Originally presented at CMG 2000

E-COMMERCE RESPONSE TIME: A REFERENCE MODEL

Chris Loosley, Keynote Systems Inc.
Richard L. Gimarc and Amy C. Spellmann, HyPerformix Inc.

cloosley@keynote.com rgimarc@hyperformix.com amy@hyperformix.com

This paper presents a framework that describes all the components that make up the response time of an e-commerce application. Understanding this fundamental framework is the key to implementing responsive e-commerce systems. The paper discusses examples of how e-commerce application response times can be improved in three ways: by reducing the overall number of components, by speeding up individual components, and by moving some components off the synchronous response time path.

Introduction

The Internet is now a fact of business life for most companies, whose customers and business partners expect to be provided with a way to do business online. People now expect to have access to Web-based business functions that they can reach at any time, from anywhere on the World Wide Web, using only a browser. And with universal access rapidly becoming the norm, it is a rare company that can afford to remain on the sidelines.

As business moves to the Web, scores of older text-based applications are being given more graphic Web front ends, while newer "client/server" applications, already possessed of graphic user interfaces, are being re-written for the Web. At the same time, the Internet is spawning an array of new technologies for producing, distributing, and using rich computer media, and many new business applications are being developed to exploit these technologies.

The exponential growth in Internet traffic has consistently kept pace with -- and sometimes threatens to overwhelm -- an equally explosive growth in network bandwidths. As a consequence, the vast shared network of networks that is the Internet is still a relatively slow and unpredictable vehicle for electronic commerce [NIEL1998].

Designing for such an environment poses a challenge for the application developer. Business transactions are not confined to the work week, when participants communicate from their offices using high-speed Internet connections. Eventually, the same business functions must be available from home in the evening through a slow dial-up connection, and in the middle of the day from a laptop with a wireless connection in an airport lounge. In all these environments, users expect the interface to be reasonably responsive, and will look for alternative ways to conduct their business if it's not.

A few years ago perhaps, when the Web was in its infancy, and Web-based access to anything was a novelty, people were more tolerant of the Internet's highly variable performance. But familiarity and competition is continually raising the bar of acceptability. Today, slow Web pages drive away customers as surely as long checkout lines in the supermarket. Even the often-quoted rule of thumb that "customers click away after waiting 8 seconds" [BICK1997] is probably too generous for many kinds of interactions, especially in business-to-business applications where the client is accustomed to high-speed connections.

The result is that e-commerce application developers *must* design with performance in mind, because users regard an unresponsive site as "unusable" [NIEL2000]. Specifically, a developer must see clearly how application design decisions will affect the response time experiences of a range of typical users. However, such insights require analysis, because a Web application's response time depends on a complex interplay of software, hardware, and networking technologies.

This paper presents a standard framework for that analysis. By enumerating the components of response time, and explaining the factors that determine the duration of each component, the response-time reference model provides the application designer with more than a simple checklist or classification scheme. If properly used, it can form the foundation of a systematic design review process, exposing likely performance problems, and revealing possibilities for improving actual and perceived response times.

Properties of a Reference Model

One might imagine that the requirements for a reference model are fairly obvious. Clearly the primary need is a useful classification of the topic of interest, or "domain of discourse." A reference model must supply a systematic scheme for classifying a topic into subtopics or components. It must clarify the boundaries between those subtopics or components, and explain how the parts combine to make up the whole.

Because more than one such decomposition is possible, we must settle on two key properties of the reference model: its *purpose* and its *level*:

- Purpose: A reference model does not exist in a vacuum; it must provide a framework for some kind of analysis relating to the domain of discourse. A useful reference model will support and encourage analysis, revealing design or tuning possibilities.
- Level: Any concept complex enough to justify a reference model can probably be decomposed into elements which can be further decomposed into smaller elements; how granular should the model be?

These issues are interrelated; neither can be resolved without first defining a target user for the model. In this case, the domain of discourse is the e-commerce application; the target user is an application or systems designer or developer, or a performance analyst concerned with e-commerce application performance management. Our hypothetical performance analyst is primarily concerned with *e-commerce application flow* and the *components of application response time*.

A Two-Dimensional Model

The choice of a hypothetical user determines our approach. Other participants in e-commerce, even though they may be concerned with aspects of e-commerce performance, may find our model to be at the wrong level. It may be too basic for a business analyst, too general for a network engineer, too broad for a database administrator, or too shallow for a systems administrator. However, this model reveals the various stages that combine to create the user's experience, and breaks the overall response-time into components that are amenable to different optimizations.

Although system-level issues like device capacities and utilizations do affect application response times, they will not be the primary focus of our model. At its simplest level, our reference model – depicted graphically in Figure 1 -- can be visualized as a two-dimensional matrix of cells, the horizontal dimension being *application flow* and the vertical *response-time components*.

Horizontal Dimension

In an e-commerce application, each business transaction is implemented by means of *a sequence of Web pages*. Without any further elaboration, this observation provides us with a natural partitioning of the horizontal dimension into Web pages, as shown in Figure 1. For any *particular* e-commerce application, there will be distinct classes of Web pages (such as Home, Login, Catalog, Order), typically with different performance characteristics. But all of that is refinement; the basic model is simply a series of Web pages. Later we illustrate this dimension of our model, using an example of an e-commerce application to show ways of reducing response times.

Vertical Dimension

To partition the vertical dimension, we must examine the process by which a single Web page is served. Partial descriptions of this process can be found in previous CMG papers such as [JAMT1997] and [LYNC1999]; for more detail see [SHUL1998] and the World Wide Web Consortium site, www.w3.org.

	Web Page 1	Web Page 2	…	…	Web Page n
DNS Lookup					
TCP Connection					
Redirection					
Server Processing					
Base Page Download					
Content Download					
Page Rendering					
User Interaction					

Figure 1. The Basic Reference Model

Level of Detail

We emphasize that the reason for partitioning overall response time *at this level of detail* is to delineate its major components, focusing attention and promoting performance analysis and optimization. The model highlights the largest components of response time, and creates a natural framework for any design and tuning work. Simply put, there are just two ways to lower the response time of an e-commerce application: make some cells of the matrix smaller, or remove some cells altogether.

The model, however, is simply a framework for this analysis. It does not supply all the information a performance analyst needs, and does not in any way preclude one from conducting more detailed analyses. As we illustrate in our later example, when considering *how* to make a particular cell smaller, it will sometimes be useful to create submodels *within* one or more of the individual cells of the matrix.

Response Time Stages

We now describe the page download process for a fictitious Web site, www.byzz.com, breaking it into eight distinct stages.

This analysis deliberately excludes two preliminary stages. A user must first *establish a physical connection* to the Internet. This can be done by connecting through a private (corporate) network that's connected directly to the Internet, or by dialing into an access device (a modem) at a local Point of Presence (POP) in a commercial Internet Service Provider's (ISP) network. That access device is connected, through one or more routers, to the Internet. Before or after establishing an Internet connection, the user also *starts a Web browser*. In our model, we assume that the browser has been started and a connection established.

In the model, Web page response time begins when a user directs the Web browser to retrieve a page by entering a URL (a uniform resource locator, or Web address, like *www.byzz.com/shop.html*), or clicks on an embedded link on a page.

A succession of eight stages follows. As the ensuing paragraphs make plain, these eight stages are not simply a convenient way to describe the subdivision of overall Web-page response time into smaller components; they are fundamentally distinct steps in the production of a Web page. Each involves a distinct domain, most depend for their performance on distinct technologies, and each requires a distinct set of skills. Figure 2 summarizes these differences.

This is an important insight. In the Preface to his book on “Web Performance Tuning“ [KILL1998], Killelea lists his audience as System Administrators, System Architects, System Integrators, Web Application Programmers, Web Content Developers, and Webmasters. We could add Database Administrators, Network Designers, and Network Engineers, and still not cover all the people who play a part in determining application performance at most large companies. To ensure acceptable e-commerce performance, companies must devise management processes that integrate these diverse skills.

Typical Response Times

In the sections that follow, we describe each stage, discuss the principal factors that determine the response time of each, and give typical response-time ranges. To illustrate the range of page sizes and response times that are representative of leading e-commerce sites, we use measurements of the Keynote Business 40 (KB40) Index of 40 leading sites. Each site is measured every 15 minutes from over 60 locations in 25 major US metropolitan areas [KEYN2000].

Stage	Domain knowledge needed to optimize performance
DNS Lookup	DNS/BIND
TCP Connection	Networking, TCP/IP, Internet connectivity
Redirection	Web site implementation, HTTP
Server Processing	Web page generation: Systems design, Systems integration, Web servers, Application servers, Database servers, Middleware
Base Page Download	Web page construction using HTML, Scripting languages
Content Download	Graphics, Content distribution networks
Page Rendering	Web browser operation, Client OS tuning
User Interaction	Human factors, User interface design using HTML

Figure 2. Knowledge Needed to Optimize Performance at each Stage

Figure 3 summarizes 3.5 million individual data points collected between September 3rd, 2000 (12:00AM) through September 17th, 2000 (11:59PM). These summary statistics are all based on the average times (measured in seconds) and the average page sizes (measured in bytes) for each of the KB40 sites. For example, "Min" is minimum of the averages computed for each of the 40 sites, not the minimum of all 3.5 million data points. For the six components of response time, Figures 4a-4f show the frequency distributions of average times for the 40 sites.

These measurements were obtained via direct high-speed connection (T1 or faster), as is common in a business environment. At these connection speeds, response times are governed by the speed of the Internet, whereas at slower speeds, the speed of the connection itself is the limiting factor. In particular, for users with slower dial-up connections, response times will be substantially longer for the *Base Page Download* and *Content Download* components, because these involve downloading the most data.

	Average	Median	Std Deviation	Bottom 15%	Top 15%	Min	Max	% Variation
DNS Lookup	0.02	0.01	0.03	0.05	0.00	0.01	0.11	1000%
TCP Connection	0.10	0.09	0.04	0.13	0.08	0.05	0.26	420%
Redirection	0.32	0.27	0.16	0.44	0.17	0.16	0.61	281%
Server Processing	0.18	0.12	0.25	0.21	0.09	0.03	1.58	5167%
Base Page Download	0.51	0.44	0.32	0.78	0.22	0.01	1.50	14900%
Content Download	1.77	1.52	1.21	2.77	0.87	0.12	5.54	4517%
Total Download Time	2.64	2.27	1.49	4.02	1.56	0.62	7.72	1145%
Base Page Size (bytes)	26190.38	22782.50	15249.81	39195.95	13125.50	1005.00	77122.00	7574%
Content Size(bytes)	47891.53	39277.50	35583.31	81465.70	12240.50	6871.00	157204.00	2188%
Total Page Size(bytes)	74081.90	65275.00	43006.64	114201.25	35103.20	26672.00	220501.00	727%
Objects(bytes)	20	17	11	30	9	3	62	1967%

Figure 3. KB40 Summary Data

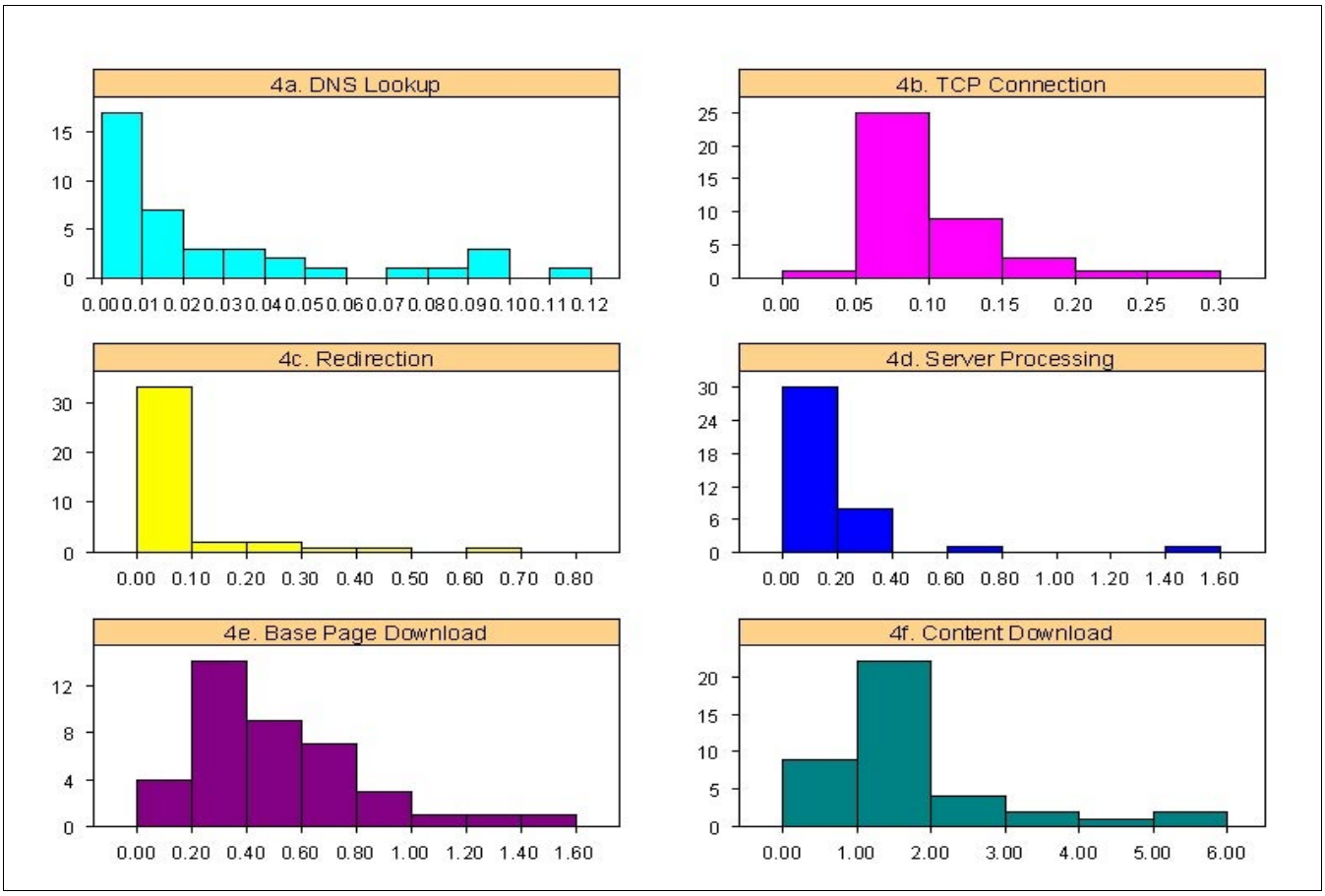

Figure 4. Component Response Time Distributions for the KB40

DNS Lookup

First the browser software sends a message over the physical connection to the local network or access provider's Domain Name System (DNS) server. The DNS is like a distributed telephone directory; it translates the *domain name* in the target URL (*www.byzz.com*) into the actual Internet Protocol (IP) address of that destination -- a four-part number like 204.10.195.74.

The translation process relies on DNS directory entries that are controlled by the domain owner, in this case, the e-commerce site *byzz.com*. Directory entries have associated time-to-live (TTL) values. If an entry expires and a URL cannot be translated locally, the request is routed to a higher level DNS server, and could eventually be sent to the *authoritative name server*, which is normally located at the e-commerce site. Latency is introduced whenever a DNS request is not satisfied from the first name server.

Figures 3 and 4a show that, for leading e-commerce sites, average DNS times fall between 0.01 and 0.11 seconds. Values in Figure 3 are rounded; the fastest sites achieve DNS Lookup times of 0.001 seconds.

Sophisticated Internet content distribution schemes are built on the basic DNS mechanisms [POTT2000]. They use frequently updated DNS entries with short TTL values to distribute requests among a set of content servers, dynamically redirecting the request to the most appropriate server. The goal is usually either to use a server that is relatively lightly loaded, or one that is close to the requester's location. However, some content servers (like those devoted to advertisement banners) may even maintain user profiles and use databases that record which content items users have previously received. In the more complex schemes, a hierarchy of communicating directory servers may handle each name request. The end result may indeed be a faster content download time, but the name server response times and the inevitable inter-server communication times involved in these dynamic schemes can lengthen the DNS Lookup time component.

TCP Connection

Once the browser software knows the IP address of the URL, it sends a *TCP connection* request to the destination address (204.10.195.74 in this example). This is analogous to dialing a telephone number on a fax machine before sending a fax. Routers in the Internet forward the connection request in a series of hops to its ultimate destination. If the destination Web server is willing to accept the connection (it is not down, or overloaded at the time), it accepts it by sending a response to the browser. The TCP Connection is now complete, and a stream of data packets can flow in both directions.

The combined time for a request and its corresponding response is often referred to as *round-trip time*. On the Internet, this name is especially apt. Unless the ISPs serving the user and the e-commerce site are both connected to the same Internet backbone provider, the routes taken by requests and responses will normally differ, and packets generally do take a round trip from browser to Web server and back again.

Because a TCP connection request is handled by communication software (the TCP stack), the response is generated without invoking any application-level processing on the Web server. Unless the server is completely swamped with requests, this is normally a very short interaction. As a result, the response time for the TCP connection stage is a very good measure of the minimum round-trip Internet latency for a browser-server pair.

Outside the firewall, TCP connection times depend on the locations of the browser and server (geographically, and their Internet backbones), on the amount of competing traffic on those backbones, and especially on the routers at their peering point(s) – the locations at which the backbone providers exchange traffic. Normally, Internet traffic is heaviest in the middle of the business day, and lightest after midnight. Most popular e-commerce sites aim to minimize their reliance on a single backbone by using multiple servers that are hosted on more than one network.

Inside the firewall, for companies that host their own Web sites (as opposed to using the services of an ISP that specializes in Web hosting), TCP connect times are also affected by the performance of the corporate network where the server resides. Load balancing devices that divide incoming requests among a network of servers inevitably add some latency, as, of course, will any congested router or switch on the path to the target server.

Figures 3 and 4b show that, for leading e-commerce sites, average TCP Connection times fall between 0.05 and 0.26 seconds.

Server Processing

The Web browser now uses the TCP connection to send a request to the Web server for a particular Web page. For example, it may ask for the page "/home.html," a common situation. Or it may ask for a more complex page, such as "/company/products/sale/item6.html." The simple set of request and reply commands used to identify the requests and their possible replies – such as "get this page," "data successfully retrieved," or "page not found" – is known as the *Hypertext Transfer Protocol*, or *HTTP.*

When requests arrive at a Web server that can satisfy them (see "Redirection", below), one of two scenarios ensues. In the simplest case -- a *static* page -- the Web server locates and sends a fixed HTML file that corresponds to the requested page. In the more

complex case -- a *dynamic* page -- the page to be sent is generated by the Web server.

During the generation process, literally anything can happen. The Web server may pass control to an application server, which may in turn retrieve content from a database server, or any other file server. Managing the interconnections among all these servers may involve a variety of middleware technologies, and any amount of network traffic. Because of this complexity, this is one stage in our model that may merit another level of analysis, and perhaps a submodel. We illustrate this in our later example.

It is difficult to generalize about server processing times. The fastest Web servers can locate static Web pages practically instantaneously, certainly in less than 0.1 seconds. For dynamic pages, because of the sheer number of possibilities, a much wider range of response times are possible. However, a time of greater than 0.5 seconds for this stage would certainly qualify as a potential target for tuning.

Figures 3 and 4d show that, for leading e-commerce sites, average Server Processing times fall between 0.03 and 1.58 seconds.

Redirection

Sometimes, instead of sending the requested page, the Web server will return a *redirect* command (HTTP 301 or 302) to the browser, instructing it to fetch a replacement URL, which is not necessarily located on a different server, but can be anywhere. This redirection process can be repeated multiple times before the browser finally contacts a server that will actually return a Web page. Perhaps the most common use of redirection in e-commerce is to route requests that will involve e-commerce transactions to a separate secure server, where the user may enter a user identifier, password, or other “login” information.

This is not the only use of redirection, however. Sites built using some high-level site generation tools use redirection to implement a dynamic dispatching mechanism that routes requests among the generated pages. In the most flagrant examples of this technique, three or four redirection steps can occur between successive pages. Because it adds server time to resolve the redirected request, and network communication time as the redirect command travels back to the browser and then on to the new target, this use of redirection should ideally be kept to a minimum, or avoided altogether. Equivalent function can be implemented much more efficiently using server-side scripts that do the job without looping back to the browser.

In most cases, the redirection stage is not present, so the time it takes is obviously zero. But when it occurs, redirection can add anything from 0.1 seconds to a full second to the average page download time. Figure 4c shows that 32 of the 40 leading e-commerce sites (80 percent) have no redirection component, while Figure 3 shows that the non-zero times range from 0.16 to 0.61 seconds.

Base Page Download

The page requested by the browser is encoded in a computer language known as *Hypertext Markup Language*, or *HTML*. HTML contains both the text of the page, and instructions for displaying that text. If the page is constructed as an HTML frameset, the highest level frames are fetched immediately following the Base Page that contains the HTML FRAMESET command.

Because it typically involves transmitting kilobytes of data from the server to the browser, this stage takes longer than any of the previous stages. And because at this stage of the process the browser is single threaded, Base Page Download time does depend directly on the size of the Base Page. Therefore reducing Base Page size will usually cut overall response time. This situation is a lot simpler than the Content Download stage, discussed next.

Even so, file transmission time is not proportional to file size. One obvious reason is that Internet throughput depends on the level of congestion at any time. Another complication is that Internet data packets vary widely in size. Although servers use Maximum Transmission Unit (MTU) settings such as 576 bytes and 1500 bytes for transmitting data, protocols require many other packets as small as 40 bytes.

TCP’s “slow-start” protocol also complicates matters. This scheme operates by first sending -- depending on the server configuration -- either one or two packets. Then, provided packets sent continue to be received successfully, the number of packets is doubled in each subsequent transmission. The purpose of this protocol is to avoid flooding the Internet with data packets that would only have to be re-transmitted when congestion occurs. But the consequence in the Web environment is that – since most files are relatively small -- TCP is usually operating at the slowest end of the slow-start process. One benefit of *persistent connections* (described below) is that they can lessen the impact of the slow-start protocol.

Figures 3 and 4e show that, for leading e-commerce sites, average Base Page Download times range from 0.01 to 1.5 seconds, with a strong peak between 0.2 and 0.4 seconds.

Content Download

Most modern Web pages comprise more than text alone; they include a lot of graphics and sometimes other pieces of content, such as *applets* (small computer programs). These *content elements* are not included in the HTML that comprises the Base Page. Instead, the HTML contains instructions for finding those items on the Web – i.e., it includes their URLs (such as www.byzz.com/page5graphics/picture8.gif).

The browser, following those instructions, then downloads each required content element.

Broadly speaking, the time for this stage is a function of the number of content elements to be downloaded and their total size. However, Content Download is another stage in our model that may merit a deeper analysis, and perhaps a submodel. Retrieving each content element involves many or all of the same stages that were needed to download the Base Page. For example, if the content is located on a different server, the browser may need to invoke DNS again. In fact, the DNS-based content distribution schemes we described earlier (see the section on *DNS Lookup*) are rarely used for the Base Page, usually coming into play only when the browser requests the content elements.

But compared to retrieving the Base Page, the content download process can be more efficient in two significant ways. The first occurs when the server and browser both use the HTTP 1.1 *persistent connection* protocol, which is implemented by most current releases of server and browser software. This protocol allows the server to retain an open TCP connection with the browser between successive browser requests, so that the browser does not have to issue a TCP connect for the second and subsequent downloads from the server [NIEL1997].

Persistent connections are something of a double-edged sword however. Although they do reduce download times, they require the server to set aside sockets for inactive TCP connections, which eat up server memory [COHE1999]. As a result, servers have a maximum concurrent connections setting, and a timeout setting for inactive connections. Ultimately, a large pool of concurrent TCP connections may consume memory that could have been used for other server functions like content caching, slowing other service times. The overall result could even be a net loss in performance.

For this reason, some heavily used sites like search engines sites do not enable persistent connections, even though they are running Web server software that supports HTTP 1.1. Another reason why some widely used sites set up their servers to use HTTP 1.0 is to reduce or eliminate the need for the special browser detection logic and redirection that is needed to deal with older browsers that cannot handle HTTP 1.1.

The second principal difference in the way browsers download content elements is that they can use more than one parallel thread. Popular browsers download two (Internet Explorer) or up to six (Netscape) content elements concurrently [WANG1998]; once any thread completes a download, it is reassigned to another. This process is plainly visible in Figure 5, which illustrates a trace of the download process for a page comprising 21 content elements, using four concurrent threads. (In this case, the “browser” is actually a Keynote Systems measurement agent, which mimics a browser’s behavior and times the outcome). Consider also the consequence of a download that takes longer than the average time: one thread will be kept busy, leaving just three to work on the remaining content elements.

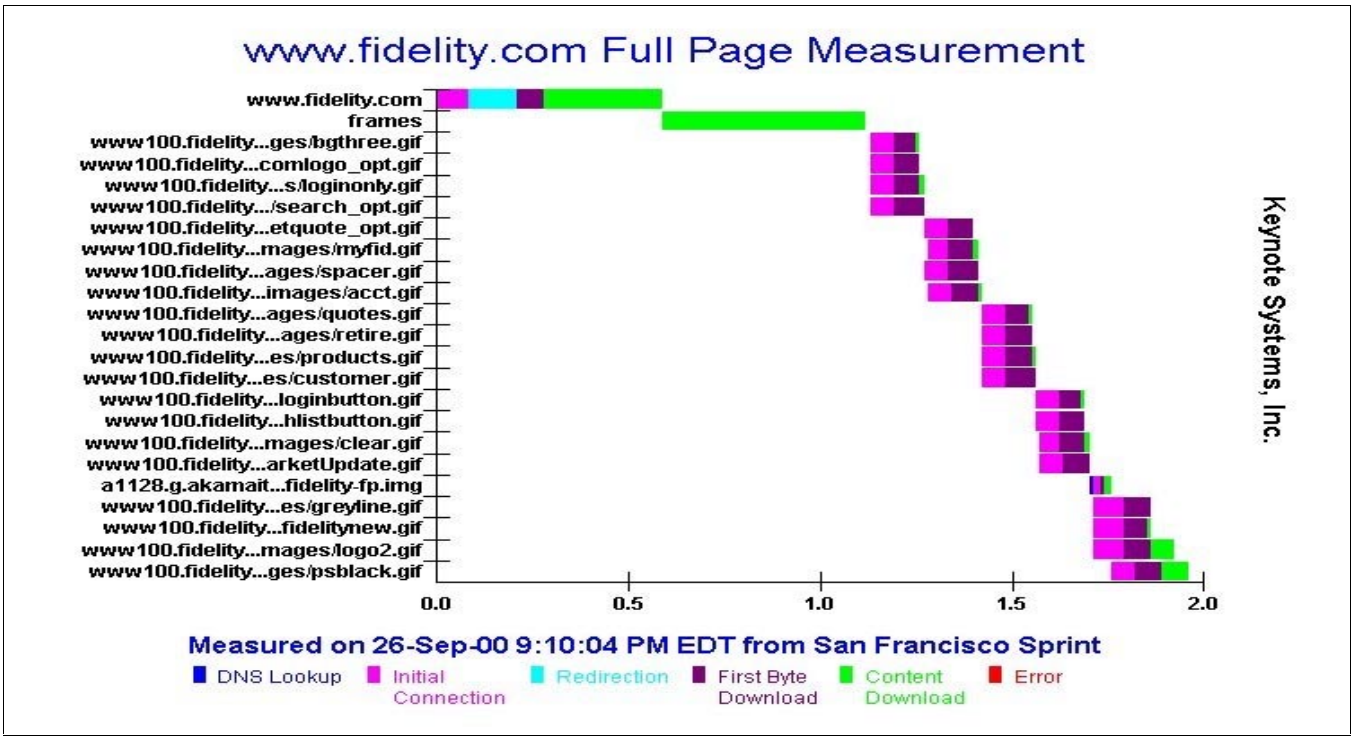

Figure 5. Page Download Trace

To sum up, although Content Download time tends to increase with the number and size of the content elements, in practice the relationship is a complex one. By using content distribution schemes, large sites can outperform smaller ones hosted from a single location. Figures 3 and 4f show that, for leading e-commerce sites, average Content Download times range up to 5.5 seconds, with a strong peak between 1 and 2 seconds.

Page Rendering

The browser usually displays the graphics as it receives them. However, some combinations of HTML commands can delay the display of parts of a page, or, in an extreme case, the entire page. One common case occurs when an HTML TABLE has been used to lay out a page, a fairly common alternative to using HTML frames. If a single TABLE is used to organize the entire page, then the browser cannot render the page until the size of each table cell is known. And if one or more cells contains a graphic (HTML IMG) element whose size in pixels was not specified (using HEIGHT and WIDTH tags), then the browser does not know the size of the image until after it has been downloaded. In this situation, the browser can either delay the rendering process until the element has been downloaded and its size established, or render the page partially and re-render it after the missing information shows up. Browsers vary in their willingness to draw and re-draw a single page, but neither result is particularly desirable.

The solution, of course, is for site developers to ensure that HEIGHT and WIDTH parameters are supplied for all imbedded graphics. A reason for *not* doing so is that the size is unknown when the HTML code is being developed. A variety of images may be possible at this page location; perhaps they will even be substituted dynamically by a script. In this situation, the browser dynamically re-sizes the image to the HEIGHT and WIDTH parameters supplied, regardless of its actual dimensions, which can result in images being displayed with a distorted aspect ratio. To solve this problem, site designers can either enforce standard image sizes, or maintain dimensional metadata with stored images, and generate the matching HTML parameters whenever an image is used.

One page design technique that reduces rendering time is to reuse content elements in different pages of a site. For most users, when a Web page contains a content element that the browser has already encountered, the element is not downloaded a second time, it is retrieved from the browser cache instead. A browser's willingness to use its cache in this way is controlled by a browser setting that site developers do not control. However, most users do have their browsers set to use caching, at least during the current session. Recognizing this, the most efficient e-commerce sites are designed to exploit this browser feature by reusing common graphic elements whenever possible.

User Interaction

Once the browser has displayed the page, the user must absorb enough of the content to proceed. The time required for this stage depends a lot on the context: the user may need to enter data into an HTML form, or simply click one of several links that will take them to the next page of the transaction. But whatever the interaction, a well-designed page can certainly help to minimize this component of overall transaction time and improve the user's perception of the application's usability and responsiveness.

[HANS1999] discusses how the "Think Time" that proponents of Software Performance Engineering (SPE) typically ignore as an uninteresting (and unmodeled) delay actually turns out to be easily the largest component of overall application response time. The paper shows how Human Factors Engineering (HFE) methods can be combined successfully with SPE, to provide new and more effective ways to improve the users' perceptions of performance.

Layer	Methods of reducing overall response times
Transaction	Reduce page count, repeat content elements across pages
Servers	Speed up Web page generation process
Web Page	Reduce page size, reduce content element count, reuse elements
HTML	Simplify and optimize HTML code, reduce page rendering time
HTTP	Maintain persistent connections between browser and server
TCP	Speed up round-trip times between browser and server
IP	Shorten routes, reduce hop-count, avoid congested peering points
Network devices	Minimize device congestion, device latencies, transmission delays

Figure 6. E-commerce Functional Layers

Reducing Response Time: A Summary

We have enumerated eight stages of an e-commerce transaction, noting the principal performance issues, and suggesting some techniques for reducing the response times of each. Our choice of a hypothetical user of our model determined this approach. However, another way to approach the question of tuning possibilities would have been to work systematically through the technology layers that underlie an e-commerce application. These are shown in Figure 6, which summarizes the principal methods of reducing overall response times through tuning actions that can be taken at each layer.

Figure 7 shows the relationship between these functional layers, the suggested tuning techniques, and the eight response-time components of our reference model. A blank cell means that the tuning action has no effect. The figure is a guide only; some tuning benefits will depend on factors that are beyond the scope of this discussion, and cannot be represented in a simple table. For example, whether or not the DNS component is affected will depend on whether a DNS lookup is needed for any page or content element.

Each non-blank cell of the table contains a code. The character indicates the effect of the tuning action. An "E" indicates that a tuning action will *eliminate* that response time stage, while an "S" indicates that a tuning action will *shorten* that response time stage. The subscript indicates the scope of the result produced by the tuning action. A "1" indicates that the action affects a single Web page of an e-commerce transaction, a "2" indicates that the action may affect more than one of the pages, and an "*" indicates that all pages will be affected.

Mathematically, eliminating any cell of the response-time matrix is merely a special case of reducing its contribution to overall response time. However, in practice there is a qualitative difference between the types of tuning actions involved, and for this reason we prefer to emphasize the idea that there are two distinct classes of tuning options. In particular, in the quest for more responsive applications in the relatively slow environment of the Internet, an important design principle is "*Whenever feasible, take work off the synchronous response-time path*". Gimarc and Spellmann observe that "The E-commerce vendor must understand and balance the synchronous and asynchronous transaction components in order to satisfy the customer's service expectations" [GIMA1999]. A later example illustrates this point.

Layer	Tuning action	DNS Lookup	TCP Connection	Redirection	Server Processing	Base Page Download	Content Download	Page Rendering	User Interaction
Transaction	Reduce page count	E_1	E_1	E_1	E_1	E_1	E_1	E_1	E_1
	Repeat content elements						S_2		
Servers	Speed up page generation				S_1				
Web Page	Reduce overall page size					S_1	S_1	S_1	S_1
	Reduce content element count or reuse elements within a page						S_1		
	Reuse domain name(s)	E_1					S_1		
HTML	Optimize HTML code					S_1		S_1	
HTTP	Maintain persistent connections		E_2			S_2	S_2		
TCP	Speed up round-trip times		S_*	S_*		S_*	S_*		
IP	Shorten routes, reduce hop-count, avoid congestion		S_*	S_*		S_*	S_*		
Network devices	Minimize device latencies and congestion		S_*	S_*		S_*	S_*		

Figure 7. E-commerce Tuning Actions and Their Benefits

Finally, we point out that Figure 7 follows the basic definition of our reference model. In particular, any DNS Lookup, Redirection, or TCP Connection times associated with downloading page content elements are included within the Content Download component, and not in the other columns, which relate to the Base Page component.

An Example: Keynote's Web Broker Trading Index

To demonstrate the reference model, a well-known Internet benchmarking application – the Keynote Web Broker Trading Index ("KBTI") -- will be used. This weekly index represents the typical response times and success rates for executing a standard stock-order transaction on about 20 leading online brokerage sites (such as DLJ Direct, Fidelity, E-Trade, and Charles Schwab). These sites are measured every 15 minutes between 9 a.m. and 4 p.m. (EST) during market trading days via T-1 lines from 10 major US metropolitan areas [KEYN2000].

The Sample KBTI Transaction

The standard Keynote broker transaction begins by entering the Web site through its home page and logging into the trading area. The transaction then obtains a stock quote, creates an order to buy stock, and logs out before confirming the order. Each transaction is created through a standard trading account set up by Keynote Systems. Demonstration accounts or other non-standard accounts are not used for these measurements. This multi-page KBTI transaction will be used to illustrate the Reference Model.

In practice, online brokerage sites may use a wide variety of servers and require anything from 6 to 10 Web pages to implement a brokerage transaction like the KBTI. For this example, we assume that the KBTI transaction is implemented using the following server environment:

- **Public Web Server** provides entry to the Web site through the site's home page
- **Secure Web Server** performs login processing and other secure actions
- **Application Server** contains the business application logic
- **Quote Server** is a stand-alone machine that provides stock quotes on demand
- **Backend Mainframe** contains the site's permanent customer records and core trading application software

We also assume that the KBTI application flow involves the following six-page sequence of Web pages:

1. **Home Page**: The browser requests the home page. The Public Web Server services this request. After the home page is displayed, the browser client then selects the link for logging into the system.
2. **Login Page**: This page allows the user to enter their id and password and log into the system. The Login Page request is sent to the Public Web Server where a redirect is performed to the Secure Web Server. At the completion of this page request the user has a form where they can enter the information required to log into the system. The user enters their identification and presses the "log in" button to log into the system.
3. **MyAccount Page**: After performing the required authentication and authorization, the system returns the MyAccount page. This is the Base Page for performing activities from the user's brokerage account.
4. **Get Quote Page**: From the MyAccount page, the user submits a request for a stock quote. Quote information is returned to the browser.
5. **Place Order Page**: The client then places an order to buy a stock, specifying quantity, price, etc. A confirmation page is returned to the browser so that the user can confirm the buy order.
6. **Logout Page**: Instead of confirming the buy order, the user cancels the order by pressing the "log out" button. This terminates the KBTI transaction, returning the user to the Login Page.

This implementation is purely fictional, but similar to many real-world implementations. The intent here is to provide a realistic application that can be used to illustrate the utility of the Reference Model for describing and evaluating performance. Figure 8 describes the sample KBTI application in terms of the Reference Model.

The "greyed out" cells in the matrix denote response time components that will appear as zero in the sample model. The following implementation notes apply to Figure 8:

1. The Login Page is displayed using a redirect command from the Public to the Secure Web **Server**. It is assumed that the browser cache has saved a copy of the Public Web Server's IP address.
2. The model assumes that the browser is using a persistent connection to the Web servers. Thus, TCP Connection only appears for the first connection made to each Web server.
3. After the browser is redirected to the Secure Web

Server on the Login Page, it will use its locally cached IP address for future communication with the Secure Web Server.

4. TCP Connection between the browser and Secure Web Server is performed as part of the Login Page's Redirection.

For simplicity, the Page Rendering and User Interaction stages are not included in our example. In practice, these components of response time could be added if estimates or measurements are available. See, for example, [HANS1999].

	Home Page	Login Page	MyAccount Page	Get Quote Page	Place Order Page	Logout Page
DNS Lookup	Look up Public Web Server	Look up Public Web Server[1]	Look up Secure Web Server[3]	Look up Secure Web Server[3]	Look up Secure Web Server[3]	Look up Secure Web Server[3]
TCP Connection	Connect to Public Web Server	Connect to Public Web Server[2]	Connect to Secure Web Server[4]	Connect to Secure Web Server[4]	Connect to Secure Web Server[4]	Connect to Secure Web Server[4]
Redirection	None	Redirect from Public to Secure Web Server	None	None	None	None
Server Processing	Fetch Home Page	Fetch secure Login Page	Perform login	Obtain stock quote	Set up a stock buy order	Cancel order and log off
Base Page Download	Send Home Page	Send Login Page	Send MyAccount Page	Send Get Quote Page	Send Place Order Page	Send Logout Page
Content Download	12 Objects	6 Objects	6 Objects	2 Objects	8 Objects	1 Object

Figure 8. Sample Implementation of the KBTI Application

	Home	Login	MyAccount	Get Quote	Place Order	Logout
DNS Lookup	0.06	-	-	-	-	-
TCP Connection	0.15	-	-	-	-	-
Redirection	-	0.19	-	-	-	-
Server Processing	0.11	0.72	2.20	0.51	4.92	0.30
Base Page Download	1.30	1.54	1.30	1.06	0.89	0.82
Content Download	3.20	1.08	0.82	0.39	1.09	1.35
Page Total	4.82	3.53	4.32	1.96	6.90	2.47
KBTI Total	24.00					

Figure 9a. KBTI Baseline Values

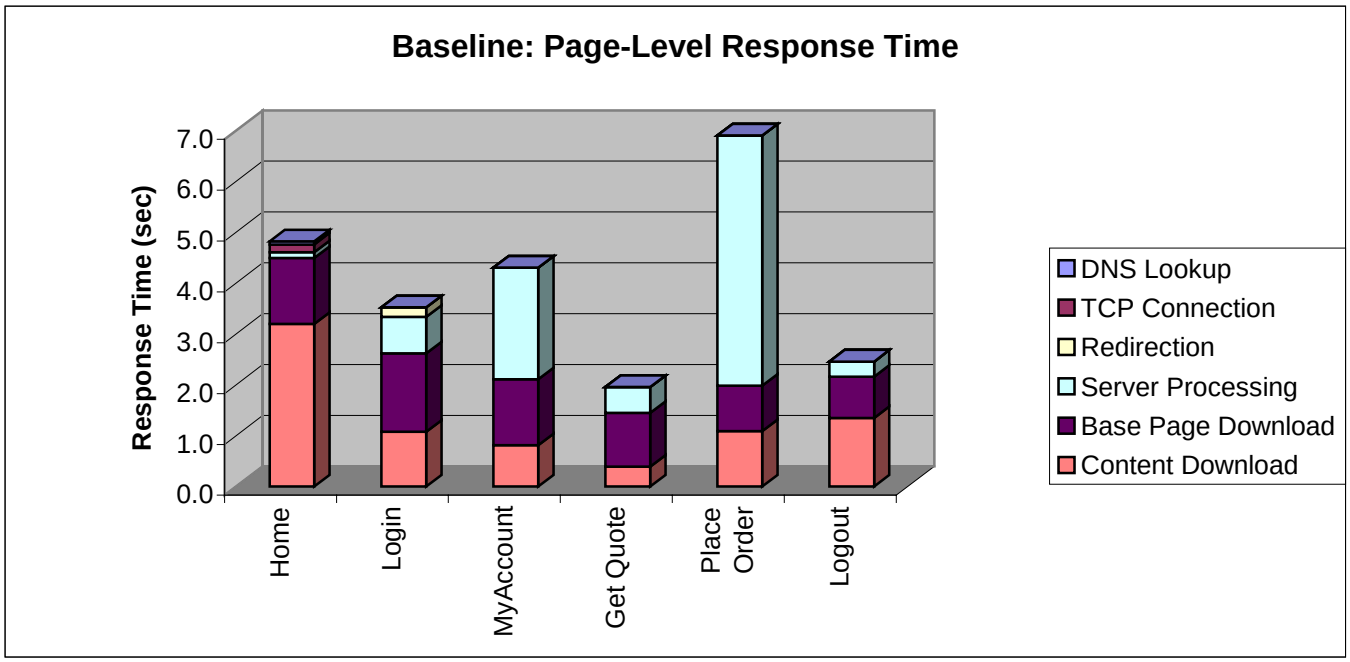

Figure 9b. Plot of KBTI Baseline Values

Baseline Performance

Figures 9a and 9b show the baseline values for the Reference Model components.

The total response time of 24 seconds is rather large compared to the (real-world) KBTI average of 13.72 seconds reported for the Index on September 18, 2000.

By examining the components identified by the Reference Model, the following areas of optimization are worth further investigation:

1. The Content Download component of the Home Page accounts for approximately 66% of the page's response time (over 3 seconds). The Reference Model defines this component as the time required to download the page's content; e.g., graphic objects. This component may be improved by reducing the amount of content to be downloaded per page and by moving static content closer to the requesting browser (content distribution).
2. The Base Page Download component is a significant factor in all page response times (29% of the overall KBTI response time). This is the time required to send the Base Page from the Web server back to the browser. Potential improvement may come from relocating the Web servers closer to the requesting browser.
3. The Place Order page is the single largest contributor to the KBTI response time. Within that page, Server Processing accounts for almost 5 seconds. It may be possible to partition Server Processing into separate synchronous and asynchronous parts (where the asynchronous part would not contribute to browser response time).

In the next section, these three alternatives will be explored in more detail.

Simulation Model of the Index

To evaluate the different optimizations identified in the previous section, a simulation model of the KBTI transaction was constructed -- see Figure 10. The model includes representations of the hardware, software, and network components that comprise this sample KBTI implementation. The round "arrow bubbles" connected to each server represent the application software services provided by that server.

The model provides a means to evaluate the component times of the Reference Model. In addition, the model enables individual Reference Model components to be represented in varying degrees of detail. For example, the Reference Model includes a Server Processing component. In the simulation model, this single component can be expanded to represent the discrete steps that comprise Server Processing. For example, the Server Processing steps for the Place Order Page include the following:

1. The Secure Web Server sends a request message to the Application Server over the switched Ethernet.
2. The Application Server performs initial processing on receipt of the message to determine other services required.

3. The Application Server then sends a request to the Quote Server to get the current stock price.
4. The Quote Server queries its database to determine the current price and returns its reply to the Application Server.
5. The Application Server continues processing and prepares a request for the Backend Mainframe.
6. The Application Server sends the order request to the Backend Mainframe.
7. The Backend Mainframe processes the order and returns a reply to the Application Server.
8. The Application Server completes order processing and sends its response back to the Secure Web Server (which will then send an order confirmation back to the browser).

The ability to represent the application at varying levels of detail enables performance analysis to be focused. Response time components that are small (e.g., DNS Lookup and TCP Connection) can safely be viewed from a high level. Larger components such as Server Processing warrant a more detailed representation. The issues being addressed by the performance analyst determine the level of detail and refinement to be included in the model.

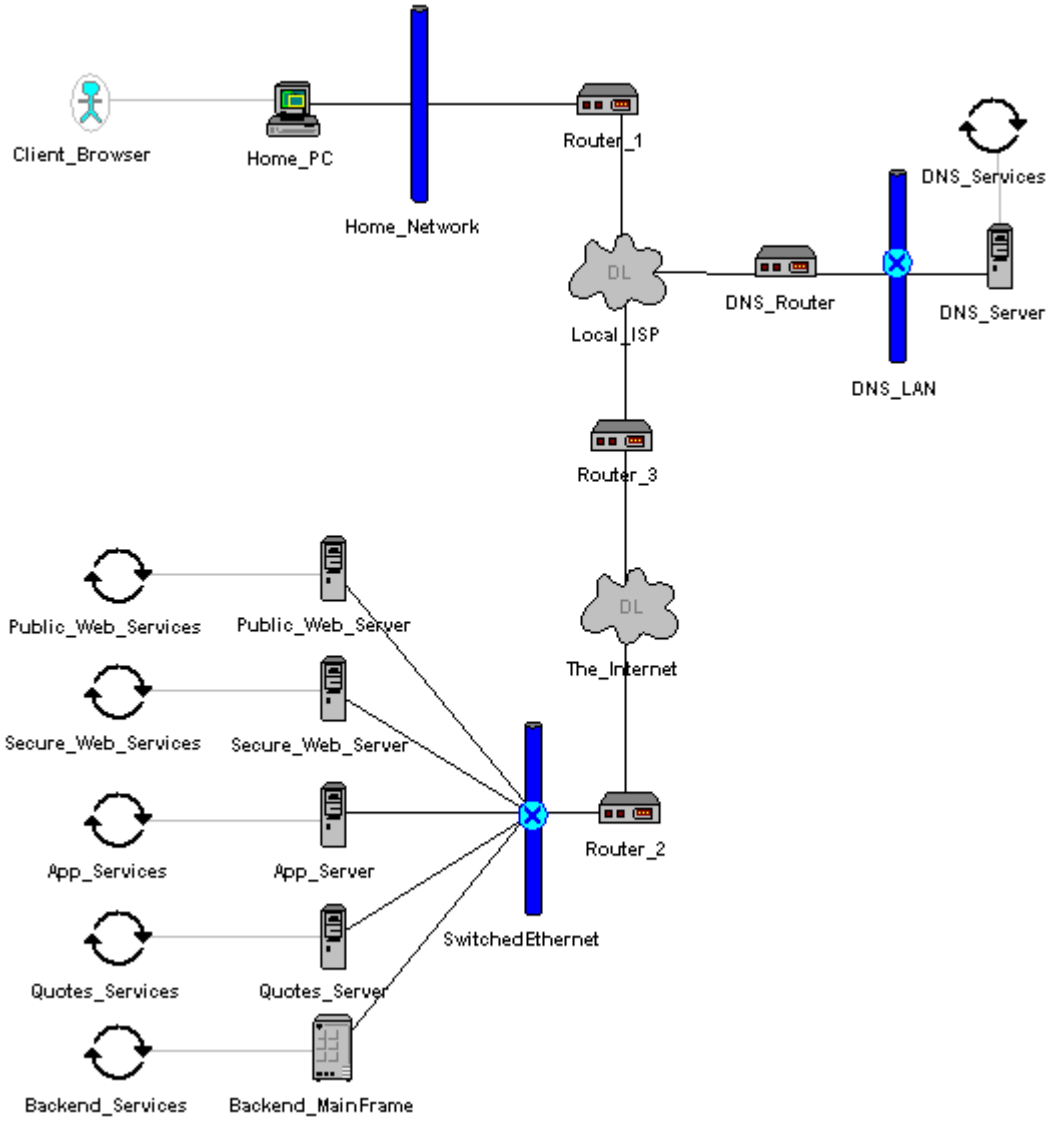

Figure 10. Simulation Model of the KBTI

Model Evaluation

The model was used to evaluate four key changes to the application implementation:

1. Reduce the number of objects on the Home Page to improve the page's Content Download component.
2. Distribute static content to decrease network and Content Download time.
3. Use asynchronous processing to improve the Server Processing for the Place Order Page.
4. Relocate Web servers to decrease Base Page and Content Download.

Finally, we looked at the cumulative effects of these changes.

1. Reduce Home Page Graphic Content

The Content Download component of the Home Page accounts for approximately 66% of the page's 4.82 second response time. A fairly simple optimization is to reduce the graphic content on this page. In this example there are 12 graphic objects on the Home Page. The model will be reevaluated after deleting the last 6 objects of varying sizes totaling 25Kb.

The Home Page results are shown in Figure 11. The Home Page response time decreased by 43%. This represents a 9% reduction in the KBTI response time.

	12-Object Home Page	6-Object Home Page
DNS Lookup	0.06	0.06
TCP Connection	0.15	0.15
Redirection	-	-
Server Processing	0.11	0.11
Base Page Download	1.30	1.30
Content Download	3.20	1.12
Page Total	4.82	2.74
KBTI Total	24.00	21.90

Figure 11. Revision to the KBTI Home Page

2. Content Distribution

Content distribution is becoming a popular method of reducing the download time of static Web page content. The general idea is to move static content closer to the requesting users. By doing so, the Content Download time for the pages containing distributed content will be reduced. In this example, the static content on the Home Page (10 objects totaling 26Kb) is moved to a content server closer to the requesting browser.

Figure 12 shows the Home Page results from this experiment compared to the baseline. Distributing static content resulted in approximately the same savings as redesigning the Home Page by eliminating 6 objects. Moving static content closer to the end user's browser reduced the Content Download component.

	Baseline Home Page	Distributed Content Home Page
DNS Lookup	0.06	0.06
TCP Connection	0.15	0.15
Redirection	-	-
Server Processing	0.11	0.11
Base Page Download	1.30	1.30
Content Download	3.20	1.07
Page Total	4.82	2.70
KBTI Total	24.00	21.85

Figure 12. Distributing the KBTI Content

3. Asynchronous Processing

Server Processing in the Place Order Page is the single largest component in our model of the KBTI transaction. In fact, the 4.9 seconds is larger than the response time for any of the other 5 pages. As Gimarc and Spellmann point out, "the introduction of asynchronous components in the E-commerce transaction requires recasting transaction boundaries in order to maintain the integrity of the customer's order." [GIMA1999] This requires the steps for the Place Order Page to be divided such that "… the state information maintained by the system and application for the transaction satisfies the application's business rules and requirements, and integrity constraints."

In this example we will modify the Application Server software so that as soon as it has saved a persistent copy of the order, it will send a confirmation back to the client's browser. The work required to perform the time-consuming order processing will be executed in parallel, or asynchronously. This parallel processing will be performed in the background while the client receives a faster response confirming that the order will be placed as requested.

The results of this change are shown in Figure 13. Partitioning the Place Order Page into synchronous and asynchronous components resulted in a 19% reduction in the KBTI response time and a 66% reduction of the page-level response time.

	Baseline Place Order Page	Asynch Place Order Page
DNS Lookup	-	-
TCP Connection	-	-
Redirection	-	-
Server Processing	4.92	0.36
Base Page Download	0.89	0.89
Content Download	1.09	1.09
Page Total	6.90	2.34
KBTI Total	24.00	19.41

Figure 13. Asynchronous Processing for the KBTI Order Page

4. Web Server Relocation

In our final experiment we will study the effect of relocating the Web servers closer to the requesting browser. This requires setting up a widely distributed network of servers, located at ISPs in major hub cities. In terms of our model, we will be moving the Public and Secure Web Servers closer to the "Local_ISP". The intent of this move is to reduce the amount of time required to transmit the Base Page and content to the client browser. Within the hub city areas, the Web servers will be located nearer to the client browsers and therefore provide faster downloads due to decreased network latency.

However, there is a second-order effect that must be considered, namely the communication that occurs between the Web servers and the other backend servers. Relocation will increase the network latencies between these servers. A balance must be struck between these two effects to determine the right combination.

Figure 14 gives the results from relocating the two Web servers. These results assume that the client browser is located near the hub city ISP.

Web server relocation reduced overall KBTI response time from 24.0 seconds down to 16.6 (a 31% reduction). Note, however, that certain times increased due to the physical separation of the Web servers from the backend servers. Server Process time for the Place Order Page increased from 4.92 seconds to 7.08 seconds.

Cumulative Changes

In the final experiment, we apply the two relocation changes (static content and Web servers) and add the asynchronous step to the Place Order Page. The results are shown in Figure 15; we've reduced the KBTI transaction response time by 50%, from 24 seconds down to 12 seconds.

	Home	Login	MyAccount	Get Quote	Place Order	Logout
DNS Lookup	0.06	-	-	-	-	-
TCP Connection	0.03	-	-	-	-	-
Redirection	-	0.04	-	-	-	-
Server Processing	0.03	0.52	2.74	1.05	7.08	1.41
Base Page Download	0.32	0.38	0.32	0.26	0.22	0.20
Content Download	0.80	0.26	0.20	0.09	0.27	0.33
Page Total	1.24	1.20	3.26	1.40	7.57	1.94
KBTI Total	16.61					

Figure 14. Distributing the KBTI Web Servers

	Home	Login	MyAccount	Get Quote	Place Order	Logout
DNS Lookup	0.06	-	-	-	-	-
TCP Connection	0.03	-	-	-	-	-
Redirection	-	0.04	-	-	-	-
Server Processing	0.03	0.52	2.74	1.05	2.51	1.41
Base Page Download	0.32	0.38	0.32	0.26	0.22	0.20
Content Download	0.80	0.26	0.20	0.09	0.27	0.33
Page Total	1.24	1.20	3.26	1.40	3.00	1.94
KBTI Total	12.04					

Figure 15. Cumulative Effects of Tuning Changes

Conclusion

We have described a simple reference model for e-commerce application response time. The model delineates the components of response time, and creates a natural framework for design and tuning work. For each component of the model, we explained the factors determining response time, and discussed the typical response times observed for leading e-commerce Web sites in September, 2000. Using an example, we showed how this conceptual reference model can be implemented as simulation model, allowing an analyst to explore the consequences of design choices on the response-time of an e-commerce transaction.

Acknowledgements

We would like to thank Ben Rushlo, Eric Siegel, Geoff Woolhouse, and Jing Zhi of Keynote Professional Services for their invaluable assistance with the research for this paper.

References

References of the form CMGCD *id* indicate a paper number on the CMG 25th Anniversary Conference Edition CD, published in 1999. When using the CD, the *id* field can be used to locate the paper quickly.

[BICK1997] Peter Bickford, "Worth the Wait?," Netscape/View Source Magazine 10/97 (1997)

[COHE1999] E. Cohen, H. Kaplan, and J. D. Oldham, "Managing TCP Connections under Persistent HTTP," Computer Networks. 31:1709--1723 (1999)

[GIMA1999] Richard L. Gimarc and Amy C. Spellmann, "Redefining Response Time in an Asynchronous World," Proc. CMG'99 International Conference, Reno (1999). CMGCD 99INT445.

[HANS1999] Craig D. Hanson and Pat V. Crain, "User and Computer Performance Optimization - A New Model for Efficiency," Proc. CMG'99 International Conference, Reno (1999). CMGCD 99INT436.

[JAMT1997] Sudha Jamthe and Subhash Agrawal, "Performance Issues of Designing a High Performance Intranet," Proc. CMG'97 International Conference, Orlando (1997). CMGCD 97INT633.

[KEYN2000] For details of the sites in Keynote Performance Indexes and the measurement locations used, see http://www.keynote.com/

[KILL1998] Patrick Killelea, "Web Performance Tuning," O'Reilly (1998)

[LYNC1999] Jacqueline A. Lynch, "Designing High Performance Web Pages," Proc. CMG'99 International Conference, Reno (1999). CMGCD 99INT332.

[NIEL1997] Henrik F. Nielsen, "Network Performance Effects of HTTP/1.1, CSS1, and PNG," http://www.w3.org/Protocols/HTTP/Performance/Pipeline.html (1997)

[NIEL1998] Jakob Nielsen, "Nielsen's Law of Internet Bandwidth", http://www.useit.com/alertbox/980405.html (1998).

[NIEL2000] Jakob Nielsen, "Designing Web Usability: The Practice of Simplicity," New Riders, p42 (2000).

[POTT2000] Kevin Potts, "Exploiting the New Internet Capacity Management Technologies," Proc. CMG2000 International Conference, Orlando (2000).

[SHUL1998] Rus Shuler, "How Does the Internet Work?," http://rus1.home.mindspring.com/whitepapers/internet_whitepaper.html (1998)

[WANG1998] Zhe Wang and Pei Cao, "Persistent Connection Behavior of Popular Browsers," http://www.cs.wisc.edu/~cao/papers/persistent-connection.html (1998)

