** -- security flaw in systems
- **Attacks** -- the means to exploit a vulnerability
- **Threats** -- motivated adversaries capable of mounting attacks
- **Countermeasures** -- technical or procedural means of addressing vulnerabilities or thwarting specific attacks

The basic tenant of security is: Security is relative. Webster defines security as “Freedom from danger, harm, risk of loss, doubt, anxiety, or fear” . One can feel “secure” in an environment if one can provide the following attributes (if necessary):

- Confidentiality : The content is private (encryption).
- Integrity : The content has not be modified (digital signatures).
- Accountability : Both the sender and receiver agree the exchange has taken place. This of course includes non-repudiation (signatures).
- Authenticity : Both sender and receiver are known and trusted (access control).
- Availability : The system works under various conditions (firewalls, virus-scan, audits, knowledge).

	Confidential	Integrity	Accountability	Authenticity	Availability	Access Control	Auditability
SSLv3	Y	Y	(Identity)	Y	N	(Mappings)	(Logging)
Firewall	(IPSec)	(IPSec)	N	N	Y	Y	Y
Certification Authority	N	N	N	(ID)	N	N	N
SET	Y	Y	Y	Y	N	N	Y
Web Server	(SSL)	(SSL)	N	Y	N	Y	Y
Browser	(SSL)	(SSL)	N	N	N	N	N
VPN (IPsec)	Y	Y	N	[System]	N	N	N
Anti-Virus	N	(Viruses)	N	N	Y	N	N
Network Sniffer	N	N	N	N	Y	N	Y
Java	(Security API)	[Verifier]	N	N	Y	Y	N
Cookies	N	N	N	(Web Server)	N	N (Web Server)	N

But how does one accomplish these daunting tasks. The following figure illustrates how one might secure an e-business solution. This diagram shows many options and does not imply that all options shown must be implemented. All options are not shown (Single sign-on, smart-cards, biometrics devices etc...)



We can see there are many options and components in a security model. The performance expert will also notice there are potential bottlenecks and long path costs. Once again, balance is the rule of the day.

Performance Issues and Experiences

SSL

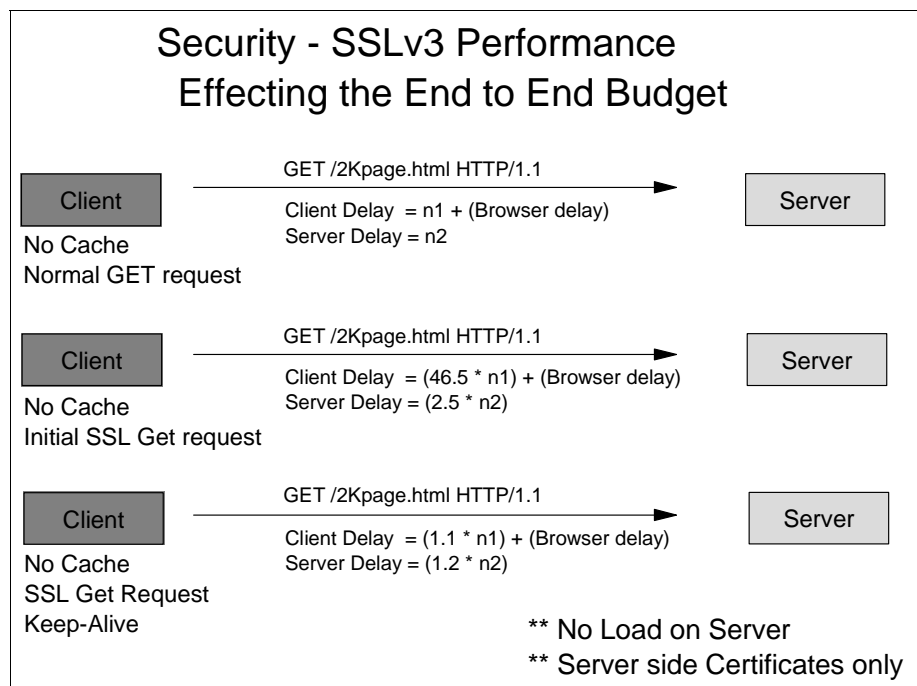
As we mentioned earlier there are two major encryption methods: symmetric (single key) and asymmetric (two keys). Symmetric encryption is much faster than asymmetric, but performance is not the only reason for choosing an encryption method. The major challenge with symmetric key encryption is key distribution. Since there is only one key everyone must have a copy of this key. Asymmetric encryption uses 2 keys, a public key and a private key. The public key is freely distributed while the private key is safe.

There are two major phases of SSL session encryption. These are: key/secret exchange and secure data communications. There are several different key exchange

algorithms (e.g. RSA, Diffie-Hellman, etc..). as well as data communications encryption algorithms (e.g. DES, IDEA, Triple DES, etc.). Each of these algorithms has performance attributes. For example, Diffie-Hellman key exchange is “faster” than RSA key exchange. Although this is true, Diffie-Hellman does NOT offer one important “security” provision. It does not guarantee identity. Unlike RSA, in which you are sure to whom you are speaking, Diffie-Hellman can make no such promise. So , choosing one algorithm based strictly on performance may be unwise.

SSLv3 (Secure Sockets Layer version 3) or TLSv1 (Transport Layer Security) has two basic “dialogs”. The first “dialog” establishes a session and exchanges secrets. This establishment comes in many flavors but the two most popular are “server certificate only” and “both server certificate and client certificate”. Which method one uses depends on the application requirements. The second “dialog” occurs when a session is already in progress. In the client section, an HTTP attribute (HTTP header field) known as *Connection*: with a value of *Keep-Alive* was introduced. If the “Keep-Alive” timer (typically 60 second, but, configurable) has not expired, the client and server can use the existing session id and secrets to communicate. This is a key performance issue, since most of the time “running” an SSL session is spent in secret negotiations (dialog one).

The following diagram illustrates the time required for session establishment and session resumption.



Note: the SSLv3 Performance diagram is a point in time measurement. The “curves” in performance are not shown. The client system is Netscape 4.03 Navigator browser on

Windows NT 4.0. The server system is IBM Domino Go Webserver on Windows NT 4.0. Both systems are IBM Pentium 300's, 128MB of RAM.

There are several interesting things to observe in the above diagram.

1. The greatest amount of time in SSL session establishment is spent on the client. The client has a lot of work to do.
2. Session establishment delay is measurable and will effect the end-to-end budget. The time it takes to set up a SSL connection should be taken seriously in the design.
3. When a session is already established, the delay in an SSL session is "reasonable". Once the SSL connection is established the dialog between the server and client go on with minimal impact. (Assuming no other system or network constraint). The "gotcha" is how many "logical sessions" can be performed within the "Keep-Alive" time.
4. Hardware encryption can greatly reduce the response time and CPU time required for the SSL handshake. For example, S/390 G3, G4, and G5 processors support hardware encryption.⁶
5. Conclusion - Use SSL only when it is required. (i.e. when confidentiality, authentication and integrity are required and not provided by some other means.)

Authentication and ACL (Access Control Lists)

Authentication will also have an effect on the end-to-end budget. The good thing is, the first time a user is prompted for his or her username and password, performance is the not the primary consideration. As long as it appears that the system is working to grant access the user is usually happy. But once the user is authenticated, he or she has little patience for additional delay.

The basic authorization technique used in a web server is to control access to part or all of the contents of a web server and is implemented via access control lists. Information then can be passed to programs based on the granting of access. Once the server loads the username and passwords, ACL performance is very good. This is the result of the client including the username and password (Base64 encoded) in the header. And, of course, the server ACL path is relatively short. Also, keep in mind that the Base64 encoding used by basic authentication is not encryption and if the packet is captured in flight, the userid and password can be easily decoded.

⁶ WebSphere Application Server 1.1 and related PTFs are required to exploit the enhanced performance.

Tests indicate that password files should be relatively small (less than 10000 users). Files with 100000 users can become a performance bottleneck. If a solution must support large numbers of users that must be authenticated, password files may be divided into multiple subset files (e.g. to support 100,000 users, divide the files into 10 files each containing 10000 users). This is supported in many web servers, including Domino Go WebServer. In this example, 10,000 users will consume about 700K of memory on an NT server. The first time a named ACL is called the server will load that ACL file into memory. Subsequent calls will access the information in memory.

Directory Server

Basic authorization is not always sufficient for enterprise access. There are two important mechanisms to extend this authentication. The first is to extend or replace the web server ACL function by mapping to a common backing store or directory server. The benefits of using a system or enterprise directory service include consolidating duplicate information into a single repository to make the network and applications easier to manage, and using an industry standard API such as the Lightweight Directory Access Protocol (LDAP) for all interactions with sensitive data. These advantages may outweigh the impact to performance, but (at a minimum) the network latency and number of trips across the network must be added to the budget calculations.

In addition to network latency and round trips there are architecture, design, and deployment considerations for the directory server itself that must be considered. Most directory server products such as the IBM eNetwork LDAP Directory provide several options to increase scalability and to provide load balancing. These features include replication (i.e. ability to create and run many instances of a complete directory to ensure there is a copy "close" to all users in the network) and split name space (i.e. to ensure that "local" data is available where it is needed without replicating it across the network where it is seldom / never used). The quantity of directory servers and their contents should be included in the architecture and design of the e-business solution.

The internal organization of data within the directory server itself can also have a major impact on the performance of your application. The namespace definition of an LDAP or X.500 directory server determines how each object and attribute is stored within the tree structure (directory / subdirectory). If, for example, your application needs to read five data "fields" (name value pairs) related to a person, you might be able to find the entry using its distinguished name (DN) and access all 5 fields, all with one LDAP call. Or, you may require 6 LDAP calls; one to find the starting entry and the reference pointers (relative distinguished names (RDN) to 5 other entries where the required information is stored. In some cases, the directory server is completely defined by a software vendor or by the customer and can not be modified to ease trans versal by your solution. In other cases, where your e-business solution is part of a major

re-engineering effort you can, and should, have a major say on namespace definition to ensure that the information you need is accessible by as few LDAP calls as possible. In either case, the number and complexity of the interfaces to a directory server must be considered in your overall design budget.

A second alternative to basic authentication using a directory server is to use SSL certificates for authentication and then perform the same mapping. This technique has the same impact as the first method, with the added benefit *and cost* of SSL.

Firewall

“Firewalls are the wrong approach. They don’t solve the general problem, and they make it very difficult or impossible to do many things. On the other hand, if I were in charge of a corporate network, I’d never consider hooking to the Internet without one.”
-- Charlie Kaufman **Network Security**

The two primary functions of a firewall are packet filtering and application proxy. Packet filtering is just that: filtering packets. Decisions to discard packets are made based on protocol, service port, IP addresses etc.. Packet filters are generally very good performers when there are just a few defined. As the number of packet filter rules grow so does the decision path length. Proxies, on the other hand, offer more robust security options. Simply stated, a proxy is a “middleman”. A client will access the proxy and then the proxy will access the server on behalf of the client. Proxies come in two flavors: transport/circuit and application. Transport proxies (e.g. SOCKS) work at the transport layer and require a “socksified” client. Application proxies require no modification to the client, but require a proxy for each application (i.e. HTTP, FTP, Telnet etc..) Think of a proxy as a translator. If you want to have a conversation with a French speaking person and needed a translator it would impact the speed in which you communicated. This can have a major impact on performance and response time. As we have mentioned several times, firewalls are implemented for security reasons. If firewalls exist in the deployment network they must be included in your budget calculations.

The National Software Testing Laboratories, KeyLabs Inc. and other groups have done performance bench-marking on firewall products. Typical traffic mixes are 75-80% HTTP and 20-25% FTP. The results vary from platform to platform and also vary depending on which options, such as logging, address translation and encryption, is enabled. One important conclusion is, compared to normal packet forwarding, firewalls with logging and proxy enabled can reduce throughput by as much as 50%.

Today’s firewall systems employ more than just proxy and packet filtering. A comprehensive firewall will do Stateful filtering/inspection, Network Address Translation (NAT), extensive logging, IP spoofing detection, SYN attack detection and anti-virus scanning. Each of these elements have a cost in the budget and must be considered in the overall performance budget.

Recommendations

There are very few e-business solutions that do not need some sort of security. So, it is safe to assume that security must be included in an end-to-end design and that security will effect the end-to-end budget.

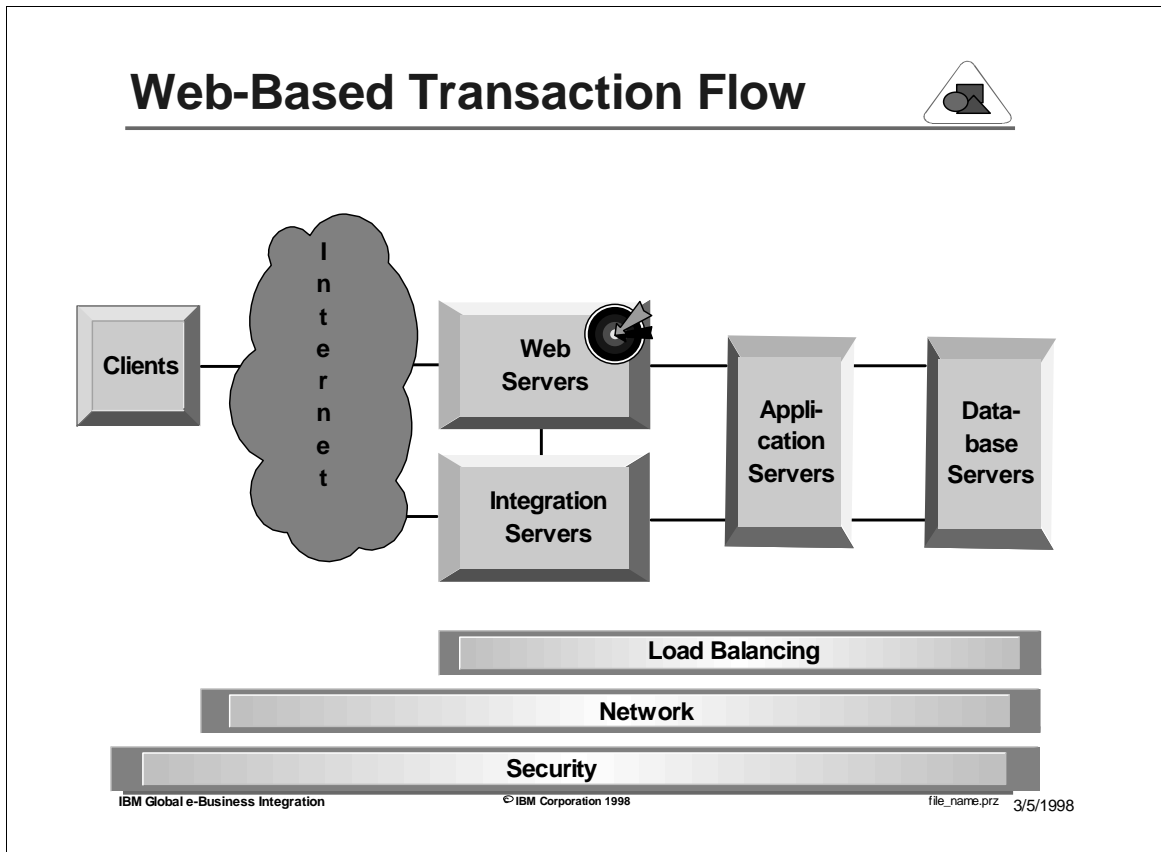
We must accept that we will implement some facet of security. We must also accept that security can, and often does, seriously impact performance. Therefore we must define the architecture and design, and our performance benchmarks or indicator tests with the security system (Firewalls, Proxies, Security Servers, Audit Loggers etc..) in mind.

The following is a list of general recommendations:

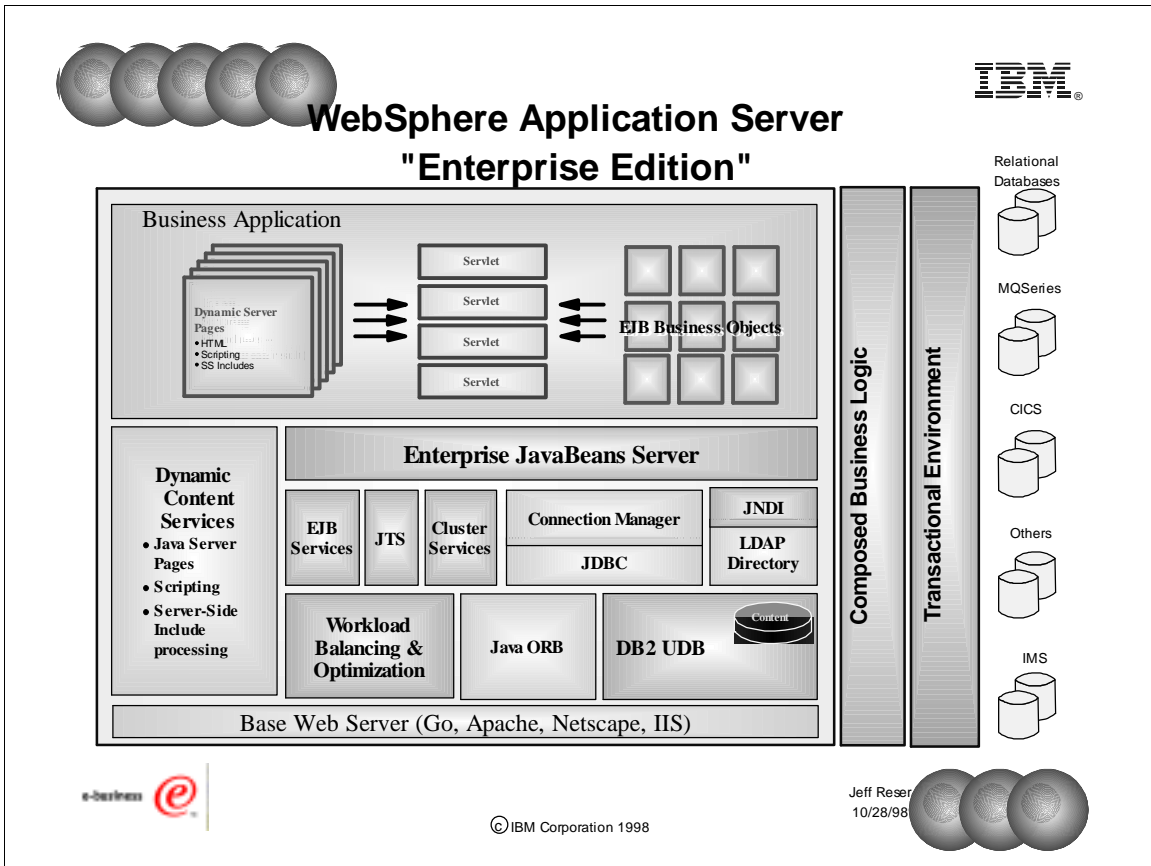
1. **The architecture and design of the e-business solution should include both the security and its impact on performance up front.**
2. **The overhead of security will impact the size and/or number of systems in the solution.**
3. **The overhead of using a directory server will impact performance.** Consider the number of servers and the namespace definition in the architecture and design of your solution.
4. **Do not over-design the security system.** If encryption is needed, only encrypt those elements that need encryption.
5. **Establish levels of trust.** Access does not have to be challenged at a trusted layer.
6. **Systems that include (require) an external Certificate Authority (CA) introduce additional and less predictable performance elements into the solution.**

Web Servers

Introduction



A web server functions as the hub of e-business solutions. The role of the web server has changed quickly and dramatically. The web publishing era, where httpd was only responsible for serving up static information is over. The web server has become the mid-tier, middleware application server connecting a web client to back-tier databases, transaction systems, and applications.



The chart above describes the components of the IBM WebSphere Application Server Enterprise Edition. This section of the paper will discuss many of the architecture and design guidelines for e-business solutions that pertain to the web application server. There is additional design and implementation guidance in the section on [Java](#) below.

Performance Issues and Experiences

Work Load Analysis

A web application server can perform a wide variety of tasks, from serving up static web pages to coordinating distributed units of work involving multiple back end systems. So, the first guideline for achieving good performance is to define and understand your workload. The need to understand the workload and its associated data flows was introduced in the [Client](#) section (know the steps involved in set-up, know how many files and applets will be loaded, etc.) and was also discussed in the [Network](#) chapter (define all data flows, define the number of round trips needed, etc.), so it should not be surprising that this requirement is repeated again here. And, as we have also said in each prior section, each interaction involving the web application server should be added to your budget.

Next, while you can run all types of workloads together on a web server, each type of workload has its own performance and scaling “signature” (use of resources, bottlenecks, etc.) Your e-business application architecture and design should take this into account; certainly if you have determined you will need more than a single web server and perhaps even if your initial assessment has determined that one web server is sufficient.

The ESS NC transaction pattern defines two logical nodes for web servers;

- An informational Web node which serves static and dynamic Web pages to users
- A transactional Web node which can access back-end systems in the trusted network for transaction processing activities.

This is a valuable segmentation for the physical implementation as well. We have found that server specialization is very common, i.e., to segregate Web servers by tasks and tune for each task differently. For example, you can segregate servers for banner advertising or database serving from the servers handling static page content. By segregating servers into task oriented groups, and monitoring the performance of each group (and therefore each task) separately, site response can be tuned more effectively.

After you segment the tasks and assign one type of work per web server, you can then evaluate the performance and scaling issues for the chosen task and select the platform and infrastructure with the “best fit”. The informational web LIB and the transactional web LIB can be served by two different types of hardware server, characterized as expensive “thoroughbreds” and commodity “pack mules”.

- *Thoroughbreds*: Handle computation intensive loads with high-end machines
- *Pack mules*: Handle static Web serving by spraying static requests across as many cheap commodity Web servers as are necessary given the traffic.

There are many ways to make an Internet e-business solution a success. However, even with good historical data and the best planning the Web can be unpredictable. Unplanned events can occur which dramatically drive up web activity in an unpredictable way.

For these reasons, we recommend that there be an expanded ‘load shedding’ plan that allows for the unexpected and helps hold down the cost of the overall solution. Some recommendations are:

1. Plan for alternate content with reduced complexity (done at Nagano)
2. Modularize content to allow selective “abandonment”
3. If using pack mules then be able to add/delete individual servers at will

Threads

Some web servers allow the configuration of both a minimum and maximum number of threads. Pick a reasonable, but large enough number of threads to handle your peak workloads, then set the maximum number of threads equal to the minimum number. This avoids the overhead of creating and destroying threads during peak processing hours. Experiment to find the right number of threads on your system. Testing for the Nagano Olympics has shown that picking too high a number can decrease your overall throughput on the system. Too low a number can even cause server failure. Remember in your calculations that the web server uses some of the allocated threads for it's own processing. In the case of the ICSS web server version that was tested for instance, nine threads were used by the server for it's own purposes.

Limit the use of Server-side includes (SSI)

Server-side includes (SSI) allow you to insert information into CGI programs and HTML documents that the server sends to the client. When server-side include processing is enabled, the web server will parse each byte of every HTML file and CGI program searching for the existence of an SSI directive and, if found, process it. This is a great feature for processing dynamic content, but it requires a large amount of CPU processing.

SSI processing can be controlled by the use of the `imbeds` directive. If you do not use SSIs, set `imbeds` off in `/etc/httpd.conf` file.

URL links

URL suffix processing (multi support) is overhead for any web server. This occurs when a URL does not exactly match the templates on the `PASS` directive. For example, if the URL is `/oh_boy.html` and your file name is actually `/oh_boy.html.ascii`, the Domino Go Webserver will do suffix processing and eventually return the file `oh_boy.html.ascii`. The Web server appends all known suffixes to the file name looking for a match. Eventually, the correct file will be returned, if it exists, but the process is CPU intensive.

Caching

The addition of memory to a system almost always improves performance. This is because physical I/O is a relatively expensive operation in terms of latency. It makes intuitive sense then, that by dedicating memory in a web server to store frequently accessed HTML pages and images, you will improve performance. As a rule of thumb, your web server should have enough RAM to accommodate all network buffers, frequently used applications, images and HTML, including those mounted via DFS. This is especially important for dynamically generated pages which can be reused, as was the case in the Olympics. For example, the IBM Internet site for the Nagano Olympics contained results pages for Free Style Skiing. These pages were generated periodically using Net.Data as new certified result data was entered into the system. Subsequent to the first request for the results page then, the Free Style Results page was stored in memory, to be retrieved over and over again by different users looking for the same results information. This “pre-request” generation of relatively static data improved response time. Server resources were efficiently used since frequently requested pages needed to be regenerated once in order to service many different users.

Logging

Web server logs, in particular the Access Log, record important information about the use of the web server. However, these logs can become very large and should be pruned and/or archived regularly. Logging can have a surprisingly large impact on response time and throughput. For this reason, you will often find that vendors turn logging off for bench marking purposes.

Of course few, if any, production sites would turn logging off. Because high use web sites have correspondingly high logging activity, logs should be placed on the fastest devices available. Striping and mirroring of logs can also help to improve performance by streamlining I/O operations. Turning off unnecessary directives such as IdentityCheck, or other log files such as the referer log can minimize logging overhead. Log analysis tools can help you to understand the way the pages in your system are used, allowing you additional insight into the proper layout of your site as well as helping you to identify opportunities to cache the most frequently used pages on your site in order to improve performance. Use the logs to identify your peak processing periods. Then collect information from your peak periods to use for trend analysis. You will particularly want to look for statistics like the number of connections per second and the number of hits per second to understand your workload. This will help you to anticipate the need to add additional servers or to upgrade existing server resources. By actively monitoring your workload, you can become proactive in managing the workload to minimize response time and maximize throughput.

CGI and Server-Side Java

The IBM Application Framework for e-business defines the infrastructure for developing e-business solutions. This includes the relationship between the web server and server-side Java elements. IBM's analysis of the performance characteristics of this environment provides some useful guidelines for e-business solution designers, including:

1. Any of the techniques used to connect a web server to application logic (CGI, in-process API, Java servlets, etc.) should be insignificant when compared to the time required to execute the application logic.
2. Most existing web applications have been implemented using CGI. IBM's analysis indicates that Java servlets provide 4X to 10X better throughput compared to CGI.
3. Java servlets run faster if they are instantiated and preloaded in `servlet.properties`. Servlet invocation by class name rather than instance name is slower.

You will find additional guidelines for designing e-business solutions using Java in the [Java](#) chapter.

Network

If there is no contention for either CPU or memory resources on your system, and you are experiencing performance problems, you may have a network issue to resolve. After all, while incoming web requests may be relatively small, outgoing web responses can contain large graphics, applets, video or audio files. It is important to make sure that the number and size of the TCP buffers be tuned appropriately on the web server platform. Because the size of requests coming into the server is so different than the requests going out of the server, many sites, including the Nagano Olympics site have routed incoming requests along relatively "thin" network pipes such as Token Ring while routing their output requests along relatively "fat" network pipes such as FDDI.

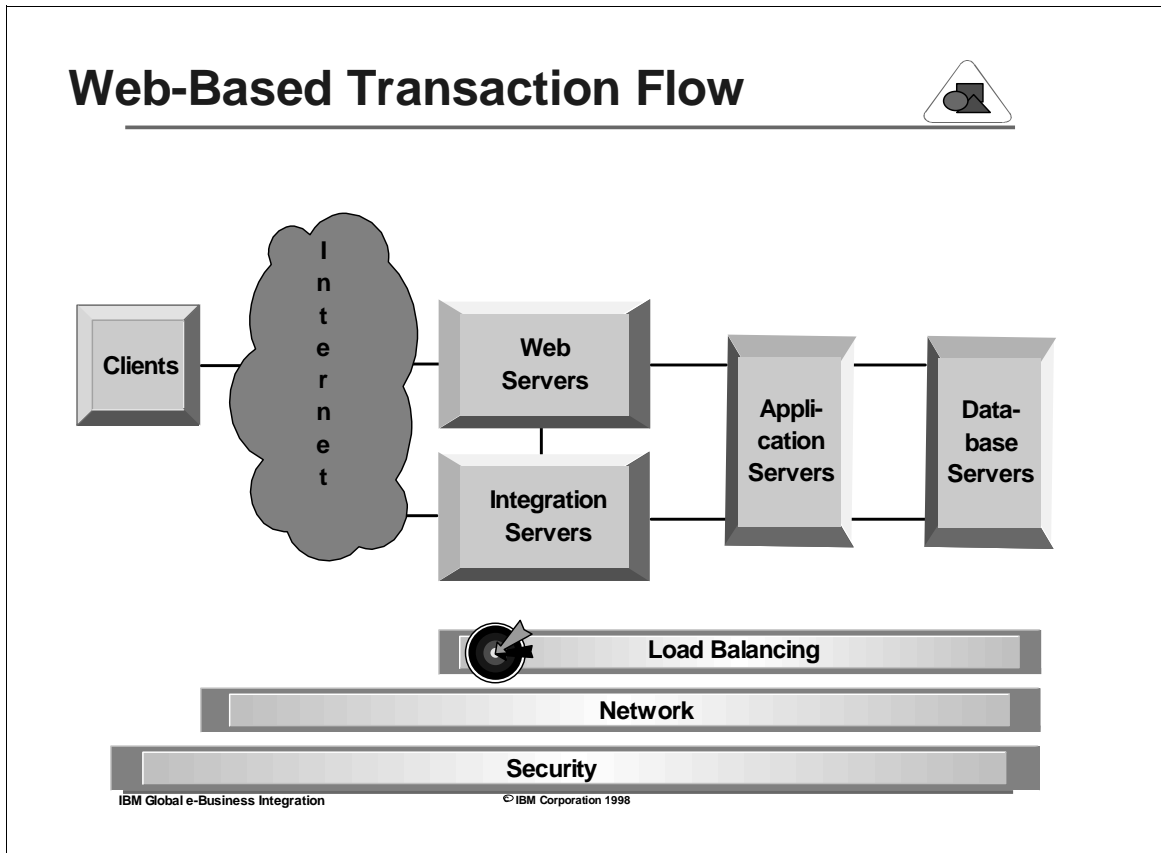
Recommendations

1. **Analyze your application workload**, segment into informational and transactional types and deploy on separate systems of appropriate strength
2. **Plan for peak access and design a 'load shedding' option**
3. **Set the minimum number of web server threads at a high enough level** to accommodate daily "peak hour" loads to avoid synchronous allocation of threads at peak request time.

4. **Monitor server statistics**, or set alerts to determine when normal settings have been exceeded. If settings are exceeded on a frequent basis, consider increasing the minimum number of threads on your server.
5. **Use fast devices for Web Server logs**. Stripe your logs across devices when possible.
6. **Locally cache all objects** which will be referenced frequently.
7. **Java servlets are much faster than CGI-BIN** programs and run faster if they are preloaded.

Load Balancing

Introduction



The prior chapter included a discussion on how logging may be used to anticipate and plan for growth of the web application server configuration. Simply stated, if you begin with one server platform and approach maximum capacity, there are three alternatives to increase throughput:

1. Change out processor - move from the current processor to a faster processor
2. Add processors or nodes - if your current processor supports SMP
3. Cluster processors - load balance among a group of processors

These options are not mutually exclusive of course, and each has its advantages and disadvantages. You can even implement a combination of all three. It is beyond the scope of this paper to explore the performance characteristics of e-business solutions among all possible permutations, but we will discuss one of them, clustering, in some detail; both because the implementation is itself based on Internet technology and also

because the type of load balancing you choose will have an impact on your application design.

Load balancing designs range from the relatively simple “round robin” Domain Name Systems (DNS) approaches to more sophisticated dispatchers such as the IBM Network Dispatcher that include server side monitors.

The relatively simple round robin type approaches for load balancing usually dispatch server requests to members in an active server group in an alternating pattern or to a specific server for a window of time. This type of load balancing system often works by using the calls made to the Domain Name System to resolve site names into IP addresses. Round robin type designs return alternating IP addresses for the servers that make up the common server pool. This type of load balancing system usually lacks a feedback loop from the servers and tends to dispatch work to servers regardless of their current workload. The systems making the requests of the servers also tend to remember which server they were initially told to use and direct all future requests for that name to the same server.

Unless the workload is very consistent across all the clients in terms of the server load required and the volumes requested, and the servers are identical in capacity, the round robin approach can cause some servers to bear more load than others. If some users generate requests for relatively more CPU intensive CGI calls or other types of dynamic pages versus some users requesting static pages that may be in cache, the round robin approaches usually does not balance the workloads. Because the round robin approach usually is not aware of current server workloads or the impact of the request being assigned, it just assigns the first request from a user to the next server in the rotation. This approach is more of a user allocation approach versus a true load balancing approach.

A more sophisticated approach is load balancing is provided by web server clustering. Most web publishing applications (i.e. those that do not do transactions that require several interactions between a browser and web server) will benefit from a load balancing configuration. An e-business solution (i.e. that include transactions) can take advantage of clustering by implementing specific session data techniques ranging from:

- **Low cluster synergy** with no direct cluster support. This implementation takes advantage of the “sticky ports” feature of Network Dispatcher to ensure that all transactional interactions (sessions) are run on the same web server.
- **Medium cluster synergy** is achieved by keeping session data in a cookie and passing it along with each interaction. This helps ensure load balancing, but incurs significant overhead for processing the cookie each time.
- **High cluster synergy** for e-business solutions is implemented by storing the session data in a distributed data store available to all web servers in the cluster.

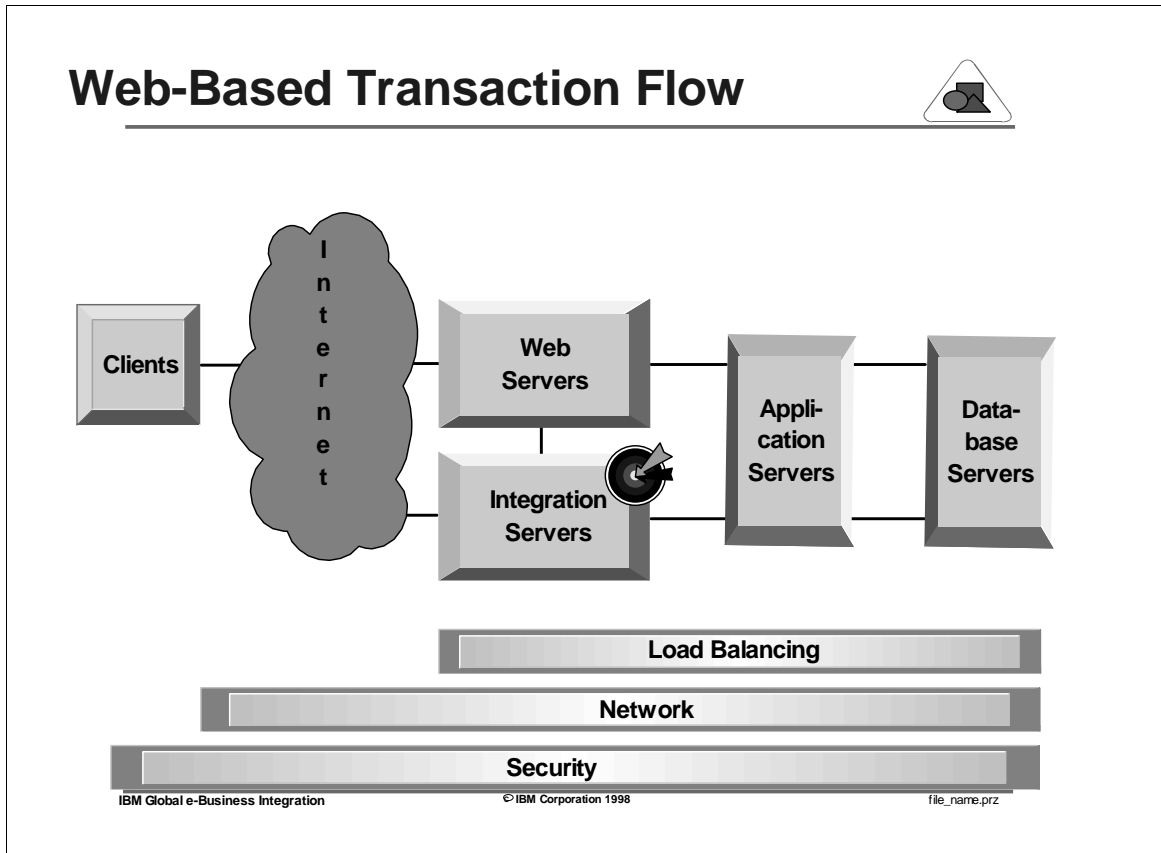
Both the Medium and High Cluster Synergy options require modifications to the e-business application. A detailed comparison of the three options is contained in Appendix C - Availability and scalability of web applications using session data and clustering.

Recommendations

1. **The type of load balancing design used can greatly affect response time and consistency of performance.** The round robin type do not actually achieve load balancing, but do provide limited user balancing. Servers can be overloaded and response times can be inconsistent across the user population.
2. **Metrics are important.** We have learned from experience that the selection of monitoring algorithms and logs used to make workload assignment decisions can make a significant difference in the load balancing design's success. For example, measuring CPU utilization on a system that is actually I/O constrained will lead to improper workload dispatching. It is important to calculate system capacity and conduct performance testing on each system to determine the sensitive metrics.
3. **Cluster design and e-business application design must be done in concert.** In particular, the success of the application is going to depend on how it handles session data and how it interacts with the load balancer

Integration Server - Object Request Brokers

Introduction



Several vendors in today's marketplace are providing *Object Request Broker* (or *ORB*) products that conform to the CORBA specification. These products vary widely in terms of back end integration capabilities, scalability, fault tolerance, manageability, and tool support. They also vary in terms of performance characteristics.

IBM's Component Broker, including CConnector and CToolkit are the industry's most comprehensive implementation of CORBA. The WebSphere family integrates the Web server, Web commerce, distributed transaction processing and distributed component technologies of the IBM WebSphere Application Server, Net.Commerce, TXSeries™ and Component Broker products.

This chapter will discuss several of the key performance-related issues that must be considered when implementing distributed CORBA-based systems. You can use the discussion points in this chapter to help you:

1. Evaluate the performance characteristics of the ORB architecture
2. Understand what you can do to can improve performance
3. Design your e-business solution with performance in mind

Performance Issues and Experiences

Application Design Considerations

It's important to design distributed object applications in a way that minimizes unnecessary network traffic and information flow. Each synchronous method called by a client or server on a remote object requires a round trip across the distributed network. (Also see the discussion of network latency and its relationship with multiple round trips in [The Network](#) section of this paper.) The following table provides some compare / contrast examples of how to achieve good performance.

<i>Performance-Unfriendly</i>	<i>Performance-Friendly</i>
Client invokes <i>create</i> method on a remote factory object, supplying the primary key attribute(s) for a new object to be created. A reference to the newly created remote object is returned, and the client invokes multiple "setter" methods to initialize the remaining state of the new object.	Client invokes <i>create</i> method on a remote factory object, supplying ALL of the attributes necessary to fully initialize the new object that is created. A reference to the newly created (and fully initialized) remote object is returned.
Client invokes a method on a remote object that changes the value of several attributes for that object. The client then invokes individual "getter methods" to obtain the state of each of those changed attributes.	Client invokes a method on a remote object that changes the value of several attributes for that object. The result of the method call returns the new values of each affected attribute in a single stream.
Client calls several methods to queue up individual items of work for a remote object. Client then calls a "dolt" method that instructs the object to process the items.	Client invokes a "dolt" method on a remote object, passing in all of the work items as arguments of that single call.
Client repeatedly calls a remote object for distinct items of information. The information is locally manipulated and summarized by the client for presentation.	Client makes a single remote method call, and the remote object prepares and then returns only the summarized information as a result.
Client invokes a query on a remote object that returns a collection of references to other remote objects. The client then invokes multiple "getter methods" on each of the remote objects for local presentation.	Client invokes a query on a remote object that returns "rows" of attribute values from other objects local to itself. The client uses these remote object values without any additional network interactions.
Client invokes a method on a remote object that returns a <i>very large</i> amount of data across the network.	Client invokes a method on a remote object that locally compresses result data and then returns it. The client decompresses this result data upon receipt.

Note that ORB vendors are increasingly offering "smart proxy" capabilities and will also be providing implementations of CORBA's pass-by-value specification. These capabilities further promote the design and use of performance-friendly interfaces.

The following techniques also have a beneficial impact on overall performance.

1. Lazy initialization

Server-side objects may incur significant overhead when producing the values of their attributes if access to back end systems is required. By producing these values only when absolutely necessary (that is, when first requested to do so by clients), overall system resource utilization can be minimized.

2. Appropriate use of “Singletons”

This technique can be viewed as a special case of lazy initialization. There are times when a single object may be used from multiple points in a client’s process (for example, the Naming Service root context, the Factory Finder of a given machine, the default Event Channel, and so on). The overhead of obtaining a reference to such “Singleton” objects should only be incurred one time. This overhead can be encapsulated and paid for once using well-known methods that are accessible from anywhere within a client program.

3. Client-side caching

The same attribute of a remote object might be used in several places within a client application. It might be displayed as text in one place and as an image in another. It might be used to properly enable or disable buttons and other visual controls. By obtaining the remote value only once and then locally caching it, overall processing expense can be minimized through the repeated use of the locally cached value.

4. Asynchronous communications and use of remote queues

By using “one-way” method calls and the rich capabilities of CORBA asynchronous method invocations, it is possible for the end-user to *perceive* improved performance. The work to be done can be off loaded into queues and scheduled for subsequent processing at optimal times. Naturally, the applicability of this technique is highly dependent on the nature of the particular application domain.

Physical Topology

Considerations that pertain to physical topology include:

1. Initiating ORB communications at appropriate points in the topology

For traditional “fat” desktop clients, it’s acceptable to initiate ORB communications from the end-user application. But for e-business solution for Internet clients (and for very thin network clients in general), it is more appropriate to initiate ORB

communications on the server side. In other words, we might prefer to replace the following configuration...

Java applet running in Web browser = ORB client

...with the following “thinner” client configuration

HTML client running in Web browser interacts with Java servlet via HTTP
Java servlet running on Web server = ORB client

The thinner client configuration allows ORB communications to run on faster and more powerful machines, and also eliminates the overhead of downloading an ORB to the client.

2. Choose the appropriate number of physical tiers

From the standpoint of performance, it *may* be desirable to combine the middle and back end tiers into a single physical tier. If we put server-side objects onto the same (potentially very powerful) physical processor as the back end systems on which they depend, we can further reduce network traffic. Of course, this reduced number of network exchanges must be weighed against the reduced number of total processors involved in the overall solution, security requirements, etc.

3. Place objects with high affinities onto the same servers

Frequently a single client method request will result in multiple collaborations between many remote objects. From a performance perspective, remote objects that frequently collaborate should be placed onto the same server to the extent that this is practical. This keeps the interactions between these objects local (that is, no network traffic is incurred between such objects). This guideline applies both to domain-specific objects (like Funds and Accounts) and, where possible, to “system objects” (like Transaction Coordinators) that may be “silently” involved in the fulfillment of a method request.

4. Cluster the right number of servers within a given tier

A robust distributed computing solution typically clusters “server clones” into “server groups”. Each server in a particular group is capable of handling the same kinds of requests. By having client requests routed to appropriate servers at appropriate times, multiple processors can efficiently share in the total work that needs to be done. This means that some of the work might be accomplished in parallel, thereby increasing overall throughput. Importantly, deploying multiple servers also facilitates high availability through the use of fail over mechanisms.

5. Place the most powerful processors into the middle and back end tiers

This point is really self-evident, but we mention it here for the sake of completeness.

The importance of higher-powered machines beyond the first tier becomes correspondingly greater for enterprises having large numbers of first-tier clients and for domains that are particularly compute-intensive with respect to the middle and back end tiers.

Backend System Integration

Integration of ORB-based systems with back end systems also have several performance issues, including:

1. Pre allocating and caching resource manager

Some of the cost associated with accessing resource managers from middle-tier objects involves establishing connections to those resource managers. By pre allocating and reusing connections, it is possible to greatly reduce the overhead of defining new ones.

2. Caching facilities local to the middle tier

Caching state information accessed from back end systems into the middle-tier environment can promote good performance. This is achievable through prefetch and look-aside algorithms which ensure that the back end will only be accessed as often as is absolutely necessary. Note that it's common, especially when integrating with existing legacy information systems, to access back end state information required by one object and, in the process, to encounter state information needed by one or more other, related objects. Caching this information as it becomes available can significantly improve overall system performance. Finally, a smart caching mechanism can ensure that updates are only made to a back end system when the underlying state information of a given object has actually changed.

3. Optimistic locking mechanisms

In some cases, applications place large numbers of "unnecessary" locks on resource managers. We say these locks may be unnecessary in the sense that no attempts to use the resource concurrently actually occur. In these situations it would be better to place *no* locks on the resource managers, and instead check for conflicting usage as part of transactional commit. Specifically, you should consider locking all affected resources only once; at the end of a given unit of work. Then compare the relevant resource manager values at that time with the values obtained when the unit of work began. In the absence of intervening changes, the current unit of work can be committed. Otherwise, it can be rolled back with an exception returned to the client. In either case, you may be able to reduce overall locking overhead and improve total system throughput and performance. Note that this "optimistic" locking strategy is inappropriate for some types of applications and domains. When updates to the same resource occurs on a frequent basis

traditional (or “pessimistic”) locking mechanisms should be used. It’s also worth mentioning that multiple resources should typically be accessed by multiple applications in the same sequence so as to reduce potential “deadlock” situations.

4. Implementation “pushdown”

ORB-based solutions that must coexist with legacy systems should complement (versus compete with) these systems. For example, a query invoked on a virtual collection of objects whose state maps to a relational database manager should first be translated into native SQL statements. These SQL statements, processed using the optimization technology provided by the database manager, can then return a result set whose values implicitly identify candidate objects satisfying the query. In this way, a minimal amount of processing is required in “object space”, leaving the bulk of the work to be handled instead by resource managers that have been perfecting high performance solutions over the course of many years. As a final point that is specific to our query example, it’s notable that an object-based implementation of query should also be able to return multiple elements back from a single method call, and that the query result set should be demand-driven (meaning that objects should only be activated on the server if a client actually requests them).

Tuning and Managing Performance⁷

In order to achieve good performance, ORB-based solutions must have a systems management dimension. This includes capturing performance-related information and tuning key system parameters across several products and components, including:

1. CBCConnector Tuning
2. Operating System Tuning
3. Java Virtual Machine Tuning

CBCConnector Tuning

1. System Management Agent - adjust or eliminate polling
2. Logging of client initiated events - evaluate turning off logging
3. SMP systems - dedicate the server to a single processor

⁷ for a complete description see “Component Broker for Windows NT and AIX Planning, Performance, and Installation Guide” SC092798-00

4. Adjust Thread Pool based on peak loads
5. Locate transaction log on fastest disk

Operating System Tuning

This section points out Operating System specific functions that can be tuned to improve CBCconnector performance. Many more details regarding operating system tuning can be found in the following references.

For the Windows NT operating system, the Windows NT Resource Kit contains a wealth of information regarding performance monitoring, diagnosing performance problems, and available performance tools. For information, see the Microsoft Windows NT Workstation Resource Kit.

There are also many sources of information for S/390 tuning and performance planning, including:

- S/390 UNIX Services Tuning -
<http://www.s390.ibm.com/oe/bpxaltun.html>
- S/390 Web Server Tuning -
<http://www.s390.ibm.com/oe/perform/dgwperf.html>
- Capacity Planning for Web Applications on OS/390 - S G245168-00 98/12/17.

A good source of information for the AIX platform is the AIX Performance Tuning Guide Versions 3.2 and 4, SC23-2365. This book includes several discussions on various general performance topics as well as AIX-specific performance topics. General (applicable to all or most systems) performance items include:

1. Locate system paging space on a disk that is separate from where the CBCconnector transaction log and Database are located.
2. If the system has multiple roles (for example, CBCconnector Server, Print Server, Mail Server, and so on), adjust the CBCconnector Server process priority to be more favored than the other non-critical tasks.

Java Virtual Machine Tuning

Heap Size

Care must be taken when tuning the **minimum** heap size. JavaBOs interact with objects on the C++ server using Java proxies to these objects. These proxies and the corresponding C++ implementation objects to which they point are not freed until the proxies are garbage collected. Since C++ objects can be an order of magnitude bigger than their lighter-weight Java proxies, setting the initial heap size too large can cause over-commitment of C++ memory on the server.

For example, setting the minimum heap size to 32 megabytes could cause C++ memory allocations of more than 320 megabytes even before the first garbage collection starts to run. This problem can be avoided by setting the minimum heap size smaller. Garbage collection then runs more frequently and dramatically reduces the C++ memory requirements.

For the above reasons, the minimum heap size should only be modified when the memory requirement of an application is known. For example, the application may pull in additional Java class libraries whose run-time sizes are known. Or experimentation may have determined the memory usage requirements. In these cases, enlarging the minimum heap size can potentially result in better performance due to fewer garbage collections before sufficient memory is allocated for the application's working set.

Much of this discussion is independent of any particular ORB-related product. Nevertheless, the author of this section wishes to acknowledge the rich runtime, systems management, and development dimensions provided by IBM's Component Broker middle ware. The robust product architecture underlying Component Broker has in fact been the inspiration for much of the content in this chapter.

Recommendations

1. **Combine client-object interactions** to reduce the number of invocations / round trips required across the network
2. **Use client side or server side caching** to avoid multiple re-instantiations of objects
3. **Use asynchronous method invocations** to reduce perceived response time
4. **Move the ORB client to the server** to reduce the size of your client workstation
5. **Pre allocate pools of resource manager connections.** Release connections back to the pre allocated pool for reuse
6. **Cache state information**
7. **Use optimistic locking** to reduce resource contention
8. **Leverage the performance characteristics of back end servers**

Integration Server - Message Servers

Introduction

Messaging and Queuing software allows secure, asynchronous delivery of information by distributed applications that communicate across many operating systems and networks. Asynchronous message delivery is important as it allows one program to send data to another program even when the second program is not available. By storing messages on persistent queues, messages can be held for guaranteed delivery even when the network between the two programs is not operating.

Factors that affect the performance and throughput of a MQSeries system include:

1. Persistence
2. Message Characteristics
3. Application Design
4. Network Utilization

Persistence

Persistent messages can be recovered after the restart of the message queue manager because they are logged. Non-persistent messages remain only in memory, are not logged and cannot withstand a system crash. Since persistent messages must be synchronously written to a log before commit processing can complete, a significant portion of the time taken to process persistent messages is the logging process. Because logging operations incur I/O delays, we recommend that you use the fastest device available for the log, on a volume with low contention. Use RAID devices or "stripe" MQ logs across several disks to streamline input/output operations. For optimal performance, make messages persistent only when necessary.

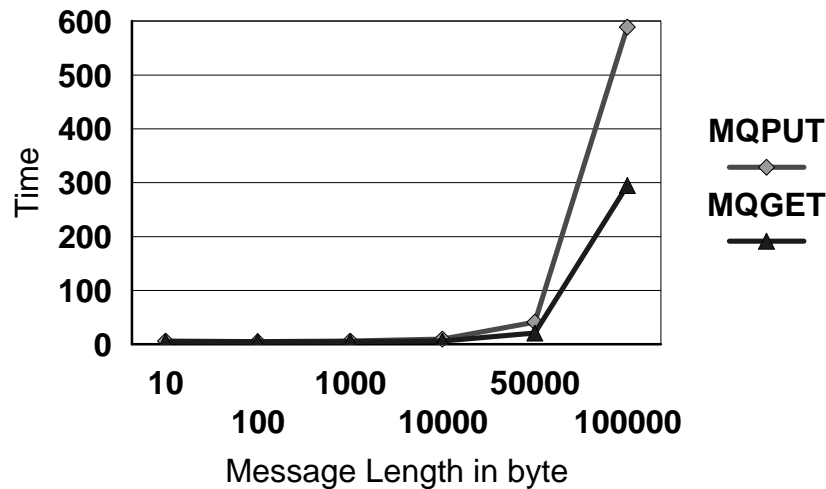
Message Characteristics

Message Size

The following figure illustrates the effect of message size on the time taken to process a message. Messages larger than 10,000 bytes take proportionally longer than shorter messages, with the time increasing with message size. All messages smaller than 1000 bytes in size take about the same length of time to process regardless of length,

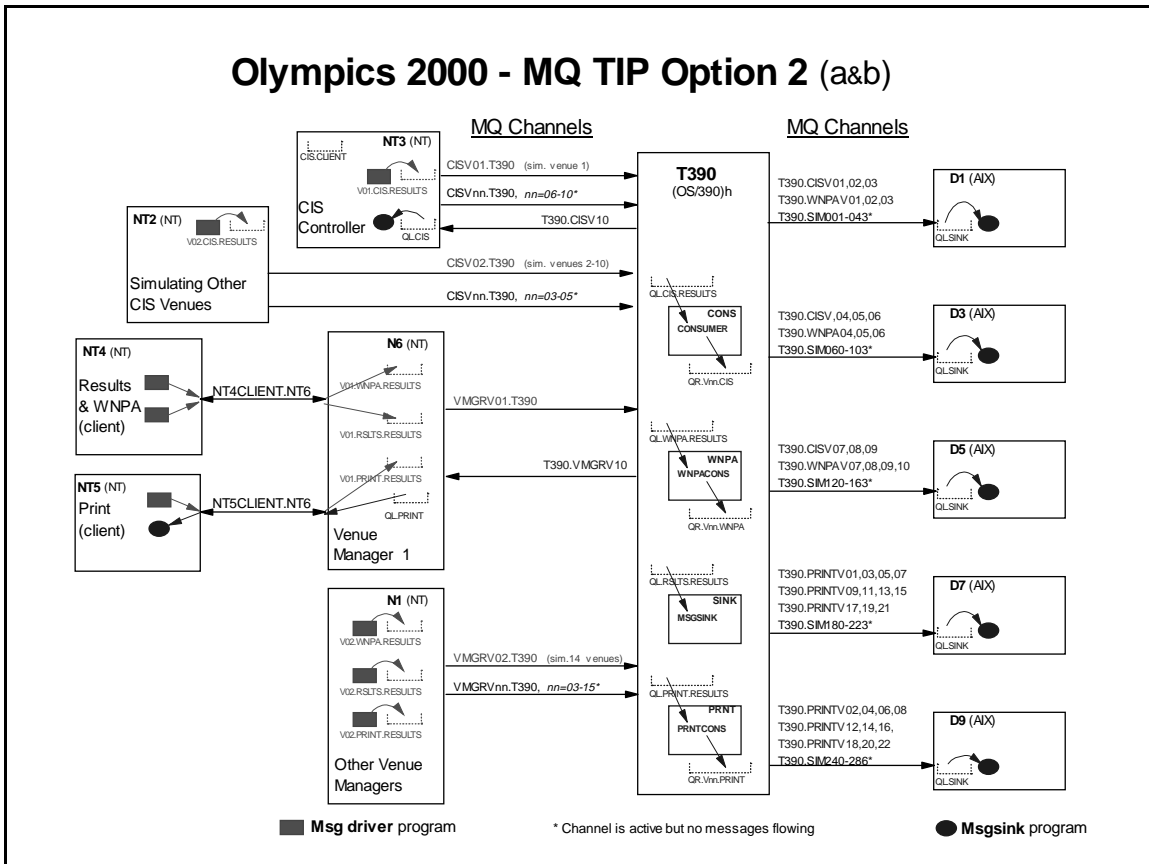
but shorter messages require less disk space during processing. Try to keep the message length under 10,000 bytes long if possible. However, sending one long message is more efficient than splitting a single message into a series of shorter messages.

MQ Performance vs. Message Size



Source: IBM Hursley, doc no. CPFIUSER.MQESAPRF.SCF

In a TIP for the Sydney Olympics, we ran workloads ranging from 2.75 to 13.75 messages input per second. Message sizes ranged from 90 to 150,000 bytes. Output messages ranged from 9 to 20 messages per second as large messages representing Print Distribution were sent to an additional 7 output queues as persistent messages. These large messages stressed the MQ logs and saturated the 16 Mbps network. Still, local and remote response times were impressive for all runs. Small messages (a thousand bytes or less) uniformly achieved subsecond response times, while large messages (14,000 bytes and over) were well under two seconds in our mixed workload.



In this scenario, an MQ application was used to send a mixture of persistent and non-persistent messages to CICS MVS, where a long-running transaction program took them off their respective queues to be processed by the target MQ subsystem. Once processed, the messages were queued to their destination NT and AIX MQ target systems.

Message Priority

Messages can be stored on the queue in different retrieval orders including FIFO (first in, first out) sequence, or FIFO within priority sequence. An application can either retrieve a specific message if the message identifier is known, or the first message on the queue. If the message queue is organized by priority, the first message with the highest priority will be retrieved, followed by the next messages within that priority. Retrieval continues until all messages in a priority are retrieved. Some products allow the retrieval of a specific message (by message id or by correlation id) either by using a sequential scan of the message queue; or by generating an index based on a message identifier.

Application design

A message queue must be opened before an application can "get" or "put" a message to the queue. The queue "open" operation is relatively expensive compared with the "get" or "put" of a message. Therefore, strive to minimize the number of "open" operations to a queue.

Similarly, an application must establish a connection to a message server prior to accessing its message queues. To minimize the time and resources consumed by connection handling, strive to minimize the number of "connect" operations. If possible, design an application to connect only once during of time in which queue access is needed. In our Olympics TIP, triggered applications which issued "gets" and "puts" to queues were started only once - when the first message of a predefined type arrived. These programs then ran in a "wait for input" mode for the duration of our processing period.

Network Utilization

If a message must to be sent to multiple recipients on one or many target machines, explore the capability of the message server to "multicast" the message. Multicasting allows the message server to send a message to multiple queues with one operation,. Therefore, multicasting has the potential to significantly reduce network traffic.

Recommendations

1. **Use persistence only when necessary.** Many messages can be recreated or resent if necessary.
2. **Use very fast log devices, and stripe log datasets when possible.** Logging occurs even without persistent messages.
3. **Eliminate unnecessary text in messages.** Not only do larger messages take additional network bandwidth, but they must be translated when crossing platform boundaries.
4. **Use long running triggered programs** rather than reloading and initializing message processing applications.

Integration Server - Host Access

Overview

One approach to transforming existing applications to e-business solutions is to extend existing host applications to the company Intranet or even the Internet. The general objective for this type of solution is to replace existing 3270, 5250, or other “green screen” applications with an updated browser-based graphical user interface. An example of this capability is IBM’s eNetwork Host On Demand, which provides Java-based host access from within a browser. This implementation works well for users that need occasional access to a host-based application, especially when deployed in company Intranets where the client system software and capacity is defined, or at least understood. Some e-business applications evolve in phases; where the initial web solution is deployed into the Intranet and later opened up to end user access via the Internet. An e-business solution intended for wide use by end users with a variety of client systems must be able to scale and provide good end user performance. Host on Demand may not provide acceptable response time and throughput in this environment. Instead, you should consider designing your solution based on a product that provides a better “match” to Internet requirements. IBM Host Publisher provides HTML-based pages that connect to host systems and includes a distributed infrastructure for scaling as well as page and session caching for better interactive response time.

Host Publisher

Host Publisher, a function of Communications Server for Windows NT, provides a Web-to-host solution specifically designed to address the unique characteristics of the Internet.

The Host Publisher components can be implemented on a single computer, or distributed among several systems. For example, the Page Server component used to generate the HTML pages from the host screens can run on the same system as the web server, or can run on a separate system if the web server is near capacity. Additional Page Servers can be added and accessed via a clustering technique that provides scaling and load balancing among the Page Servers. Experiments show that response time in a loaded system can be substantially reduced by clustering.

Recommendations

1. Host Publisher's Page Server tends to be CPU bound, rather than Memory or I/O bound. **A larger CPU size will provide greater response time and capacity.**
2. **Use Connection Caching and Page Caching whenever possible.**
3. **Turn off Logging, unless needed.**

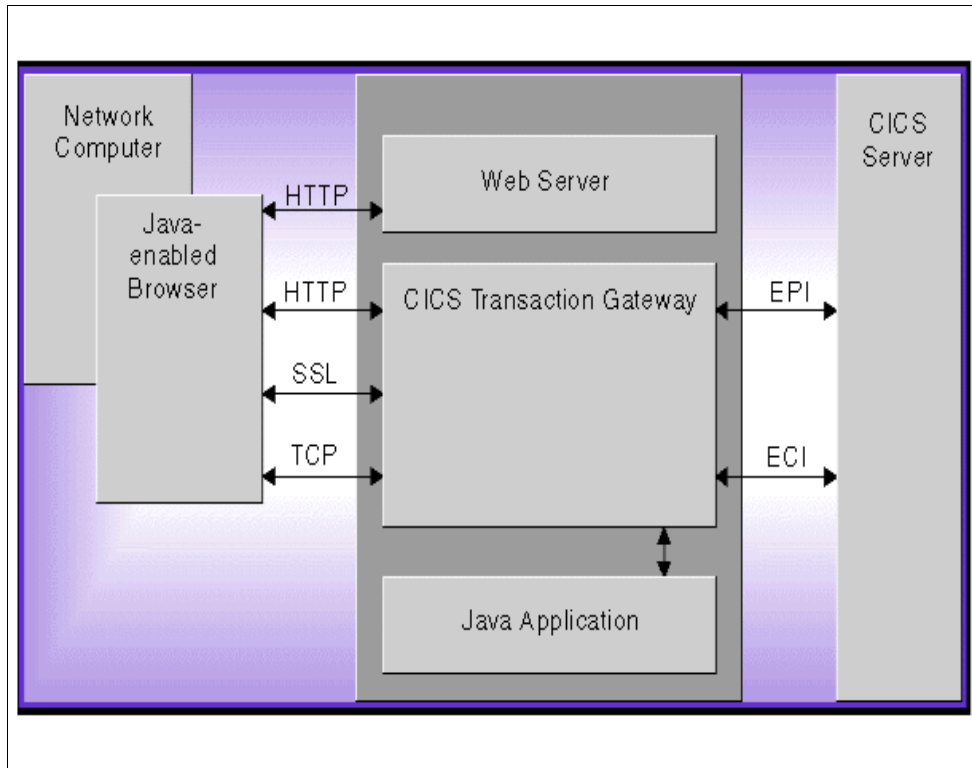
Integration Server - CICS Transaction Gateway

The CICS Transaction Gateway 3.0 is an enhanced and consolidated gateway for CICS, derived from the CICS Java Gateway 2.01 and replacing both that product and the CICS Internet Gateway.

CICS Transaction Gateway enables any Web browser, Network Computer or Internet-enabled consumer device to access business applications running on CICS servers, using one of three possible methods:

1. All Web browsers support HTTP. The CICS Transaction Gateway will render existing CICS 3270 applications into HTML automatically, and transmit to the browser using HTTP. Customers can also create their own Java servlets which present information from CICS applications in HTML forms, customised as required.
2. Java-enabled Web browsers can run Java applets. The CICS Transaction Gateway enables customer applets to access CICS 3270 applications and CICS programs using supplied Java classes and Java beans.
3. ORB-enabled Web browsers can run Java beans which interoperate with server-side Java beans (running on the CICS Transaction Gateway) via the CORBA IIOP protocol. The server-side beans can then invoke CICS 3270 applications and CICS programs using supplied Java classes.

Its main logic is implemented in Java and it executes on the Java Virtual Machine for the platform which is hosting the gateway function.



The Gateway supports two major forms of connectivity with CICS applications, using the underlying CICS Client. These are characterised by the client programming interfaces they use:

- ECI (External Call Interface) provides an RPC-like method of invoking a remote CICS program. It is equivalent to the CICS DPL (Distributed Program Link) function and enables the client to pass a variable length COMMAREA (Communications Area) to the called program
- EPI (External Presentation Interface) provides a screen scraper-like method of invoking a remote CICS 3270 transaction. It enables the client to pass a CICS TR (transaction routing) datastream to the transaction invoked. The main element of this datastream is an embedded 3270 datastream.

Both ECI and EPI invocations may be carried over a variety of network transports. The primary options are SNA LU6.2 and TCP/IP sockets, which are supported in most CICS Gateway/Client and CICS Server environments.

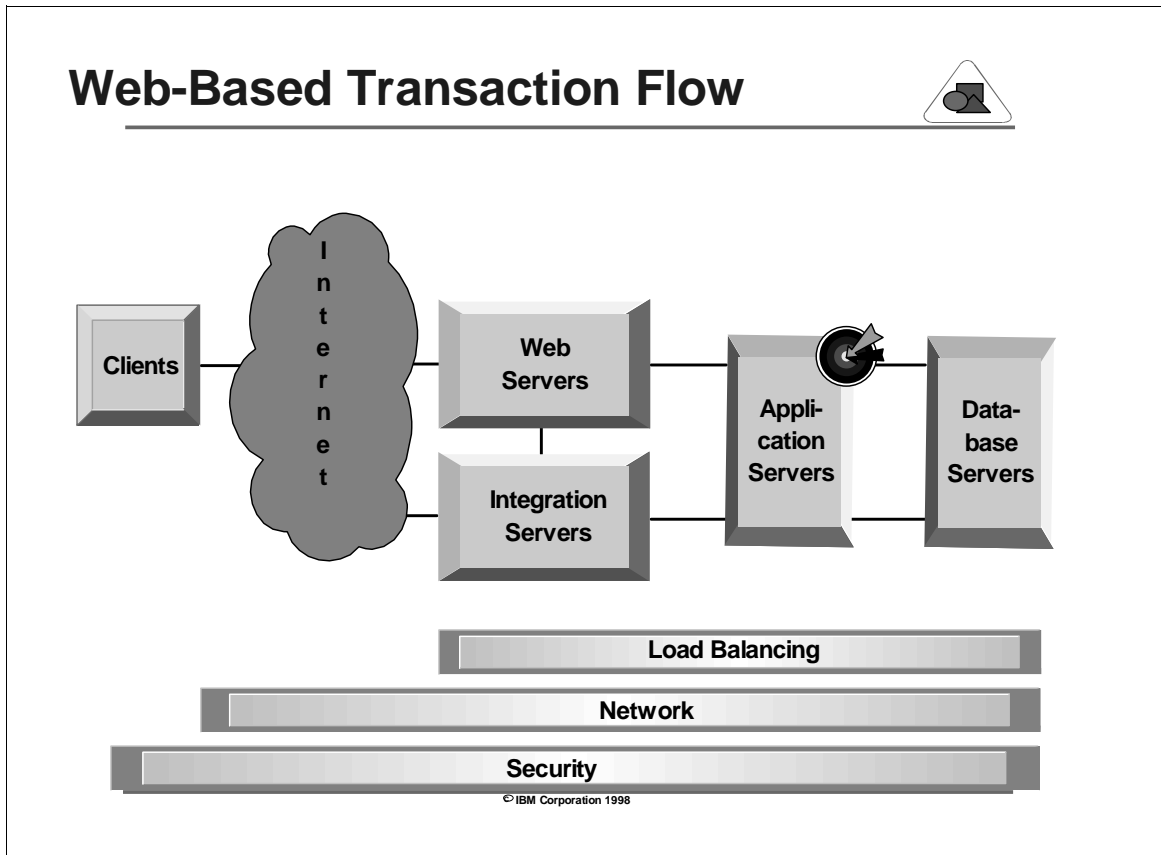
When planning an implementation of the CICS Transaction Gateway for production, it is important to do capacity planning. The major variables affecting the transaction rate achieved with a given CPU capacity are:

1. Choice of protocol for Browser-to-Gateway and Gateway-to-CICS server communications
2. Length of data transmitted (ECI COMMAREA or EPI datastream)
3. Number of concurrent users driving Gateway requests

Recommendations

1. **Use TCP/IP sockets protocol for Browser-to-Gateway communication;** avoid using HTTP
2. **Use TCP/IP sockets protocol for Gateway-to-CICS communication;** if communicating with a CICS/390 server, use TCP62 (Gateway on Windows and OS/2 platforms) or use an additional TXSeries gateway (Gateway on Unix platforms); avoid using SNA LU6.2
3. **Design applications to transmit short data messages;** ECI applications should pass business data parameters only, not presentation data; EPI applications should use the Gateway template mapping to HTML
4. **Verify scalability via a pilot implementation** before deploying applications with very large numbers of concurrent users.
5. **Use multiple gateways to limit the number of users connected to a single gateway to less than 200.**

Introduction



This section deals with the performance issues of e-business solutions developed using Java. Java has had an enormous impact on both client-side and server-side application design and deployment. With all of its advantages and strong industry commitment, Java is key to the success of IBM's e-business initiatives.

Java's performance has been greatly improved with the introduction of new compiler technologies and improved JVMs, but it is still a young technology. Attention must be paid to the architecture and the application design of Java based e-business systems.

Based on our early experience of developing multi-tier enterprise level Java applications, good system throughput with Java can be achieved. In this chapter we will discuss the following performance considerations:

1. Java Virtual Machine
2. Java Application Design
3. Java Server Application Architecture

Performance Issues and Experience

Java Virtual Machines

Bytecode Interpretation

Java is an interpreted language. Java source code must first be compiled into portable bytecodes that can then be interpreted in the Java virtual machine (JVM) on the local system. Naturally, when looking for performance improvements for your Java applications, the quality of the JVM is the place to start. For example, all JVMs implement an advanced feature called Garbage Collection (GC) to boost the programming productivity and to avoid the common pitfalls of C/C++ programming - memory leaking. However, GC can slow down the Java program execution because most GCs utilize “stop and copy” technology. Recently, a new kind of technology for GC has been developed which is called Concurrent GC. The Concurrent GC is able to perform garbage collection without stopping the program execution.

Just-In-Time Compiler

A Just-In-Time (JIT) compiler converts bytecodes into native code on-the-fly with some optimization, and it runs much faster than just a JVM. A JIT works well for computational intensive programs that execute the same segment of instructions repeatedly. But it does not yield significant improvement for programs that are I/O intensive or cause a lot of garbage collection. This is because the program code takes up such a small amount of time and thus any optimizations would become negligible. In addition, JITs are limited in the extent of optimizations they can do because their compile is a runtime cost, unlike static compilers.

Static Compiler

A static compiler compiles Java source code into native code that is executed without interpretation. The compilation is not done on-the-fly, but is instead a separate step in the development process. However, it is important to note that the portability of the Java application will not be effected because the bytecodes of the application are not altered or destroyed during the static compilation process.

Adaptive Compilers

Adaptive compilers combine the strength of JIT compilers with adaptive optimization techniques. The idea is to intelligently select, compile and optimize frequently used and/or resource intensive method. The compiled methods are then stored in a cache, ready to be executed when the methods are re-invoked. Note that as with a JIT, results of compilation are not kept between runs/users.

Selecting JVMs

High performance JVMs are critical to good performance. Here are some considerations when evaluating JVMs.

1. Java compiler and virtual machine technology changes rapidly. Today a JIT version of a compiler may provide the best performance for your solution, tomorrow it may be a static compiler, and next week there may be a new technique.
2. A JVM should be certified for portability by a recognized certification authority such as Javasoft.
3. JVMs which have been optimized for a specific operating system tend to perform better than other JVMs.
4. Systems with symmetric multiprocessors (SMP) tend to perform better because of the thread support in Java. Use JVMs which effectively support SMP systems.
5. Evaluate the tradeoff between the stability of the current release of a JVM and the performance enhancements available in the newest beta release and/or production version.

Java Performance Best Practices, Guidelines and Helpful Tools

Network Computers and Java

Network Computers (NCs) are an interesting case for performance. NCs, including the IBM Network Station, are easy to manage and provide the function required for e-business solutions such as a web browser and Java. They are often selected as the client system for e-business solutions. When Java is used for the client side of the application, special attention must be paid to the network computer memory model. Many network computers, including the Network Station, have a flat memory model (i.e. no support for virtual memory) because they have no local disk. Ensuring that your design "fits" well with the constrained NC model is quite important if you must support them. The second, and perhaps more important, reason for including NC considerations in a general discussion of performance is that many of the guidelines for using Java on an NC should apply to any well written, client-side Java code.

Garbage Collection

1. Less Garbage = Less Garbage Collection

Java has a simple memory management scheme. Memory can be dynamically allocated in a program. When that memory is no longer referenced, it is *garbage collected* and reused later. This feature eases the burden greatly for the programmers and eliminates the “memory leaks” problem commonly found in C programs.

Theoretically, the Java garbage collector is executed only when the system load is low, and it is not supposed to affect an application’s performance. However, critical components in an e-business solution may not offer such opportunities, and inevitably compete with garbage collection for system cycles. In addition to using a JVM with a high performance garbage collector, the logical way to improve performance is to reduce the workload of the garbage collector.

Here are some ways to reduce garbage collection.

- a. **Reuse existing objects.** Creating objects is more expensive than reusing them. Declare objects as static and reinitialize them as they are reused.
- b. **Create object pools.** Place inactive objects into an object pool. When a new object is needed, look for an existing object within your object pool. Similarly, when an object is no longer needed, return it to the object pool.
- c. **Avoid creating unnecessary (temporary) objects.** For example, `System.currentTimeMillis()` could be used instead of creating a new `Date` object.

2. Avoid Java Memory “Leaks”

Java does not suffer from memory “leaks” in the normal sense, but it does suffer from “memory retention”, that result in a similar result. Garbage collection only recovers memory that the JVM believes is no longer in use. There are several reasons why the garbage collector might believe an object is in use when in fact it is not. These include:

- a. Another object within the Java app with a significantly longer life span references the object. The following situations might cause this:
 - i) A class saves an object reference for one time use and does not null it afterwards. For example, the class might create an image object to display a splash screen at startup.

ii) A class uses static variables to hold an object reference. When the program terminates the static reference persists until the class is unloaded.

To prevent these memory "leaks", `null` references to objects once they are no longer needed. This should be done for each object reference. But, for even a moderately complex application, this can be an enormous task, especially if the class has already been written. It requires not only that a `null` statement be coded, but that it is placed in the right section of code. In these situations identify those portions of the class with significantly different life spans and examine the logic to ensure that stale object references are nulled.

b. A class/thread registers itself with a server class/thread but does not de-register itself on termination. Whether this is your own or a third party's server class, de-register your class/instance when you no longer need the server's services.

c. A JVM object is not released using the appropriate release mechanisms. Failure to do so can turn these "temporary" JVM objects in to "permanent" JVM objects. A prime example is failure to flush all images retrieved via the AWT Toolkit's `getImage` method. These images can consume substantial amounts of memory and will remain permanently allocated until a flush is performed on the image object. When the underlying JVM logic does not define automatic resource recovery, proper Java coding is required for releasing JVM and indeed any third-party objects. Simply nulling all your references to these objects may not be sufficient.

d. A local variable in a long-running method can unexpectedly hold a stale object reference. A common example is illustrated in the two code samples below that implement the `run` method for a thread. Because this can be difficult to understand at first glance, the first example presents the correct method.

Note the apparently superfluous `item = null` statement just before calling the `waitUntilItemAvailable()` method. Without the `item = null` statement, `item` would still be set to its previous value. Normally this would be irrelevant because the next statement would quickly execute replacing the old value of `item` with a new value. But, in this case, the `waitUntilItemAvailable()` method waits for a work item to be posted to its queue before returning and work items are only posted rarely. Consequently hours might pass before `waitUntilItemAvailable()` returns and during this whole period the object referenced by `item` remains referenced and ineligible for garbage collection.

```
// Does not hold object
public void run()
{
    while (true)
    {
        Launchable item;
        item = null;
    }
}
```

```

        // ensure null while waiting for
// waitUntilItemAvailable to return
        item = waitUntilItemAvailable();
        item.runItem();
    }
}

// Results in holding object
public void run()
{
    while (true)
    {
// waitUntilItemAvailable waits until object is posted
// for work by another thread
        Launchable item = waitUntilItemAvailable();
        // the local variable item references
        // last object returned by
        // waitUntilItemAvailable
        item.runItem();
    }
}

```

e. Garbage collection implementation varies from JVM to JVM. In many JVMs, the garbage collector, due to its inexact algorithm, sometimes erroneously believes an object is in use when it is not. This can result in a memory “leak”. The scope of such a memory leak can be reduced by always nulling references to objects once they are no longer needed. Since a reference by one in-use object will cause the referenced object to be marked as in use, eliminating the reference will allow the JVM to garbage collect the object even if its referencer is erroneously marked in use. Of course, as mentioned earlier, nulling all references will potentially increase code size.

In summary:

- As a general rule, anytime your code interacts with any outside code, including the JVM, third party code, etc., take care to follow the provided mechanisms for disengaging yourself from that contact.
- null unused references as soon as practical.

3. Avoid using finalizers when possible

Whenever an object finalizer needs to be executed, it gets placed into a finalization queue. On most systems, this queue will be run only when a ‘first level’ garbage collection **fails**. First level garbage collection only fails when room for a new object cannot be found in the heap or the overall free heap space drops below 25%. This

behavior causes all the resources associated with that object (heap and possibly peer system memory) to be retained until that time, which can be many garbage collection cycles later. This time delay between queuing a finalizer and actually running the finalizer can be considerable, causing potential shortages of system resources (heap, system memory). Not using a finalizer on the object will avoid this delay.

It should also be noted that finalization can be invoked from a Java application by calling the `runFinalization()` method in `java.lang.Runtime`. Doing this will cause everything in the finalization queue to be executed immediately and avoid the time delay problem.

Overall Design Considerations

1. Recycle and/or cache objects

The cost of object creation and the overhead of garbage collection is quite high. If existing objects can be reused, then savings in both memory and runtime performance can be realized. Look for opportunities in writing code where existing objects can be reused and recycled instead of creating new ones. Cache objects in frequently used methods so that they will persist between calls, but make sure that the cached object's state is set properly before subsequent use. For example, if you cached a date object, then prior to using it again, ensure that it is set to the proper date. Some objects are not as straightforward as a date object, so use care.

2. Variables

In contrast to a compiled language such as C++, application performance in Java is noticeably affected by what types of variables are accessed and how they are accessed. For example, while stack variables are directly addressable (and may even be placed in registers), instance variables typically require an extra level of indirection to be accessed.

This implies the potential value of data location shifting, changing the storage location of data based on the access patterns. For example, a data-intensive operation would benefit from first copying instance variables into stack variables, operating on the stack variables, and, finally, copying the stack variables back to the permanent instance variables.

This technique is particularly useful when a method variable is accessed repeatedly within a loop. For example, the common loop construct:

```
for (int i = 0; ++ i <= limit; )
```

can be improved by 25 percent (5 percent with a JIT compiler) by rewriting it as:

```
for (int i = limit; -- i >= 0; )
```

to reduce the number of accesses to the limit variable.

3. Avoid explicitly calling the garbage collector

Invoking garbage collection when responsiveness is expected, (e.g. when processing a mouse button event) can slow the program down at a time when the user is expecting fast processing. In most circumstances, invoking `system.gc()` explicitly will not be needed.

4. Compile Java files with the optimizer on

When compiling java programs, use the `-o` option to invoke the optimizer. Optimized code eliminates line numbers, is smaller, and could improve load time and application performance.

5. Reduce Class Hierarchy

In general, object creation gets slower as the class hierarchy grows deeper. In addition, a class hierarchy with a longer depth can cause longer load time of the applet because additional classes must be transferred across the network. You should avoid, when possible, specializing for minor variations that could otherwise be represented by a state variable. However, this must be done with care as it might sacrifice the object-oriented design of the application

6. Avoid synchronizing classes - limit synchronization when possible

The JVM uses class locks while resolving class references. Class references are resolved on an as used basis (versus resolving at class load time) so it is not easy to predict when the JVM will want a class lock. So, do not synchronize on a class.

An alternate approach would be to create a special purpose lock object instead. For example:

```
private static final Object lockObject = new Object();
```

However, synchronization activity can consume significant system resources and affect Java performance. This may be true even if there is only one thread being executed, due to constant checking of lock availability. As a result, synchronization should be reduced as much as possible.

7. Use Lazy Evaluation

Any large application or general purpose framework is likely to have a relatively significant amount of static data around behind the scenes. One way to initialize such data is to use static class initialization blocks. This mechanism is simple and supported by the Java language. However, it can increase an application's load time, since all such initialization is done before the application starts, even if it is not needed until hours later (or perhaps never needed during a particular run of the application.)

Lazy evaluation is a reliable and much used mechanism for deferring initialization of static data until it is actually needed. So any data never used is never created, and any data not actually needed to bootstrap a subsystem does not place a startup time burden on its client applications.

In object oriented languages, lazy evaluation is quite easy to implement. You merely make your data private and provide a public getter method (something you would want to do anyway.) Since any access to the data is through the getter, it can allocate the object upon demand. This does imply the need for some synchronization, i.e. you must make sure that two threads don't simultaneously try to create the static object. This is easily done through a synchronized code block that does the actual object creation.

The minimum synchronization overhead can be obtained by first checking the static reference for null and only entering the synchronized block if it is still null. A subsequent check is still required after entering the block to insure that another thread has not already created the object. But, once the object actually gets created, the only performance overhead imposed is that of a null reference check on each subsequent entry to the getter.

To do the synchronization, the class' class member is used since it's a static object that is already available to us. This avoids a bootstrapping issue of having to synchronize in order to create an object in order to synchronize the creation of an object, etc...

Here is an example of a class which does lazy evaluation of a data member:

```
class Foo
{
    // A getter method that lazily creates the static Bar member
    public static Bar getBar()
    {
        if (theBar == null)
        {
            synchronized(Foo.class)
            {
                if (theBar == null)
                {
                    //
                }
            }
        }
    }
}
```

```

        // Allocate a Bar. Assume this constructor does a
        // significant amount of setup.
        //
        theBar = new Bar();
    }
}
return theBar;
}
private static Bar theBar;
}

```

Another aspect of lazy evaluation involves complex, usually tabbed, GUI objects. Where possible, only immediately create the components that are initially visible to the user then create the others either in the background or upon their coming into view. This strategy will increase the perceived performance of the application by displaying the relevant information as soon as possible, and will often save on memory overhead if the user never even accesses the other components.

For example, if a complex dialog has ten tabbed panels, each of which has an average of fifteen objects on them, building and placing these one hundred and fifty objects makes no sense when the user might only interact with the initially displayed panel or a small number of them.

8. Optimized Classes

Well written Java applications can suffer performance degradation when inappropriate classes are used. For instance, standard Java class libraries rely on general purpose interfaces to hide underlying implementation details. While modular, generic implementations may limit overall library performance.

An example of this is the interaction between the `ByteArrayOutputStream` and `DataOutputStream` classes. `ByteArrayOutputStream` encapsulates the writing of bytes into a byte array. `DataOutputStream` serializes high level Java built-in types (`int`, `long`, `float`, `String`, etc.) to an object exposing the `OutputStream` interface. However, as it is used to support a variety of output media including files, TCP/IP connections, and memory buffers, the `OutputStream` interface only allows one byte of data to be output at a time (via a synchronized method). When the `DataOutputStream` is attached to `ByteArrayOutputStream`, performance is sub-optimal as data is only produced one byte at a time (through the `OutputStream` interface). To output each byte, the `DataOutputStream` incurs the overhead of calling a synchronized method, checking for array overflow, copying the output byte into the byte array, and incrementing the output byte counter.

To achieve optimal performance, an optimized class could be implemented to merge the functionality of the `ByteArrayOutputStream` and `DataOutputStream` classes. To place data into a byte array, the optimized class simply copies the data into the buffer, checking the array bounds and incrementing

the byte counter only *once* for each data item output. For applications that do not require synchronized access to the stream, the methods in the optimized class can be asynchronousized to further boost performance.

9. Use `StringBuffer` when doing excessive string manipulations.

Since strings are immutable, any change to a string will create at least one more string object. This degrades performance and unnecessarily creates objects that will eventually need to be garbage collected. `StringBuffers`, however, are modifiable and can be used to avoid creating temporary `String` objects.

10. Exception handling

The JVM provides native instructions to support exception handling, and its execution is relatively fast. According to one study, a *try-catch* clause imposes little or no overhead if the exception is not thrown. The cost of handling a thrown exception becomes significant only if an exception is signaled.

To take the advantage of this, implement exception handling instead of explicit error checking when exceptions are expected to be rarely thrown. For example, the following code segment

```
if ((i >= 0) && (i < array.length))
    x = array[ i ];
else
    // error
```

could be rewritten as

```
try {
    x = array[ i ]
}
catch (ArrayOutOfBoundsException e) {
    // error
}
```

11. Explicitly close files

When using the `FileInputStream` method, do not rely on the object finalizer to close the file for you. Explicitly close the file when you are done. The problem with allowing the finalizer to close the file is there is a potentially long delay before the finalizers are actually run. This will keep the operating system file handle in use until the time the finalizers do run potentially creating a situation where the pool of operating system file handles become exhausted. Explicitly closing the file will cause the file handle to be released immediately.

12. Optimize use of the Java Heap

a. Images use system memory. As a rule of thumb, when an image is being rendered by the graphic subsystem, it will require about 2 times the image size in system memory. So, if you are rendering an image that is roughly 1 megabyte, it will require about 2 megabytes of system memory to contain. This can quickly deplete available system memory. It is also important to ensure that images and graphic objects are released properly after they are used. Failing to do so can cause sizable “leaks” in the system memory area.

b. Large objects (over 1 megabyte in size) may encounter allocation problems within the Java heap. Due to fragmentation that occurs within the heap, the overall free heap space is not generally available as a contiguous block of memory. Since the JVM requires that all objects be allocated within a contiguous heap area, allocation of large objects in a fragmented heap may be problematic. As a rule of thumb, about 20% of the available heap space can be obtained for a single large object allocation. So, if an application had 5 megabytes of free heap space available, it could expect to be able to allocate a 1 megabyte object. Whenever reasonable, splitting a large object into two (or more) smaller objects is a potential solution to this problem.

c. The Java heap must be large enough to contain all of the Java application’s “working set of objects” plus a buffer. A reasonable size for this heap buffer is to have at least 30-40% free heap space after a garbage collection.

On the other hand, creating a heap that is too large for the application is not good either. First, this will reduce the amount of system memory available to the application and secondly the garbage collection costs for managing the larger heap will increase. Viewing the `-verbosegc` messages while the application is running will lend insight into determining the correct heap size for a given application.

13. Limit number of threads

Every java thread created requires dedicated memory for its’ native stack frame. On many systems, the size of this native stack frame is controlled by the `-ss` Java command line parameter and is the same for every Java thread created.⁸ The default stack size on some platforms is as much as 32 kilobytes. For an application that has 20 threads this represents 32KB*20 or 640KB. Limiting the number of threads in the application will help reduce the system memory requirements. It may also improve performance by giving more CPU time to threads doing real work.

14. Avoid Zombie Objects

When a `java.lang.Thread` object is created (by calling `Thread.run`) but never

⁸ The JVM for the AS/400 does not support the `-ss` parameter.

started (no call to its `start()` method), a reference to it will remain in the `ThreadGroup` in which it was created. Calling the `run()` method directly will run the function locally in the caller's `Thread`. A new thread is not actually started. This `Thread` object will never be collected and it becomes a zombie `Thread` holding on to its resources. The only way to remove such a `Thread` is to call its `stop()` method followed by calling its `start()` method. In general, the creation of a `Thread` should be delayed until just before it is started.

15. Packaging

Java 1.1 introduced the Java archive (JAR file): a packaging and performance enhancing mechanism. The JAR format allows a Java applet and its requisite components (all the associated class files, images, sounds, and so forth) to be downloaded by a browser in a single HTTP transaction, rather than by a new connection as each is needed. This greatly improves the speed with which an applet can be loaded onto a Web page and begin executing.

The JAR format also supports compression, which reduces the size of the file and improves download time still further. This mechanism should be used judiciously, however. Recall the 80 percent /20 percent rule: 80 percent of a piece of code's execution time is expended in 20% of the code. In a typical application, there are large chunks of code that do not get touched in any given execution. Downloading this code is wasteful and time-consuming.

Instead of packaging all the code for an applet into a single JAR, the download time can be optimized by creating a JAR file for the most frequently used 20 percent only. The other 80 percent of the code can be split into second-tier packages (if other appropriate groupings can be identified).

JAR files are specified as part of the `<APPLET>` tag embedded in the HTML file, as this example shows:

```
<APPLET ARCHIVE = "TH.jar, TH1.jar" CODE = "Thing.class"
WIDTH = 350 HEIGHT = 100>
<PARAM NAME = "theSound" VALUE = "playme.au">
</APPLET>
```

Note the way multiple archive files are specified in a comma-separated list. If archives are used, all needed resources must be packaged in archives. This is the case in practice, but it is different from the documentation, which says that if a resource cannot be found in an archive, it is retrieved "in the normal fashion." It is important to bear in mind that JAR files are completely loaded into memory when referenced. This can cause problems in low-memory situations.

16. Class Preloading

One technique that was evaluated was that of preloading classes before they were

needed. This is used to improve the apparent interactive performance of an applet or application. Without preloading, whenever the user calls up a new type of window or makes use of some functionality for the first time, the user can experience a long delay as the JVM loads classes (perhaps across the network).

To ensure that the class preloading doesn't adversely affect the execution of the main application, it is common to perform the loading in a separate thread. One easy way of initiating the loading is to use the `java.lang.Class.forName()` method, which will cause a class to be loaded by the JVM but which will not instantiate an object or call any method of the class.

```
public class PreLoad implements Runnable
{
    String [] classes;
    public PreLoad (String [] classes)
    {
        this.classes = classes;
    }
    public void run ()
    {
        try
        {
            for (int i = 0; i < classes.length; i ++)
                Class.forName (classes [i]); // perform the preload
        }
        catch (ClassNotFoundException cnfe)
        {
            // do something...
        }
    }
}
```

To use this facility requires code like:

```
void doThePreLoad1()
{
    final String [] list = {
        "java.awt.Button",
        "java.util.Vector",
        // more...
        "org.moon.LMC.Thing",
    };
    new Thread (new PreLoad (list)).start ();
}
```

The major drawback of this technique is the necessity to pepper calls to the `doThePreLoad1()`, `doThePreLoad2()`..., `doThePreLoadX()` methods through an application's code. Determining precisely when to initiate a preload is an

art.

17. Set the `java.net.Socket.setTcpNoDelay` method to `(true)`

If your solution uses Java sockets, evaluate / experiment with disabling the `TCP_NODELAY` option of TCP/IP by setting the `java.net.Socket.setTcpNoDelay` method to `(true)`. This method will disable Nagle's algorithm, also known as Nagleling, (described in RFC 896), which conserves bandwidth by minimizing the number of segments that are sent. When disabled, data will be sent earlier, at the cost of an increase in bandwidth consumption. Experiments within IBM have reported 10X to 50X increase in performance by disabling the method.

18. JDBC Data Access

When using the `ResultSet.getXXX` methods in JDBC make sure that the get call you use is the of the same type as the way the data is stored in the database. So, if there are integers in the database, please use the `ResultSet.getInt()` method to read it from the result set.

19. Using JNI

a. When using JNI and you need to get a member of a primitive array object (from C to JAVA) then use the `Get<PrimitiveType>ArrayRegion` call instead of the `Get<PrimitiveType>Array` calls.

b. Group native operations to reduce the number of JNI calls.

Consider consolidating transactions or operations to minimize the number of JNI calls needed to accomplish a task. Reducing the number of times the JNI overhead needs to be paid will improve performance.

20. Data Types

a. Primitive types are faster than classes encapsulating types. Avoid the costs of object creation and manipulation by using primitive types for variables when prudent. Memory can be reduced and variable access times can be improved.

In the following example, the second declaration is smaller and quicker:

```
Currency {  
    public double amount;  
}  
  
double currency_amount;
```

b. Unlike C++, casting in Java is not done at compile time. Since there is a cost at run-time, avoid unnecessary recasting of variables.

c. Use `int` instead of `long` when possible on 32-bit systems.

`long` is 64-bit while `int` is 32-bit data type. 32-bit operations are executed faster than 64-bit on 32-bit systems. Example 1 took about half of the time of Example 2. It is worthwhile noting that Example 2 will run faster on systems with 64-bit addressing such as the AS/400.

Example 1:

```
int i;
    long q;
    {
        int j = 0;
        for ( i = 0; i<250000;i++)
            {
                j = j + 1;
            }
    }
```

Example 2:

```
int i;
long q;
{
    int j = 0;
    for ( q = 0; q<250000;q++)
        {
            j = j + 1;
        }
}
```

d. Use `static final` when creating constants.

When data is invariant, declare it as `static` and `final`. By reducing the number of times variables need to be initialized and giving better optimization information to the JVM, performance can be improved. In this example, the first array will need to be created and initialized each time the object `test` is instantiated, but the second array will only be initialized once.

```
class test {

int myarray[] = { 1,2,3,4,5,6,7,8,9,10,
                  2,3,4,5,6,7,8,9,10,11,
                  3,4,5,6,7,8,9,10,11,12,
                  4,5,6,7,8,9,10,11,12,13,
                  5,6,7,8,9,10,11,12,13,14 };
```

```

static final int myarray2[] = { 1,2,3,4,5,6,7,8,9,10,
                               2,3,4,5,6,7,8,9,10,11,
                               3,4,5,6,7,8,9,10,11,12,
                               4,5,6,7,8,9,10,11,12,13,
                               5,6,7,8,9,10,11,12,13,14 };

.
.
.
}

```

21. Avoid excessive writing to the Java console.

Writing information to the java console takes time and resources, and should not be done unless necessary. Although helpful in debugging, writing to the console usually involves a great deal of string manipulations, text formatting, and output. These are typically slow operations. Even when the console is not displayed, performance can be adversely affected.

22. When possible, declare methods as `final`.

Declare methods as `final` whenever possible. Final methods can be handled better by the JVM, leading to improved performance. In this example, the second method will execute faster than the first one.

```

void doThing()
{
    for (int i=0; i<100; ++i)
        dosomething;
}

final void doThingfinal()
{
    for (int i=0; i<100; ++i)
        dosomething;
}

```

23. Reorder the order in `CLASSPATH`

When the JVM is looking for a class, it searches the directories in the order they are given in `CLASSPATH`. You can improve the startup times of your Java applications by reordering the paths in the `CLASSPATH` environment variable. Place the most used libraries earlier in the `CLASSPATH`.

24. Arrays are faster than Vectors

By avoiding vectors when arrays will suffice, application performance can be improved. If you use a vector, remember that it is faster to add or delete items from the end of the vector.

25. Always use flush with `getImage`

If an image is retrieved via `getToolkit.getImage` (the `getImage` method is in the `AbstractWindowsToolkit` class and should always be present), it is important to always use a flush to ensure that the images are discarded. It is not sufficient to set the image reference to `null`. There is an internal hashtable which is maintained with the object reference that will only be released when a flush is issued. Failure to do this will result in the memory associated with the image never being reclaimed.

26. Do not overuse class variables.

Performance can be improved by using local variables. The code in example 1 will execute faster than the code in Example 2.

Example1:

```
public void loop() {
    int j = 0;
    for ( int i = 0; i<250000;i++)
    {
        j = j + 1;
    }
}
```

Example 2:

```
int i;
public void loop() {
    int j = 0;
    for (i = 0; i<250000;i++)
    {
        j = j + 1;
    }
}
```

27. Know when to use immutable objects.

There are many valid reasons to use immutable objects. "The main disadvantage of immutable objects can be performance, so it is important to know when and where to use them.

The

Java API provides two very similar classes, one immutable and one mutable. `String` was designed to be immutable, and thus simpler and safer. `StringBuffer` was designed to be mutable, with tremendous advantages in performance. For example, look at the following code, where one uses string concatenation and the other uses the `StringBuffer` append method. (The `doSomething()` method is overloaded to take characters as either `String` or `StringBuffer`.)

Case 1

```
for (i = 0; i < source.length; ++i) {
    doSomething(i + ": " + source[i]);
}
```

Case 2

```
StringBuffer temp = new StringBuffer();
for (i = 0; i < source.length; ++i) {
    temp.setLength(0); // clear previous contents
    temp.append(i).append(": ").append(source[i]);
    doSomething(temp);
}
```

Behind your back, the compiler will optimize Case 1, but it still involves the creation of two objects per iteration. Case 2, on the other hand, avoids any object creation. Because of this, Case 2 can be over 1000% faster than Case 1 (depending on your JIT, of course).

28. Consider performance when selecting third party components

Selection of high quality third party components can be a make or break decision for some applications. Since third party components cannot often be modified by the client, if that code lies within a critical performance path of the application and performs badly, no amount of optimization elsewhere in the application will likely make up for it. We have seen some applications whose performance has been potentially fatally flawed by use of poorly performing components, and these were components with high market visibility and vendor name recognition.

29. Run "before" and "after" benchmarks

The only way to determine if a technique is going to improve the performance is to measure it. There are many theories on what should make a program faster. However, only measurements will tell you for sure.

30. Use strength reduction

Strength reduction means that you use a less computing-intensive technique than a functionally equivalent operation. The most common example of strength reduction is using the shift operator to multiply and divide integers by a power of 2. For example, `x >> 2` can be used in place of `x / 4`, and `x << 1` replaces `x * 2`.

Tools

1. Profiling

Profiling is a valuable technique for estimating the relative importance of blocks of code (typically methods) in a class. If the java interpreter is invoked with the `-prof` switch, it will create a file with the name `java.prof` in the invocation directory. At execution's end, this file can be examined. This information can provide information to help when deciding which methods should be further optimized.

Several profilers exist, including:

IBM's own Jinsight tool -

<http://www.alphaWorks.ibm.com/formula/jinsight>

2. JVMPI

Java has evolved very quickly. One of areas that is currently getting a lot of attention is performance; both improvements to the JVM itself as well as tooling to help developers tune their code. The JVM Profiler Interface (JVMPI)⁹ is one example. IBM has included the API in the IBM WIN32 JDK 1.1.7. It "is a two-way function call interface between the Java virtual machine and an in-process profiler agent. On one hand, the virtual machine notifies the profiler agent of various events, corresponding to, for example, heap allocation, thread start, etc. On the other hand, the profiler agent issues controls and requests for more information through the JVMPI. For example, the profiler agent can turn on/off a specific event notification, based on the needs of the profiler front-end."¹⁰

3. Reduce the Size of Java Class File Archives

Java programs are routinely transmitted over low-bandwidth network connections as compressed class file archives (i.e., zip files and jar files). The size of an archive has a direct impact on the time required to download a program, and therefore on browser response time for web pages that contain Java applets. Archive size also

⁹ see

<http://java.sun.com/products/jdk/1.2/docs/guide/jvmpi/jvmpi.html>
for details

¹⁰ *ibid.*

affects the initialization time required by Java virtual machines (“class loading”).

The Jikes Application eXtractor (JAX) was developed by IBM to show how a number of compiler optimization and program transformation techniques can be used to reduce the size of class file archives. JAX has been evaluated using a set of realistic benchmarks ranging from 12 to 2,050 classes (the corresponding archives range from 13,312 to 5,710,539 bytes). A reduction in archive size between 30.1% and 84.0% was measured.

JAX is available for download and use at:

<http://www.alphaWorks.ibm.com/formula/JAX>

4. TOAD

Use TOAD to Analyze Non-distributed programs as well as RMI Distributed ones. The IBM TOAD tool includes Static Analysis which allows you to analyze a Java program without running it, by using JAN. JAN, Toad's Java Analysis tool, provides an effective technique for building the call graph of a given Java application. This view may be used for program understanding, as well as for optimization purposes; such as code size reduction and devirtualization. JAN can analyze non-distributed applications, as well as RMI-based distributed ones.

TOAD is available for download and use at:

<http://www.alphaWorks.ibm.com/formula/TOAD>

Sources of Information

This list of Best Practices, Guidelines and Helpful Tools above was compiled from several sources across IBM, including:

- “Java Thin-Client Programming for a Network Computing Environment” SG24-5115-00
- “Java Performance Tuning Tips 1.0” at <http://www.software.ibm.com/os/warp/performance/javatip.htm#Table>
- IBM Alphaworks <http://www.alphaWorks.ibm.com/>
- Java programming Guidelines from The eNetwork On-Demand Server development team at RTP
- and **many** tips from the Austin NCSD External Performance team

Architect Java for Performance - 2-Tier Versus 3-Tier e-business System

A key aspect of multi-tier architectural design is how and where the client/server application is split into functional (logical) units that can be assigned to the (physical) client and one or more (physical) servers. The typical functional units are the user interface, the business or application logic and the shared data. Though 2-tier and 3-tier are the two general terms commonly used to describe multi-tier architectures, there are many variations of multi-tier architectures, depending on how the application is split and on the technology selected communication between tiers. Note that the application partitioning discussed here is “logical” partitioning. Thus, one could implement a “logical” 3-tier architecture on a “physical” 2-tier system if desired.

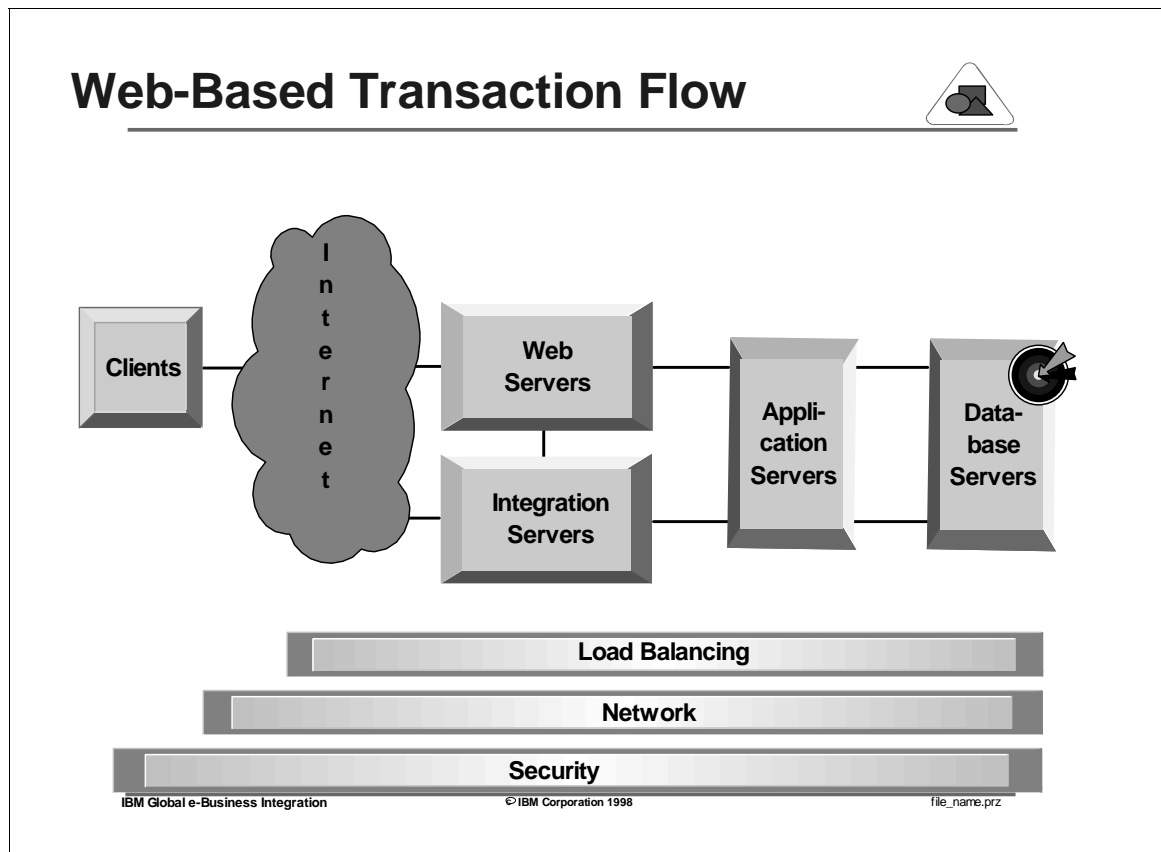
In a 2-tier client/server system, the application logic is implemented inside the user interface on the client and/or within the database on the server. In a 3-tier Application Framework for e-business system, the application logic generally resides in the logical middle tier, which is separated from the user interface and from the shared data. In general, this approach results in solutions that exhibit better scalability, flexibility and reliability. We recommend that Java-based solutions conform to the 3-tier logical model because of these advantages.

Server Application Architectures

A server object, running inside of one JVM, can serve requests from one client or from multiple clients. Server objects of the former type are called “unshared” objects, while the ones of the later type are called “shared” objects. Typically, multiple server objects are running simultaneously to support large number of clients. In the case of unshared objects, a separate server JVM will be started up for each client, with each request run as a separate thread. While sometimes it is necessary to implement unshared server objects, shared server objects are more robust and scaleable.

Database

Introduction



Existing business applications require the ability to create, replace, update and delete data which is stored persistently in a databases and file systems. Most e-business solutions have the similar data access requirements. Many businesses are working through the architectural and technology issues involved in extending their existing applications and data stores to the Internet. This *is* e-business. The stateless paradigm of the web browser changes the environment in which the enabled client interacts with data sources across the network. In the client/server paradigm, the client has direct control the of the commit point, even to a remote data source. The Application Framework for e-business moves this function to the server side, introducing the need for these tasks to be handled properly by software working in behalf of the client.

And, there are performance and throughput considerations relating to the use of databases in this environment. The following topics deserve special attention and will be covered in detail:

1. Connection Management
2. Locking
3. SQL

Performance Issues and Tradeoffs

Connection Management

Every request to a database requires the use of a database connection. A web application server makes data requests on behalf of the web client by either directly accessing the data system or by utilizing a data access gateway to perform that service.

In either case, building database connections dynamically to serve single requests can add significant time to a web 'transaction'. In a high throughput environment where many concurrent requests are being serviced, the resources spent in building and tearing down database connections can be detrimental to performance and scalability. A much less resource intensive way to handle these requests is to create a pool of pre-established database connections.

For example, in a recent performance test, response times for simple transactions were reduced from .27 seconds to .09 seconds simply by making use of pre-existing database connections, rather than closing and allocating new ones.

Associated with a database connection is the concept of a user and user authority on one or more data objects. System resources spent in ascertaining a user's data rights can significantly add to response time and resource utilization. Servicing data requests in an un-trusted Internet environment typically requires much more extensive authentication checking than servicing data requests in a secure, intranet environment where clients are "trusted".

Time-outs are another issue for database servicing web requests, particularly where TCP/IP requests are made. The TCP/IP default time-out period is two hours. This means that a database connection can remain idle up to two hours before being notified that a network failure has occurred. If the database connection is holding locks on behalf of an application, other applications contending for the same resources will be negatively impacted. In addition, database servers typically allocate a limited number of connections to be used by remote applications. Unnecessarily idle connections represent wasted resources which could be used by other applications.

Locking

Locking mechanisms are used by database managers to handle resource allocation among many users without compromising data integrity. While necessary and valuable, locking tends to impact the ability to handle multiple requests for the same data resource concurrently. Until the introduction of distributed databases in the 1980's many database vendors lived with their customer's complaints that page level locking was too restrictive. Many customers wanted the enhanced data availability that could be had by locking at the row instead of the page level. But with the addition of network latencies that accompanied distributed access came a heightened need for finer granularity in data and index locking mechanisms. Today, most large database vendors support row level locking.

The way in which the web application server maintains state can add to lock contention issues for the database. Because in web applications every request is a separate entity, applications which need to maintain information between requests must decide the method in which they will preserve state. Assuming a web application performs remote updates, as in the case of electronic commerce, which options are available for maintaining state, while minimizing lock contention?

The first, and least exclusive alternative is optimistic locking. With optimistic locking, the ability to apply updates is verified incrementally as the business logic progresses. Updates are not actually attempted until all business conditions are met. The side effect of this approach is that the original data values must be revalidated before updates are actually applied. For example, if the customer is browsing an online catalog and places an item into his shopping cart, then at checkout time he or she may find that the desired item is out of stock.

A second option is to logically flag data fields as updated in order to reserve them until the end of the unit of work, at which time they will be permanently changed. While in this case the shopper is able to buy exactly what they want, at checkout time the 'flags' must be removed.

Finally, updates can be incrementally applied as the business logic progresses. As in the previous case, the shopper is assured of the items they put into their basket. However, if the shopper changes their mind, this approach can restrict other users from acquiring a desired resource while requiring application programmers to include compensating transaction logic into their code.

Each option introduces performance benefits, and introduces business and application tradeoffs. Pick the alternative that best fits your application update needs.

SQL

The efficiency of the SQL access strategy to a relational database is the most important factor in determining the elapsed time of SQL execution. Of course, the access path selected is as much a function of database design as SQL coding, but in general, given a good database design, 95% of a SQL statement response time is a function of the selected access path.

If we assume that the SQL has been coded in the most efficient manner, what can we do to reduce resource usage and elapsed time? First, like any SQL based application, object checking, authorization checking and access path selection must be performed before SQL can be executed. In static SQL, these tasks are done only once at bind time. With dynamic SQL, all three tasks are performed at first request, and depending on the database management system and server configuration, might be required for each request. The key then, to reducing response time is to limit the number of pre-execution tasks which must be accomplished, and to limit the number of SQL requests necessary to satisfy the business request.

In one example, the time needed to build a catalog page was unacceptable to the customer. Our analysis showed that no single SQL interaction seemed to be a problem. Most SQL interactions took 200 milliseconds or less, a few up to 500 milliseconds. The unacceptable response time, as it turned out, was the sheer number of SQL requests required to build a single page. In this example a network analyzer showed that 140 to 200 trips were made between the web server to the database server to build the requested page. This by itself, was a problem for the single web user, but it also degraded the performance of the entire web site. At peak load times, the number of network requests to and from the database created bottlenecks in the web server's network stack. One answer to problems of this sort is to reduce the number of object requests per page. SQL object requests can be minimized through a combination of caching, stored procedures and web page simplification.

On S/390, stored procedures are particularly efficient. Not only do they use static SQL, but S/390 Workload Manager and dispatcher have enhancements especially designed to make DB2 Stored Procedures perform well. Another S/390 hint is to be sure to turn on the Prepared Statement Cache if you do use dynamic SQL to avoid the overhead of repeated Prepares of the same SQL statements. This requires `MAXKEEPD` to be specified in `DB2 ZPARMS`, and for the plan or package to be bound with `KEEPDYNAMIC(YES)`.

Recommendations

1. **Make use of pre-allocated pools of database manager connections.**
2. **Minimize database authorization checking where possible.**
3. **Select an appropriate timeout value for remote database connections.**
4. **Use optimistic locking or logical locking for distributed applications where possible.**
5. **Make use of static SQL for best performance.** SQLJ is a good alternative to JDBC for example.

Appendix A - Dallas GEI

This section will describe the role of the Dallas Global e-business Integration Center (GEI)¹¹ and will also describe the process used by the GEI; **Design for Performance**.

GEI is an IBM competency center, specializing in network computing and complex client/server solutions in a multivendor environment. This expertise focuses on the design and testing of e-business solutions. The GEI has over 50 technical experts in core competency areas such as: application development tools (i.e. presentation), middleware, database, web technologies, networking, and systems management.

The Design for Performance Process consists of five major steps:

1. Design Review
2. Proof of Technology
3. Testing for Indicators of Performance
4. Performance Reviews
5. Performance Assessment Test

Where you begin in the process and how many steps you complete depends on where you are in the life-cycle of your project and on your application's performance requirements. A low volume, departmental decision support system may not need the rigor of this process at all. On the other hand, an e-business system with one or more of the following characteristics is a strong candidate for all five steps of the process:

- High volume
- Critical response time
- Transactional
- New technologies

Design with Performance Process

Background

While each application solution has its own unique performance issues, there are also general "patterns" which emerge over time. The GEI first noticed these patterns in our work with client / server based applications and later with web based and e-business solutions. We found that many customers would pilot fully developed applications only to find out that their middleware and/or hardware servers just did not have the capacity to handle the workload required by the business. Either the platform was too small, or the software package was not scaleable. Unfortunately, this often meant the customer had to go "back to the drawing board" to re-engineer the application solution. These

¹¹ see <http://www.osc.ibm.com/>

folks really needed a “holistic” approach to performance testing incorporating hardware, software and network components in a multi-vendor environment. This is the primary reason that the GEI created an end-to-end performance testing specialty that offered Performance Assessment Tests (PAT).

The PAT process provides response time and server sizing information, and identifies bottlenecks in the hardware and software tested. This satisfied the needs of customers who were in the systems-integration, or pilot stages of application development and roll out.

However, as the GEI became more experienced in performance testing, another need became clear. Traditional end-to-end performance tests are too late in the life-cycle of a project that is venturing for the first time into new application paradigms, tools and development methods. In today’s development arena, development tools and middle ware are evolving at a frenetic pace. The traditional application development cycle has been reduced to a “web year” (typically six months). In addition to the sizing and service level agreement needs fulfilled by the more traditional PATs, a new kind of test is needed. We call this offering Test for Indicators of Performance (TIP).

A TIP is a “light weight” performance test, early in the development cycle of an application. A “risky” application, one with many unknowns in terms of tools and technologies is an ideal candidate for a TIP in the application prototype stage. A TIP gives you an “*indicator of performance and scalability*”. In other words, a PAT assumes a stable, complete application for which you develop customer scenarios, workload mixes and complete database populations in order to simulate the performance and throughput you will achieve on roll out. A TIP does not assume anything other than bare application functionality and test data. The purpose of the TIP is to give you an early warning indicator that the application architecture, development paradigm, or tool set will not achieve the stated business objectives. This can be especially critical when components of the architecture or application are on the “bleeding edge” of technology.

For example - in 1997, the GEI tested some of the Nagano Olympics applications. During that testing we quickly found “bottlenecks” in the chosen CGI implementation due to the throughput and response time requirements of the application set tested. One application was rewritten by IBM Japan to incorporate native API processing, another was able to use FastCGIs written by Hawthorne Research instead. Both applications used pooled database connections - one used specially designed memory mapped caches. The point is that application refinements were made relatively early on in the development process so that the roll out date (in this case unchangeable) could be met while achieving business goals for the applications.

TIP versus PAT

	Application Life cycle	Functionality Required	Test Goals	Environment for test	Tools
TIP	Prototype	Minimal	“In the Ballpark” Response Time Scaleability Potential	No special machinery Can use development environment	Ad-Hoc or Regression Test Tools or Shareware
PAT	Systems-Integration	Representative Production Workload	Specific Response Time with Specific Throughput	Very Structured - Isolated for Repeatable Results	Repeatable Test Tools with Granularity in Workload capability and Analysis and Monitoring

Now that we’ve covered the TIP and PAT steps of the Design for Performance process, we will discuss the other steps of the process as well as how the steps relate to one another.

How the process applies to your project and which steps you may choose to use depends on where you are currently in the development of your application architecture and how you rate the “business risks” associated with your project. What follows is a listing of the steps, and a brief description of how they might be used in your project.

Design Review

A Design Review is a discussion-oriented session that is customized to meet a particular customer’s requirements. The length of session, technical skill mix, and agenda vary according to the application and customer needs. The purpose of the review is to analyze an application architecture, evaluate the feasibility of the proposed design, identify potential problems (e.g., systems, network, data placement, application design) and document recommendations and alternative solutions.

The output of the architecture review typically includes more than one potential architectural alternative. Each documented alternative will have included with it the “plusses” and “minuses” of the alternative as they relate to the business environment, problem and objectives. For example, one component of an architecture is security.

Each of the documented architectural alternatives will be discussed in terms of it's strengths and weaknesses as they relate to the different solution components - in this case security.

Proof of Technology

A Proof of Technology (POT) validates the functional capabilities of key components in a proposed e-business solution, including logical and physical architecture, connectivity, data access, platforms and tools. This is a quick and effective way to get experience with the elements of your proposed solution, identify potential problems and find alternative architectural solutions.

Testing for Indicators of Performance (TIP)

The TIP has already been discussed in detail in this section. One or more TIP scenarios will be developed from the POT. Since the POT may have included more than one architectural alternative, the TIP can then also be used to compare and contrast those alternatives for performance and scale ability.

Performance Review

Once the architecture has been selected and development begun, periodic checkpoints should be taken during application development. These checkpoints should be focused on making sure that the application once developed, will meet performance and resource consumption goals. These checkpoints should be taken as a cross functional team. As an example, the database designer should work closely with the application developers to make sure that the SQL for the application and the Database Design work in harmony to accomplish business goals.

Performance Assessment Test (PAT)

Finally, when the solution is ready for system-integration testing and/or pilot, a PAT can be run. As previously discussed, this test will help to ensure adequate network and hardware capacity for peak periods. In addition Service Levels can be established and any last bottlenecks in the system - like database locking contention can be addressed.

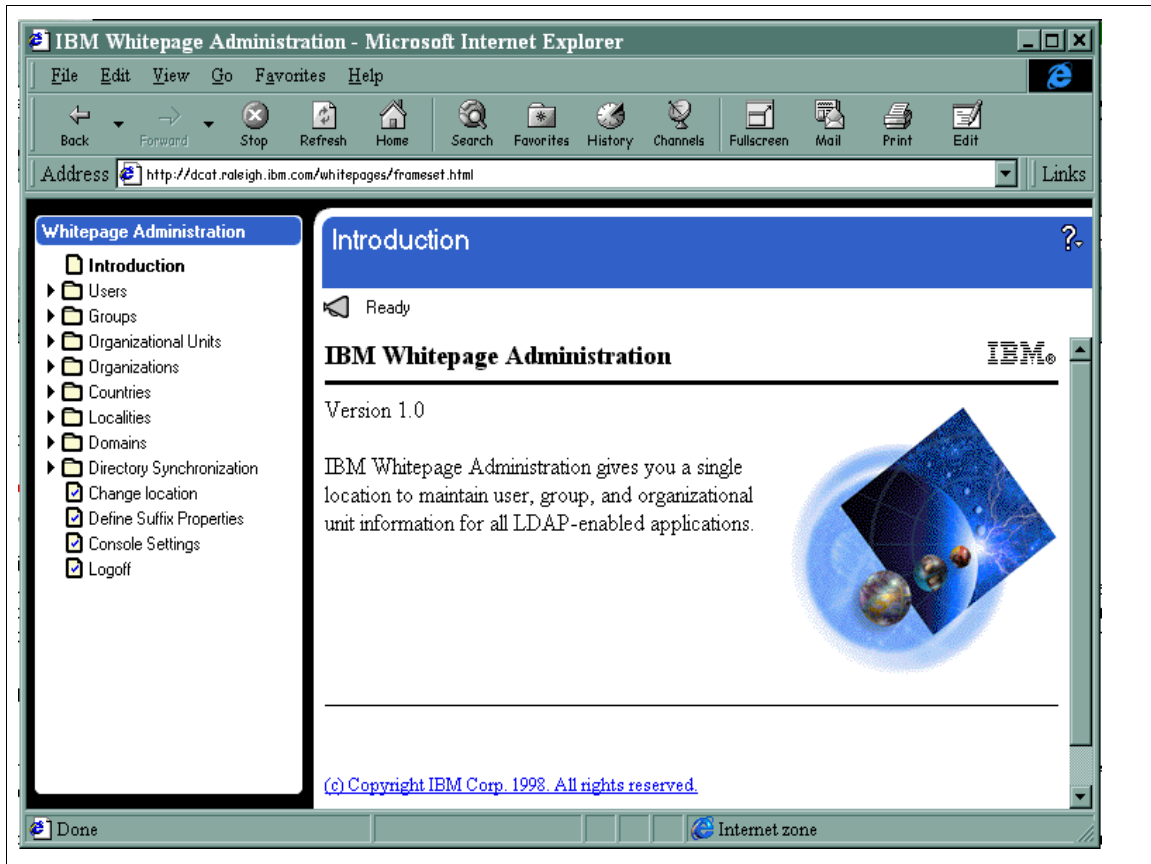
Appendix B - Concept2 Client Framework

Concept2 was developed by IBM to provide a standard browser-based client platform capable of administering multiple IBM software products in a consistent way. Like many advanced designs, Concept2 experienced some concerns with performance. This section describes the technical issues and the “tricks” used to attain good performance.

Overview

Concept2 is a framework for developing user interfaces that are hosted in a browser. This framework is intended to support very rapid prototyping and easy refinement into a complete web-based lightweight client application. The main features of this framework are encapsulated navigation, stable title areas, canonical representation of the hierarchy of interfaces offered by an application, and very broad accommodation of HTML hosted implementation technologies for included interfaces.

The standard parts of a Concept2 application are supplied by a set of applets which are not modified by application implementors. These applets are hosted in HTML also supplied as part of the Concept2 framework. The non-standard parts of a Concept2 application are accessed by interacting with the standard parts. The non-standard parts consist of work area content (mostly task interfaces), help interfaces, and launched interfaces contained in separate browser or java frames.



One of the presumptions behind the Concept2 design is that some users, such as middleware administrators, will probably prefer an interface that supports a paradigm of completing work, rather than exploring information. Thus an emphasis was placed on emulating the "serious" parts of desktop application behavior. On the other hand, since remote access to function is less reliable than local access, it was obvious that there would be some differences between the behavior of Web applications and Desktop applications. Considerable effort was expended in keeping the differences in behavior small and comprehensible.

In order to look more like a desktop application, Concept2 was designed for fast, consistent interaction between its visual components. The goal was to emulate the speed of a desktop applications response to trivial interactions. The goal was accomplished by producing standard Java components that interact through inter-applet Communications in the browser.

This trick is notoriously hard to stabilize, and has eluded people on a lot of projects. It appears from our experience that the core of the trick is a completely counterintuitive degree of exactness in standardizing the strings that define the origin of the applet, including key fields in the `<applet...>...</applet>` tags. These strings are used in slightly different ways by different browsers under different circumstances, and even

a slight departure from the restrictions we apply seems to result in unreliability once the application is put into a production-like environment.

For applets to be placed reliably in the same Applet Class Loader, (and thus be able to communicate via static method calls) they must agree on all of the following elements.

1. They must all be coded in HTML that was accessed using URLs that have exactly the same protocol, host address, and port number. Two applet instances loaded in different instances of the same browser, cannot communicate if one is loaded as `http:\\freeradical.raleigh.ibm.com\demo\xtalk.html` and the other is loaded as `http:\\freeradical\demo\xtalk.html`. They can only communicate reliably in the entire set of IE 4+ and NN 4+ browsers if the address part of the URL matches exactly, no aliases, no coding of port numbers on one & not the other, etc. In Concept2 we smooth over this problem by loading all of the content from the machine using relative URLs based on the URL that started the session.

2. They must all have the `codebase=` parameter specified in the applet tag, with an absolute virtual path that matches exactly. Two applet instances loaded in different instances of the same browser, cannot communicate reliably if one specifies `codebase=../classes` and the other specifies `codebase=classes`, even if these two URL paths result in getting to the same place. The only safe way to communicate seems to be to use a codebase parameter that starts with a `/`, which makes the resulting Virtual Path absolute instead of relative to the virtual path to the HTML hosting the applet. Without this, someone may prototype in HTML files, all from the same virtual directory, and then use a CGI or Servlet in the final solution and having mysterious communication failures in some browsers. It does imply that the HTML is generated to know the exact virtual path or alias required to get to the classes.

3. They must all have the `archive=` parameter for their jar file specified in a `<param...>` tag, between the `<applet...>` tag and the `</applet>` tag. The value of the parameter in each must be an absolute virtual path (starts with a `/`) that matches exactly. Two applet instances, loaded in different instances of the same browser, cannot communicate reliably if one specifies `archive=../classes/my.jar` and the other specifies `archive=classes/my.jar`, even if these two URL paths result in getting to the same place. The safe way to communicate seems to be to use an archive parameter that starts with a `/`, which makes the resulting Virtual Directory Path absolute instead of relative to the virtual path to the HTML hosting the applet. It does imply that the HTML is generated to know the exact virtual path or alias required to get to the classes.

Managing this trick reliably allowed the client side components in the browser to synchronize their state at a speed that was not dependent on Network access times, making them appear as tightly integrated as desktop application components.

In order to minimize the confusion of inconsistent performance for nontrivial operations, Concept2 used inter-applet event flows to model a "Loading..." state for an interface. A Loading message that identifies the next task interface is shown in one of the stable applets, and a new HTML document or object is loaded. When the HTML arrives, an applet defined in it called a HeraldApplet communicates with the rest of the framework to report that the interface is ready. Upon receiving this event, the Loading message is changed to Ready, or a more informative message carried by the HeraldApplet tags. Thus, when response turned slow, a familiar and reassuring level of feedback on the application state was available. This encouraged users to wait the required amount of time for longer operations.

Concept2 Performance Concerns

The main long-running performance concern in Concept2 was start up time. This led to continuous efforts to minimize the download time for the jar file and distribute the creation of objects over time. They also led to efforts to mask loading time with new user interface features, such as a start up page with an animated gif, and a sign-on page that appears while the framework is still loading.

During some periods we also had concerns about the speed of processing repaint events and the visual effect this speed had on the visible changes in components as a mouse "flew" over, or scrolling occurred.

Concept2 Application Performance Observations

We have observed some performance anomalies, and have done some work to circumvent them.

- IE 4.0-4.01 - Constructing URLs that contain different host addresses and comparing them causes uncatchable Security related exceptions. These exceptions seem to cause a disproportionate slowdown of IE after they occur. Creating the URLs using a 9.37.80.245 syntax rather than a `freeradical.raleigh.ibm.com` syntax avoids the exceptions, but has ramifications for Web Server configuration.
- Strange delays when vanilla CGI programs generating HTML are hosted in Netscape Fasttrack server and are accessed by IE 4.0. One possibility is that FastTrack leaves off some end-of-document trailer that causes IE to leave the read waiting until it times out. We decided not to support Netscape Fasttrack Server.

Concept2 Bundle Issues

One of the main features of Concept2 applications is the ease of their integration into a bundled application. While this is a good feature for applications to have, it can have an unintended impact on performance by increasing the size of the resultant application. These very large applications may then have unacceptable start up times.

A bundle interface includes all of the content from one or more stand-alone Concept2 applications. (Each Concept2 application is pre-wired for this sort of inclusion in a bundle). For example IBM NT Suites R2 includes such a bundle interface, including the content of the Concept2 applications for administrating several of the products in the suite.

For this reason the elements that synchronize the integrated applications are constructed after the rest of the framework has been exposed to the user. These are roughly proportional in size and download time to the number of integrated applications, which seems like a good balance.

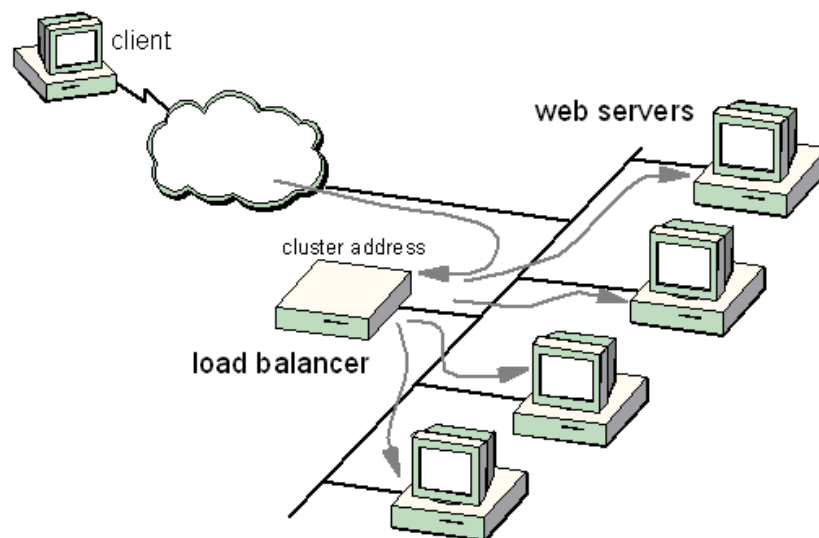
Appendix C - Availability and scalability of web applications using session data and clustering¹²

In the web world, a clustered environment is a set of web servers that appear to browsers on the Internet as a single server. A clustered environment requires a **load balancer**, the software or hardware responsible for making the set of web servers appear to be a single server. For example, the Network Dispatcher component of IBM WebSphere Performance Pack provides this function.

Why is clustering desirable? **The primary reason sites use clustering is to increase availability.**

Planning for availability involves reviewing your configuration to identify and eliminate “single points of failure”. A single point of failure is any single entity in your configuration which, if it fails, will cause the entire configuration to be unavailable. For a web site, if there is only one web server, then the server is a single point of failure. If the web server unexpectedly fails, then the whole site is unavailable. Furthermore, if the web server must be taken off-line for maintenance, for example, to upgrade or add software or hardware, then the whole site is off-line.

Clustering technology employs multiple web servers. Clustering software or hardware detects when a web server becomes unavailable, and routes requests only to the available web servers. In this way, if one web server fails, the web site remains available. Note that the load balancer must also have some availability mechanism, such as the hot stand-by capability of Network Dispatcher, to avoid having it become a single point of failure.



¹² Susan Hanis, IBM RTP, February 2, 1999

Besides increasing availability, clustering improves **performance**. If you need to increase your web site capacity and you have only one web server, then you may exchange your web server machine for something faster. Each time you do, the web site will be unavailable during the time required to swap the hardware. Over time, if you have done this repeatedly, you will have a pile of unused hardware.

In contrast, clustering technology lets you increase your web site capacity by adding more web servers. The site stays up while you install the new hardware and you can continue to use all your old hardware after each upgrade. The load-balancing software or hardware that makes your web servers appear to be a single web server balances the requests over the available web servers, directing new requests to the least busy available server.

Another reason to use clustering, closely related to performance, is **scalability**. A web site is scalable if

- as you add resources, the capacity of the site increases proportionately; and,
- as the number of concurrent users increases, the capacity of the site does not decline; and,
- as you add to the volume of web content, the capacity of the site does not decline.

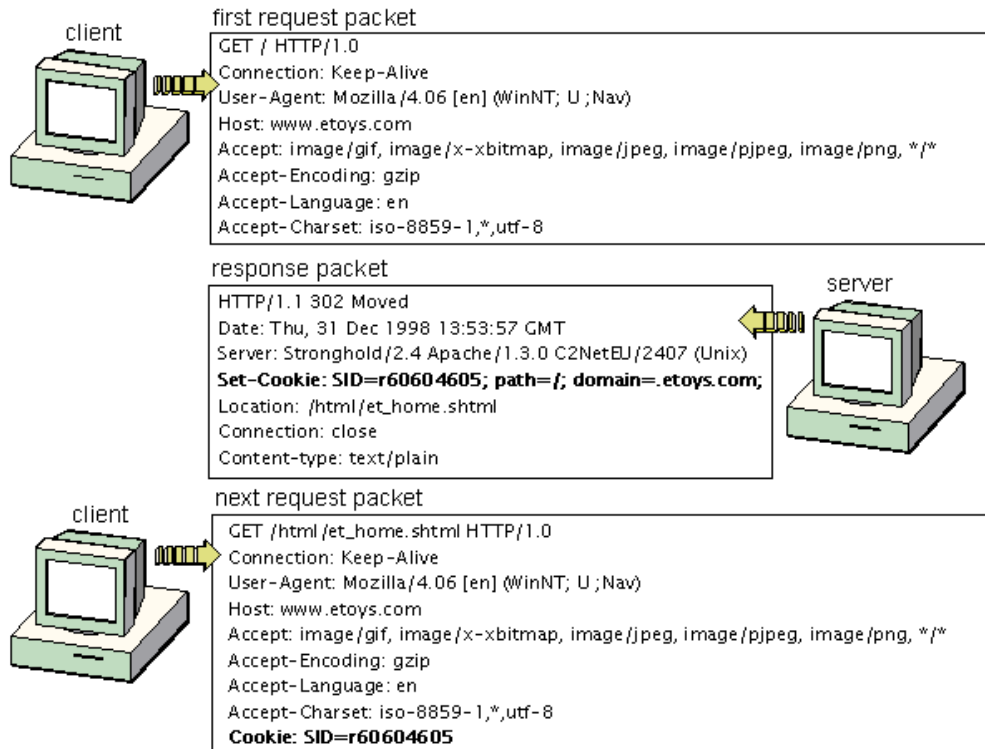
A cluster of web servers is generally more scalable than a single web server, because you can add resources to a single web server without necessarily increasing the capacity of the site. In contrast, if you add a web server to a cluster, it almost always increases the site capacity.

Session data

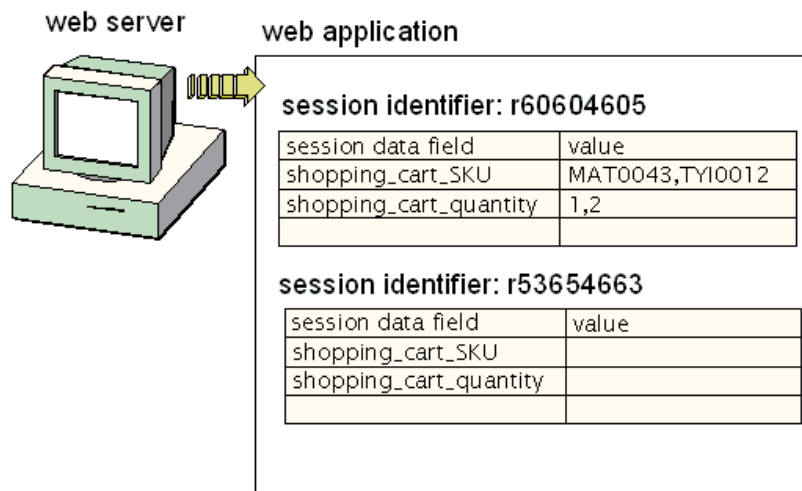
Now let's define web application session data. What is it, and why is it desirable?

In the simplest case, a web server responds to requests from a client without regard to any previous requests from that same client. That works fairly well if the web server is primarily providing **information**; however, it doesn't work so well if the web server is processing **transactions**. Processing transactions often requires multiple steps, and the required input from the client can't be contained within a single request, usually because some of the input depends on the output from a previous step.

Therefore most transaction-oriented web application servers have a mechanism to define **sessions**. A session is a set of interactions between a web server and a browser, where information about previous interactions is carried forward to future interactions. The information that is carried forward is the **session data**, and the web server might assign each session a **session identifier**. Usually, the session data is kept on the web server and the session identifier is passed to the client, which then includes this session identifier with subsequent requests to the same web site.



An example of session data is a virtual shopping cart. Suppose you are at a web site that offers merchandise for sale, and you enter a search request. You select an item to “add to your cart”, then enter another search request and select another item to “add to your cart”. After these two interactions, you expect to have two items in your virtual shopping cart. Thanks to session data, the first item you added to your shopping cart did not disappear when you added the second item.



Managing session data in a cluster

Now let's look at what happens when we take a web application that depends on session data and deploy it in a cluster. As you might expect, the availability, performance, and scalability characteristics are affected by the design of the web application. In particular, the success of the application depends on how it handles session data and how it interacts with the load balancer.

In the following sections, we will examine several possible implementations, indicating how well the application should fare in a clustered environment.

We will examine three techniques that result in low cluster synergy:

- Sticky ports
- Site partitioning
- Session data is kept in the request

Then we will examine three techniques that result in medium cluster synergy:

- Session data is propagated to all servers
- Session data is propagated on demand
- Session data is kept in a central data store

Finally we will examine a technique that results in high cluster synergy:

- Session data is kept in a distributed data store

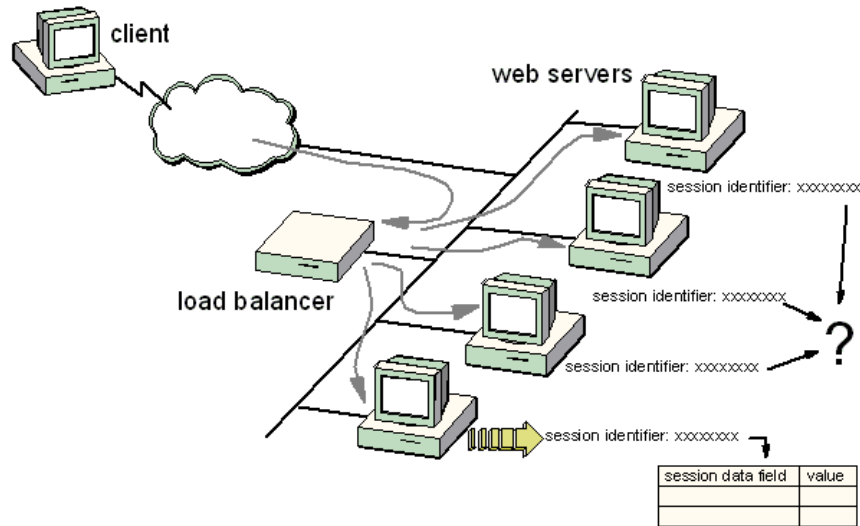
Low cluster synergy (no direct cluster support)

Web applications in this category have usually been written without any thought that they might some day be run in a clustered environment.

Often this type of web application keeps the session data in memory on the web server. Or it may keep the session data on a remote session server, but only the originating web server knows the location of that session server. In the latter case, only the session identifier is kept on the web server.

For most applications in this class, subsequent requests from the same client must be sent to the same web server, or the session data is lost. This can create unpredictable behavior. For example, the contents of a shopping cart will disappear when a request is routed to another web server. The shopping cart contents might reappear if a later request is sent to the first server. Multiple versions of the shopping cart can exist for the same user.

A client that needs to be sent back to the same server is said to have **affinity** for that server, and the application is said to require **server affinity**.



Sticky ports

For applications with low cluster synergy, a simple, standard approach is to configure the cluster's port as "sticky". For Network Dispatcher, stickiness means that subsequent connections from the same client are sent to the same server, if that server is still available. (The client is identified by its IP address.)

Sticky ports generally allow web applications to function properly. However, there are some disadvantages:

- Sticky ports may defeat load-balancing algorithms, because the decision about where to send subsequent requests is based on which server the previous connection used, rather than which server is least busy. In the worst case, all the active sessions on a web site are handled by a single server, regardless of the number of servers actually available, simply because all the sessions started on that server.
- Sticky ports are overkill. All requests to the cluster generate affinity to a particular server, even if no session data is associated with the requests. That is, even if there is nothing in the shopping cart, a user is repeatedly routed back to the same server, even if it is busier than other servers.
- Stickiness can also defeat site availability. If a server that contains session information dies, then session information is lost for all users currently using that server. This is a characteristic of applications that require server affinity. So it is not so much a disadvantage of sticky ports as it is an application design limit that can't

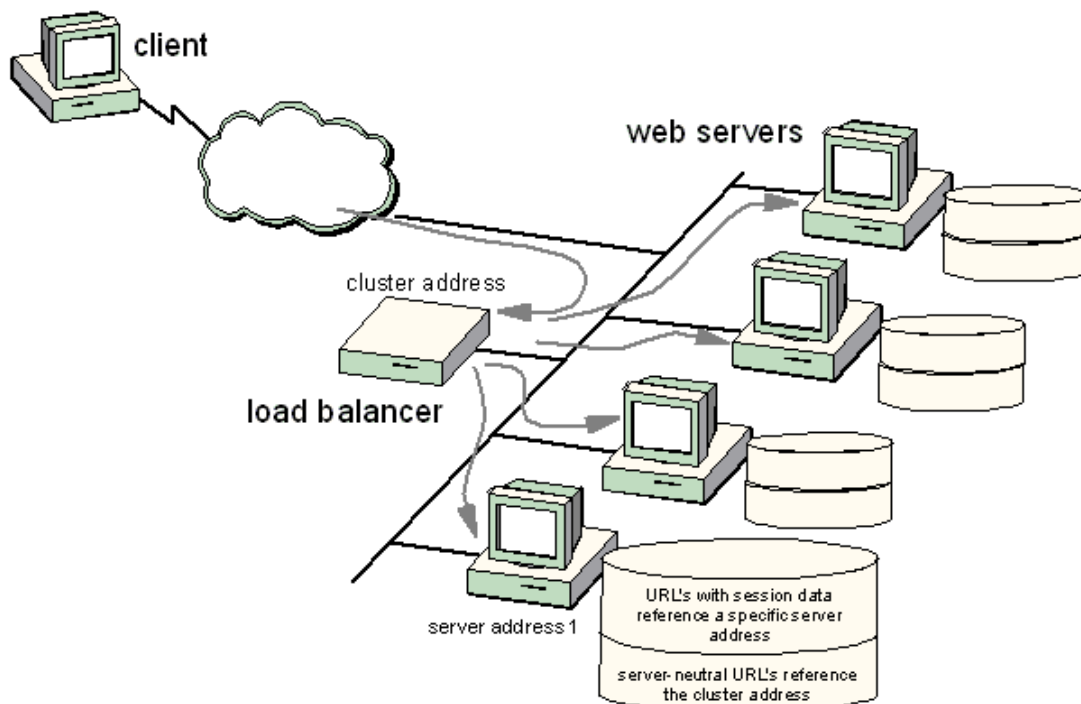
be overcome with sticky ports.

- Stickiness is based on the client IP address. That is, it assumes a 1-to-1 correlation between a person and an IP address. Quite often on the Internet, the “client” IP address of a web request is not the actual user’s machine, but is the IP address of a proxy or socks server. With stickiness, all requests from the same proxy server will be directed to the same web server, whether they come from the same end user or not. Even if the proxy server passes the client identity in the request headers, this information is not detected, and therefore is not used, by most load balancers through the sticky port process.

Site partitioning

This approach tries to address two problems caused by using sticky ports: First, that they are overkill, creating server affinity for every request, whether or not session data is being used; and second, that this overkill causes load-balancing algorithms to be defeated.

With this approach, the site map is partitioned into a section that doesn’t require server affinity (server neutral) and a section that does require affinity. The goal is to keep the section that requires affinity small. The URLs for the server neutral content use the site cluster address and are load balanced. These URLs include a “session start” page and all static objects (graphics, boiler plate text, and so on). In contrast, the URLs that launch applications with server affinity use the specific server address of the server that the “session start” request was routed to.

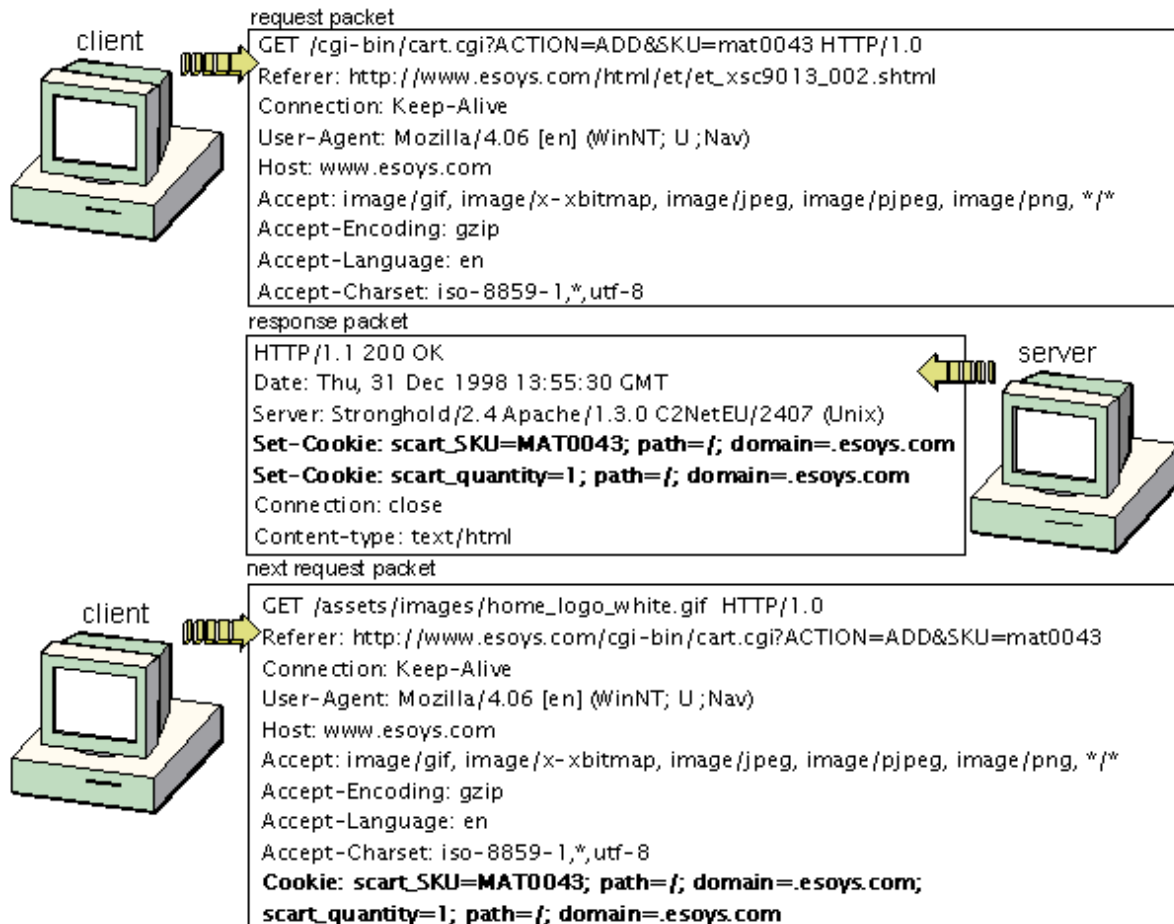


Site partitioning requires web authors to specify absolute, rather than relative, URLs. Unfortunately, using absolute URLs can cause another problem. Clients who bookmark a page on a site will use that URL with a specific web server reference, which means that when they return to the site at a later date, they will go directly to the specific web server, even if they haven't established a session. Therefore the applications need to have error pages to respond to specific site URLs with missing or expired session information by sending them back to the "session start" page.

Separating the URLs that require server affinity from those that don't reduces the amount of work impacted by the negative effects of sticky ports, even if both sets of URLs are clustered. Sticky ports are not required for the server-neutral files that represent the large majority of the bytes delivered. Splitting the site into two partitions also increases the probability that client connections to the partition with affinity will be of shorter duration and won't overlap across different clients.

Session data is kept in the request

With this approach, instead of keeping the session data on the web server and sending a session identifier to the client, the application sends the session data itself to the client to be included with subsequent requests. Usually this is accomplished by means of cookies, but might also be done by rewriting the site URLs to include the session data.



Storing session data in the request allows any web server in the cluster to handle any request from any client, because all of the relevant session data is available on every request.

Unfortunately, the web server must parse (or re-parse) the session data on every request, which can add significantly to the amount of time and processor resources required to handle the request. Also, there are practical limits on the size of URLs and cookies, as well as limits on the number of cookies that a browser will accept from a web server. Both scalability and performance can suffer with this approach.

However, the biggest problem with this approach is security. Because all of the session data is included on each request, it essentially becomes public information, available to anyone “sniffing” the Internet. The privacy of the clients’ transactions is seriously compromised.

Load balancing

Applications in the low cluster synergy class do not have a specific feedback mechanism in place to enhance the load-balancing decision. The load balancer must rely on its standard feedback mechanisms to determine if a web server is unavailable or overloaded.

The standard HTTP advisor for Network Dispatcher is used to detect whether the web server software on each server is responding to requests. The advisor sends a trivial HTTP request to each server at regular intervals. The HTTP request is "HEAD / ". This is similar to the request, "GET /", but it tells the server to respond with only HTTP headers, not the actual file contents. If the advisor receives HTTP header "200 OK", then it knows that the web server can successfully respond to requests for the home page. ("/" implies the home page for the site.)

But it's possible that the web server software is healthy and able to respond to requests for static pages and at the same time a key web application on that server is failing. For example, the web server can return a search page to the client, but the search application does not have connectivity to the database or has experienced some other failure. Because the web server continues to respond successfully to the advisor, Network Dispatcher considers it available and continues to route requests there. That's undesirable because the search application is really not available on that server. What's worse is that the web server responds to requests for the search application with an error page. Typically it takes much less time to return the error page than to return the results of the search application. Network Dispatcher then sees that connections are active on this web server for less time than on the other web servers. It deduces that the failing server has more capacity than the other web servers, and routes more incoming requests to the failing server than to the functioning servers.

For this reason, standard Network Dispatcher advisors are suitable when the web site primarily delivers static pages but are not sufficient when the web site is delivering an application. Custom advisors can solve this problem and will be discussed in the next section.

Medium cluster synergy

For applications with medium cluster synergy, session data will be preserved even if subsequent connections are sent to a different server. Significant cost in performance or scalability is often incurred, and such techniques might not offer a high-availability solution.

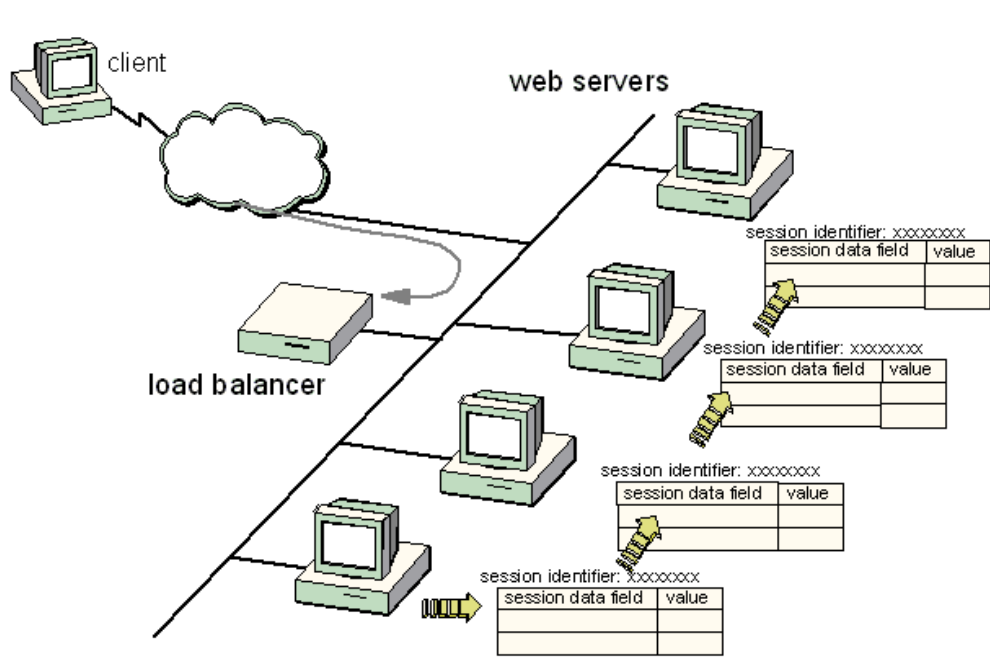
In this section we will examine three techniques that result in medium cluster synergy:

- Session data is propagated to all servers
- Session data is propagated on demand

- Session data is kept in a central data store

Session data is propagated to all servers

With this approach, every time session data is created or changed, it is copied to all of the servers in the cluster. This is great for availability, because every server always has the session data for any incoming request. It's not so great for scalability. As the number of servers grows, copying the data for a single session to all the servers requires an increasing amount of time. As the number of sessions grows, more and more of the site's resources are spent propagating the session data across all nodes.



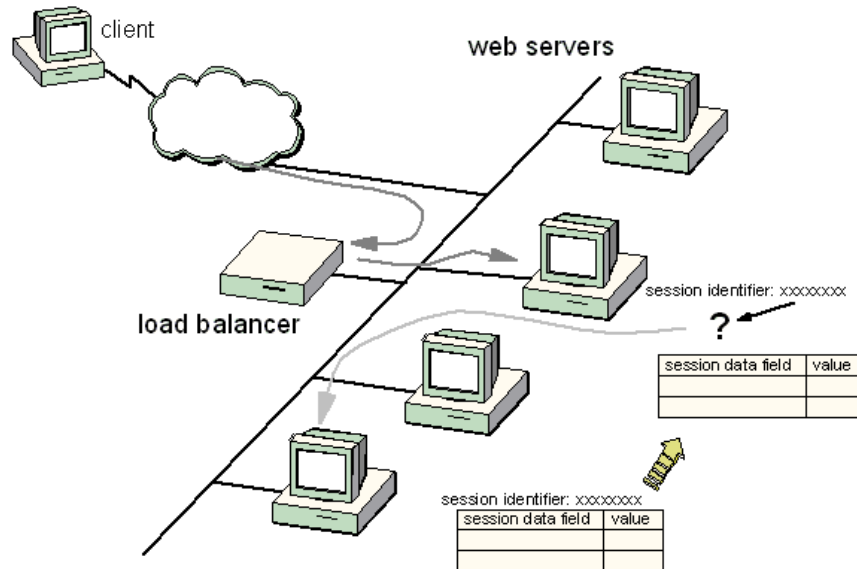
Session data is propagated on-demand

Using this technique, the web servers are aware of the other web servers in the cluster and can retrieve session data from the “owning” server. Two flavors are possible:

- The server must poll the other servers in the cluster to find out which one “owns” the session.
- The server can determine from some known data (a cookie, a URL, a client IP address, and so on) which of the other servers in the cluster “owns” the session.

The first case creates a substantial scalability problem. As the number of server nodes and the number of concurrent sessions increases, more and more time is spent contacting other nodes in search of session data.

The second case is better but still presents a scalability concern. As the number of servers and sessions increases, the odds of a request returning to the “owning” server will eventually become so low that an extra request to get the session data from the owning server is implied on nearly every client request.



The performance, availability, and scalability characteristics of applications using this approach can be substantially improved if an adequate mechanism for the non-owning web servers to cache session data is in place. That is, if the non-owning web server has already retrieved the session data once, and it can be determined that the session data is still up-to-date, then the non-owning web server does not have to request the session data from the owning web server again, improving performance. This technique can also improve availability. If the owning server fails but the session data has been cached on another web server, then that server could potentially take over and become the owning web server for that session.

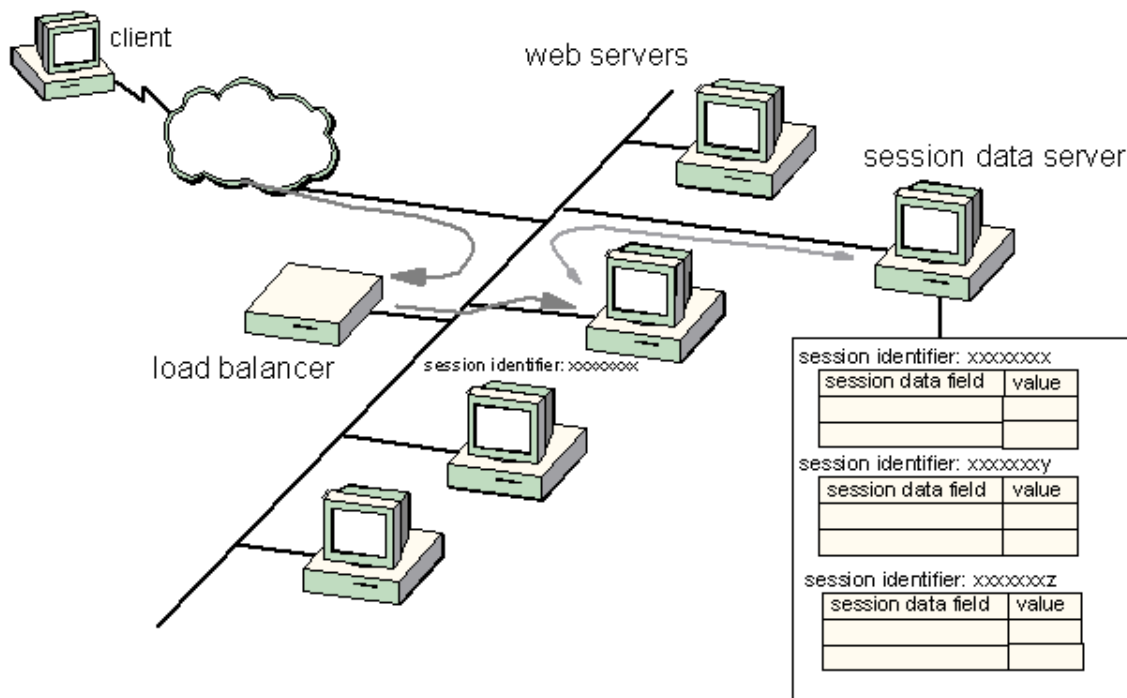
Further performance gains can be realized if the load-balancer and web servers can work together to encourage but not require affinity between a specific client and a specific web server. For example, if the web servers in the cluster are able to determine the owning web server from some known data (such as a cookie, a URL, or a client IP address), then the load balancer can also use this known data to determine the owning web server. Given this information, the load balancer might prefer to send the request to the owning web server, unless it has determined that the owning web server is currently overloaded.

To allow the web servers and load balancer to determine the owning web server from the contents of a cookie, either the web application or the load balancer must create a cookie that includes the identity of, or a mapping to, the owning web server. It is tempting to include a host name or IP address in the cookie, but this practice is discouraged because it poses security risks. If information about the owning server is included in a cookie, it should be encoded.

Encoding reduces the security risk but does not eliminate it. As an alternative strategy, instead of forcing the load balancer to examine cookies to create server affinity, a web application can notify the load balancer when a client has done something that requires server affinity, and notify it later when the session data has expired and server affinity is no longer required.

Session data is kept in a central data store

This solution allows any server to respond to requests from any client and access the session data from a central data store. Because all servers can access the central data store, it doesn't matter which server handles the request.



This solution increases the availability of the web servers but it creates a new single point of failure: If the central data store becomes unavailable, then all session data is lost.

It also can be a problem for scalability. As the numbers of sessions and servers grows, the central data store will take longer and longer to service requests for session data. Lock serialization becomes a concern. Most browsers will open multiple parallel connections to retrieve the contents of a web page faster. If requests that require session state are sent on different connections routed to different web servers, the web servers take turns locking each other out of the central data store as they access the session data.

Load balancing feedback

Applications with medium cluster synergy have mechanisms in place to enhance feedback to the load balancer. Earlier we noted that standard Network Dispatcher advisors are suitable when the web site primarily delivers static pages but are not sufficient when the web site is delivering an application. Custom advisors can solve this problem.

A custom advisor runs on the Network Dispatcher machine and provides load-balancing input the same way a standard advisor does, but it allows the user to monitor application-specific status on the load-balanced servers. For example, the custom advisor can invoke a URL on a load-balanced server. That URL can be a special web application that monitors the health or performance of the other web applications on that server. Sample code is provided for the advisor program which runs on the Network Dispatcher machine. The web application that monitors the other web applications is unique to each particular web application server environment and is not supplied by Network Dispatcher.

High cluster synergy

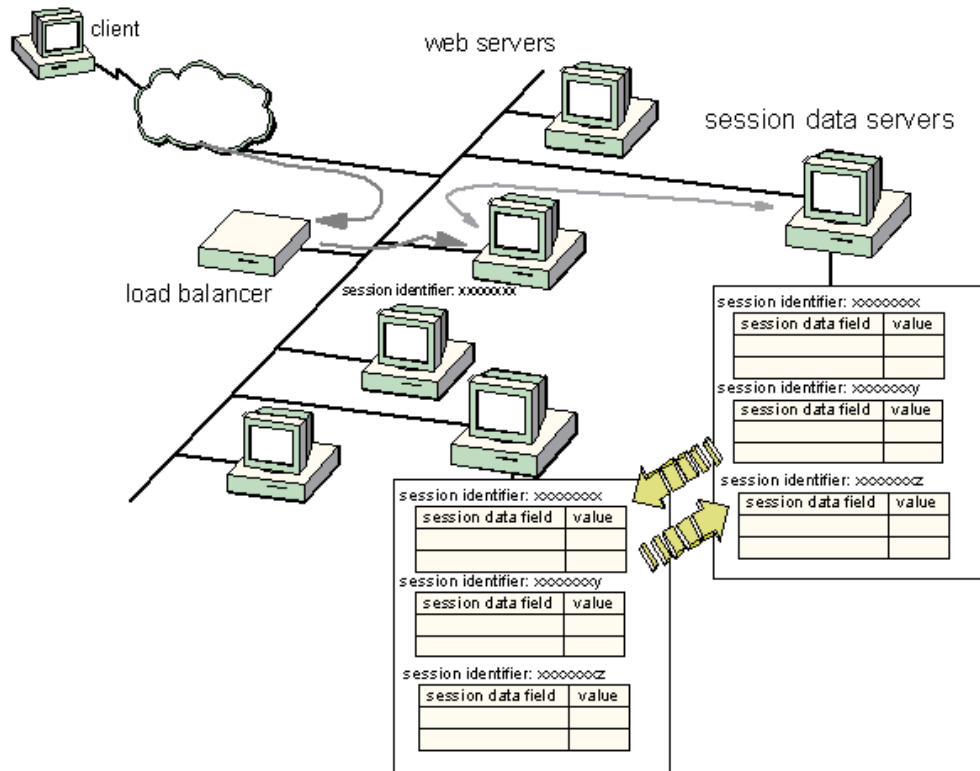
Session data is kept in a distributed data store

For applications with high cluster synergy, session data is kept in a distributed data store that has its own availability and scalability characteristics. Session data is cached at the originating server.

The originating server takes an overt action to encourage the load balancer to return subsequent requests from the same identity (based on client IP address or URL or header content).

Alternatively, the load balancer can detect the preferred server from the content of the request. If that server is not available or is heavily loaded, another server can handle the request by retrieving the session data from the session data store.

The session data store is distributed. That is, as the number of sessions increases and the number of server nodes increases, requests to the session data store can be distributed over multiple data store nodes. If one data store node dies, another copy of the session data is available from another node.



Load balancing feedback

Applications with high cluster synergy have feedback mechanisms that detect whether the application is healthy, whether the web server is healthy, and whether access to back-end data sources is available.

Applications in this class incorporate feedback mechanisms that alert the load balancer that they may be overloaded in spite of appearing “normal” based on default observations. For example, a web application might include a Network Dispatcher custom advisor that would indicate when some sort of garbage collection or log roll-over is in progress. Such events do not affect the availability of the server but result in temporarily decreased server capacity.

Summary

The following table summarizes the session clustering techniques we’ve discussed. Note that for most of the proposed solutions, there is at least one “drop-dead” issue, indicated with a gray background, which ultimately limits the effectiveness of the approach. The two solutions that don’t have a drop-dead issue are propagation-on-demand and the distributed data store.

Level of cluster synergy	Low			Medium			High
Description / characteristics	Session data is stored in memory on the owning web server or in a location known only to that server. Session data is not accessible from other servers in the cluster. Subsequent requests from the client must be handled by the same web server or the session information is lost.	Site is partitioned into "stateful" and "nonstateful" sections. URLs and links are constructed so that only stateful requests use sticky ports.	All session data is contained in the request itself; for example, stored in cookies sent with each request from the browser, allowing any web server in the cluster to handle any request without loss of session data.	Session data is propagated to all servers in the cluster.	Session data is kept on the owning server but can be retrieved by another server in the cluster. Also known as "propagation on demand". Session data may be cached by other servers. Servers may interact with the load balancer to encourage affinity.	Session data is kept in a central data store.	Session data is kept in a distributed data store.
Feedback mechanisms	Load balancer can tell whether trivial requests are responded to.			Load balancer can tell that the application on the server is active and healthy; for example, it knows whether the application on the server has a working connection to the enterprise database server.			Servers inform the load balancer if they are overloaded in spite of appearing normal.
Typical ND cluster implementation	Sticky port, standard advisors		Standard advisors	Custom advisors or ISS daemons on the server feedback load information to the Network Dispatcher.			
Availability	A lost server results in lost sessions. Loss is limited to the sessions on that server, and is no worse than in a non-clustered environment.		Good availability. Lost server does not imply any lost sessions.	A lost server results in some lost sessions. Loss is limited to the sessions on that server and other servers may have cached the lost session data.		Loss of the central data store results in lost sessions. All sessions are lost if the central data store is unavailable.	Loss of a server or a data store node does not imply lost sessions if the distributed data store has availability characteristics of its own.
Scalability	Limited by the effects of stickiness on load-balancing. In the worst case, all active sessions might have to be handled by the same server, regardless of the number of available servers.	The cost of maintaining accurate URLs and links will grow as the site grows. The web application developers must be aware of and conform to the site partitioning plan.	The server must parse cookies or headers on each request, increasing the time required to process the request. There are limits on the size of URLs and cookies and on the number of cookies.	As the number of servers increases, it will take longer and longer to propagate the data for one session. As the number of concurrent sessions increases, more time will be spent propagating the session data.	As the number of sessions and servers increases, it will be less likely that the subsequent request will return to the same server. Intra-cluster communication increases unless servers interact with load balancer to generate affinity.	As the number of sessions and servers increases, it takes longer and longer for the central data store to satisfy requests.	Good if distributed data store is scalable. The cost of storing and retrieving session data does not increase as the number of sessions or the number of servers increases.
Load balancing	Rules about which server should handle the request are based on the active sessions and not on the load characteristics of the server, which can defeat load balancing algorithms.	For the stateful partition, sticky ports can defeat load balancing algorithms.		Because the load balancer is not aware of the intra-cluster communication associated with maintaining session state, it might direct requests to a machine that is busier than it can detect.			
Security	Session data is protected by the web server.		Session data is public.	Session data is protected by the web server.		Session data is protected by the data store.	

End of Document