Enigma of Estoeric Nothingness

A Performance Study of Web Access to CICS
by Phil Wakelin, Graham Rawson and et al.
ISBN: 0738415286
IBM Redbooks © 2000 , 244 pages
A detailed examination of potential hardware and software bottlenecks encountered when exposing CICS systems onto the Internet.

Alan Zeichick

A Performance Study of Web Access to CICS

For CICS Transaction Server Version 1 Release 3 and OS/390 Version 2 Release 7

Phil Wakelin,

Graham Rawson,

Per Fremstad,

Dave Scott,

Andy Abbey

For CICS Transaction Server Version 1 Release 3 and OS/390 Version 2 Release 7International Technical Support Organization

www.redbooks.ibm.com

IBM International Technical Support Organization

Take Note!	Before using this information and the product it supports, be sure to read the general information in Appendix D "Special notices" on page 209 .

First Edition (February 2000)

This edition applies to Version 1, Release 3 of CICS Transaction Server for OS/390 (program number 5655-147); Version 3, Release 1 of the CICS Transaction Gateway for OS/390 (program number 5648-B43), and Version 1, Release 1 of WebSphere Application Server for OS/390; for use together with the OS/390 Version 2 Release 7 Operating System.

Comments may be addressed to:

IBM Corporation, International Technical Support Organization
Dept. QXXE Building 80-E2

650 Harry Road
San Jose, California 95120-6099

When you send information to IBM, you grant IBM a non-exclusive right to use or distribute the information in any way it believes appropriate without incurring any obligation to you.

2000-10-25

Preface

The objective of this redbook is to help you understand the performance impact of Web-enabling your CICS-based applications. It gives detailed performance measurements and capacity planning information for Web access to CICS Transaction Server V1.3 when using OS/390 V2.7. The redbook *Revealed! Architecting Web Access to CICS* , SG24-5466 explains the choices available to help you decide which is the best solution to choose.

The CICS Web-enabling technologies covered in this redbook are: the CICS Web support function of CICS Transaction Server V1.3, including usage of the 3270 bridge; the OS/390 Web server, which is currently available as OS/390 WebSphere Application Server; and the CICS Transaction Gateway for OS/390 V3.1. It also contains performance information on securing CICS Web support using SSL.

First, we give an overview of the different technologies and discuss the key factors affecting performance of CICS and Web solutions. Following this, there is a summary of the performance figures for each of the technologies we cover. Included is a simple methodology for OS/390 capacity planning when using each technology, and a worked example of how to apply this methodology to the sample "Trader" application.

We then present a summary of our conclusions and performance recommendations, and go on to describe a fictional story of the Trader Company to illustrate how our capacity planning calculations could be used. Finally, all the actual performance data and the details of the test environments are documented.

The studies presented in this book were designed for the purpose of comparing the OS/390 CPU usage of each technology. They were all simple test applications and were run in controlled laboratory conditions at the IBM Hursley Laboratory, UK. As such, the results provide a good comparison of each technology and with care can be used for capacity planning purposes. However, any capacity planning estimate you use, whatever the source, should always be verified on a test system before the application is put into production.

This redbook applies to Version 1, Release 3 of CICS Transaction Server for OS/390 (program number 5655-147); Version 3, Release 1 of the CICS Transaction Gateway for OS/390 (program number 5648-B43), and Version 1, Release 1 of WebSphere Application Server for OS/390; for use together with the

OS/390 Version 2 Release 7 Operating System.

The team that wrote this redbook

This redbook was produced by a team of specialists from around the world working at the International Technical Support Organization San Jose Center.

Phil Wakelin is a senior I/T specialist at the International Technical Support Organization, San Jose Center, and has 9 years experience working on most platforms and versions of CICS. He writes extensively and teaches IBM classes on all areas of CICS, specializing in the CICS client-server and Web technology. Before recently joining the ITSO, Phil worked in the Installation Support Center, IBM UK as a pre-sales support specialist for CICS client-server.

Graham Rawson is a member of the CICS/390 Performance group working in the CICS/390 Development group based at IBM Hursley Laboratory in Winchester, England. He has 15 years of experience with CICS. He has recently worked as a CICS Technical Support specialist with the Installation Support Centre specializing in CICS/390, CICSPlex SM, and MVS Parallel Sysplex exploitation.

Per Fremstad is a certified I/T specialist from IBM Norway, currently on assignment to the EMEA S/390 New Technology Center in Montpellier, France. He has worked for IBM since 1982 and has extensive experience with S/390 and Large Systems. His areas of expertise include the Web and the Web enabling of applications on OS/390. He teaches frequently on Web and Java topics, especially at the IBM's customer briefing center in Montpellier.

Dave Scott is a senior I/T specialist with IBM Global Services, in the US. He has 12 years experience working with CICS, the past 2 years with IBM. His current responsibilities include working in the field with customers implementing a variety of CICS solutions.

Andy Abbey is a member of the CICS/390 Performance group working in the CICS/390 Development group based at IBM Hursley Laboratory in Winchester, England. He has spent 13 of the 25 years he has worked for IBM within the CICS group, both as a developer and as a performance specialist.

Thanks to the following people for their invaluable contributions to this project:

Yvonne Lyon, Emma Jacobs, Elsa Martinez, HansPeter Nagel, Eugene Deborin, Mary Comianos, Laymond Pon, John Byrd of the International Technical Support Organization, San Jose Center

John Burgess, Paul Harris, Richard Cowgill of the IBM CICS/390 Performance group, IBM Hursley.

Nigel Williams, Geoff Sharman, Chris Goodall, Steve Longhurst, Peter Havercan, Steve Wood, John Kaputin, IBM Hursley.

Carl Parris, Judi Bank, IBM Poughkeepsie.

Carol Shanesy, Leigh Compton, IBM Dallas Systems Support Center.

Norbert Verbestel, IBM Belgium

John Green, IBM Toronto.

Bob Yelavich, CICS consultant.

Comments welcome

Your comments are important to us!

We want our redbooks to be as helpful as possible. Please send us your comments about this or other redbooks in one of the following ways:

- Use the on-line evaluation form found at http://www.redbooks.ibm.com/
- Send your comments in an Internet note to < redbook@us.ibm.com >

Part 1: Performance and CICS Web-enabling

Chapter List

Chapter 1: CICS and Web-enabling

Overview

In this section of the redbook we give a brief overview of the CICS Web-enabling technologies we cover in this book, together with reference information, if you wish to find out more about these technologies.

A detailed overview of the current strategic CICS Web-enabling options is given in the redbook *Revealed! Architecting Web Access to CICS*, SG24-4566, and the *CICS Web/selection guide whitepaper*, available at:

- http://www.ibm.com/software/ts/cics/library/whitepapers/cicsweb

These two sources of information detail the following four CICS Web-enabling technologies:

- CICS Web support (CWS)
- CICS Transaction Gateway (CTG)
- CICS CORBA client support
- Host On-Demand

This performance study only covers the CWS function in CICS TS V1.3 and the OS/390 CICS Transaction Gateway. If you wish to find more information about CICS CORBA Client support and Host On-Demand, you should refer to the following documentation:

- *Java Application Development for CICS* , SG24-5275
- *IBM SecureWay Host On-Demand: Enterprise Communication Era Network Computing* , SG24-2149

First, we will give a brief overview of the principles of CICS modular design as it relates to CICS Web-enabling, before presenting a general introduction to the CWS and the OS/390 CTG technologies.

1.1 The separation of presentation and business logic

A sound principle of modular programming in CICS application design is to separate the presentation logic from the business logic. Such a modular design provides a separation of functions. Communication between the programs is by using the EXEC CICS LINK command, and data is passed between such programs in a COMMAREA (communication area). The structure of this data in the COMMAREA is also part of the application design. This is illustrated in Figure 1 .

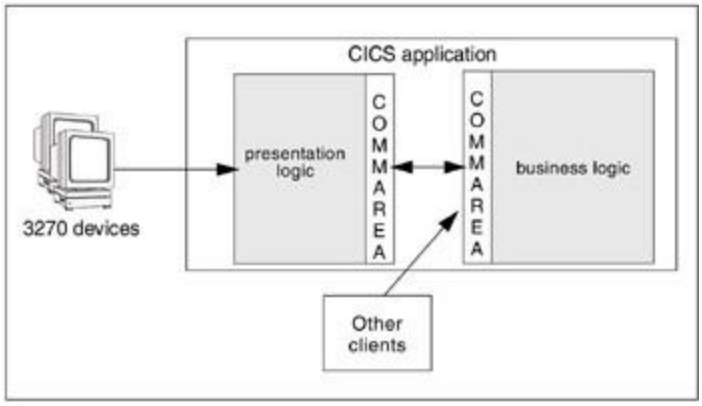


Figure 1: Separation of CICS business and presentation logic

The separation of business and presentation logic enables the programs that control the user interface (presentation logic) to be separated from the programs that perform the actual business requests (such updating database entries). These programs are still executed together as a single CICS task, but if designed in this modular form, then they can readily exploit the distributed program link (DPL) and workload management functions provided by CICS to spread work within a sysplex or between CICS systems distributed across a network.

Further, if the business logic of a transaction is isolated from the presentation logic and given a communication area (COMMAREA) interface, it is available for reuse with different presentation methods. This means it can be invoked from a variety of sources; such as:

- From the CICS Universal Client using the External Call Interface (ECI) running on a workstation.
- From a program where the presentation logic is HTTP-based (Web-aware).

- From a Java applet or servlet using the facilities of the CICS Transaction Gateway and the ECI Java methods
- From a CORBA client using the IIOP protocol and the JCICS classes.
- From another program running in the OS/390 Sysplex using the EXCI (External CICS Interface) interface (such as a Web server ICAPI or CGI program).
- From any program which uses a CICS LINK and a COMMAREA structure to pass data.

Don't forget that there is a restriction on the size of data that can be passed in a CICS COMMAREA. The maximum size of this area is 32 KB. With CWS in CICS TS V1.3 you now have the choice of using the WEB API to send and receive HTTP datastreams and so are no longer subject to this 32 KB restriction.

Many legacy applications were not designed or written with a separation of presentation and business logic, and are often deemed too difficult or costly to re-engineer. For that reason IBM has developed Web-enabling technologies which allow re-use of the 3270 interface as well as technologies which utilize a callable COMMAREA interface.

1.2 CICS Web support

CICS Web support (CWS) provides client Web browsers with direct access to CICS programs or transactions running in an OS/390 CICS region. The base requirements for this function are provided in CICS/ESA V4.1, but significant enhancements are provided in CICS Transaction Server (CICS TS) V 1.3, which is the subject of this redbook.

CWS and CWI

In CICS Transaction Server V1.3, the CICS Web functionality, previously known as the CICS Web Interface (CWI), was split into the listener support for TCP/IP and the protocol support for HTTP, and was also internally redesigned. This book now refers to the CICS HTTP protocol support as CICS Web support.

1.2.1 CICS Web support

CICS Web support (CWS) is a set of resources supplied with CICS TS V1.3 that provide CICS with some functionality similar to a real Web server. A summary of this function is illustrated in Figure 2 .

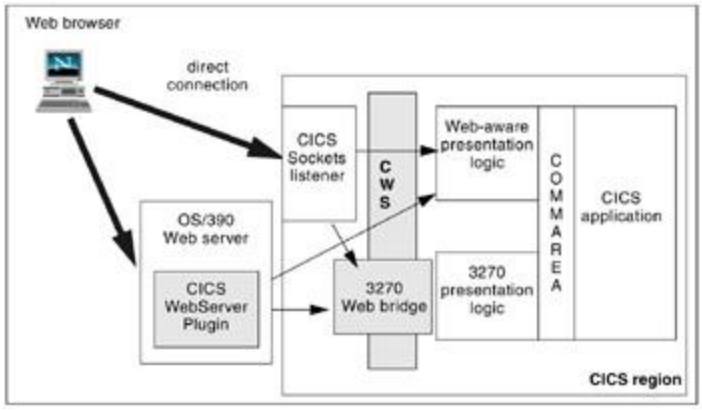


Figure 2: CICS Web support

CWS provides a native HTTP interface to CICS, this interface can be used by both 3270 based transactions and applications that provide a callable COMMAREA interface. Two different configurations can be used to route the HTTP requests into the CICS region. Both configurations allow the use of the same facilities in CICS, although the configuration of the two options is significantly different. These configurations are as follows:

- A **direct connection** from a Web browser to CICS. This uses the facilities of the CICS Sockets listener to pass the requests directly into CICS Web support.

- Through the OS/390 Web server using the facilities of the **CICS WebServer Plugin** (DFHWBAPI). This is a CICS supplied extension to the OS/390 Web server. It routes requests into the CICS Web support in a CICS region using the EXCI communication mechanism.

With both, the direct connection and the CICS WebServer Plugin, CWS can be used to invoke two types of CICS applications.

- To invoke a **3270 transaction** , the facilities of the CICS 3270 bridge are used. The 3270 transaction remains unchanged and the 3270 output is converted to HTML. We will refer to this function as the **3270 Web bridge** . This function is only available when using CICS Transaction Server V1.2 or higher.

- To invoke an application that provides a callable COMMAREA interface, some new CICS presentation logic must be written. This logic uses CICS facilities to interpret, act upon, and then build and return the HTTP datastream. We will refer to a CICS application containing such logic as " **Web-aware** ". This Web-aware logic can be contained either within the program or in a separate presentation module that is linked to by the application. To create this Web-aware presentation logic there are two different methods provided by CWS:
  - WEB API
  - COMMAREA manipulation

The WEB API, together with the DOCUMENT API and TCPIP API, provide a rich set of functions to interpret, manipulate, and build the HTTP datastream within a CICS application. They are part of the new function of CWS in CICS TS V1.3, and are described in more detail in chapter 12 of *CICS Internet Guide* , SC34-5445, and chapter 3 of *CICS Transaction Server for OS/390 Version 1, Release 3: Web Support and 3270 Bridge* , SG24-5480.

The COMMAREA manipulation technique was originally introduced with CWI support in CICS/ESA V4.1. It uses the CICS COMMAREA as a buffer for transferring the HTTP datastream along with a range of utility programs to manipulate the datastream. The CWS HTML template manager program (DFHWBTL) is used to build the response. This technique is still available in CICS TS V1.3, but for ease of use and higher functionality, we recommend use of the WEB API.

1.2.2 Using a CWS direct connection

Figure 3 illustrates the major components of CICS Web support when using Web-aware presentation logic via a direct connection to CICS.

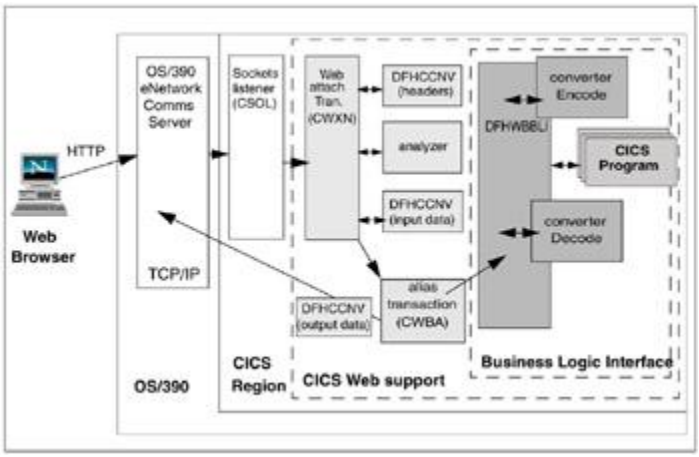


Figure 3: CICS Web support — direct connection

CICS Sockets listener

- The CICS Sockets domain provides TCP/IP support to handle requests for internal CICS functions that use TCP/IP services, currently HTTP and IIOP support. The CICS Sockets listener is an internal CICS function serviced by the private CSOL transaction, and should not be confused with the CICS TCP/IP Sockets interface. Unlike the CICS Sockets listener, the CICS TCP/IP Sockets interface provides an application level socket interface to the CICS application, and is described further in the redbook *CICS/ESA and TCP/IP for MVS Sockets Interface* , GG24-4026.

Web attach transaction

- The Web attach transaction(CWXN) performs the Web attach processing. It invokes the DFHCCNV data conversion routines, links to the specified analyzer, and then invokes the alias. The CWXN task will terminate after invoking the alias, unless persistent HTTP connections are used.

DFHCCNV

- The DFHCCNV data conversion routines are invoked by the Web Attach processing to convert the HTTP headers and user data from the ASCII code page of the Web browser client to EBCDIC and back.

Analyzer

- The purpose of the analyzer is to analyze the incoming HTTP request. It decides if the request will be executed in the CICS system and if so, which resources are required. It uses the information in the URL to decide the name of the alias transaction, converter and user program to be invoked. The analyzer can also be modified so as to use HTTP basic authentication to check the authenticity of each HTTP request.

Alias

- The alias transaction is invoked by the analyzer. The default alias transaction code is CWBA, but this can be modified. The Alias transaction invokes the program DFHWBA, which links to the business logic interface.

Business logic interface

- The business logic interface (BLI) is an externally callable interface that allows a client to invoke the business logic in an application. It is implemented by the module DFHWBBLI. It provides a mechanism for implementing Web-aware presentation logic in the " **converter** ". The converter provides **Decode** and **Encode** routines to receive and send the HTTP presentation logic. Note that it is possible to bypass the converter and implement the Web-aware logic in a separate module which would communicate directly with the business logic via a COMMAREA interface.

1.2.3 Using the CICS WebServer Plugin

An alternative approach to accessing CICS Web support is through the services of the OS/390 Web server, using the CICS WebServer Plugin, (DFHWBAPI). In this implementation, some of the function previously handled through the CICS-supplied programs for CICS Web support is now replaced by function within the Web server.

The OS/390 Web server has been rebranded at various times to reflect its positioning within IBM's Internet product portfolio. The Internet Connection Secure Server (ICSS) Web server became the Lotus Domino Go Webserver for OS/390, which has now been rebranded as WebSphere Application Server for OS/390. Whatever server you are using, we will refer to it as the OS/390 Web server.

The CICS WebServer Plugin replaces the functionality of the CWS analyzer, described previously. The OS/390 Web server has to be configured with a service directive in order to function with the CICS WebServer Plugin. This configuration is described in the *CICS Internet Guide* , SC34-5445. Using this service directive, the OS/390 Web server receives the HTTP request, builds an EXCI request, and invokes the BLI using the CSMI mirror transaction in the target CICS region. The HTTP datastream is passed to the BLI in the EXCI COMMAREA.

Figure 4 illustrates the major components of CICS Web support when using Web-aware presentation logic via the CICS WebServer Plugin.

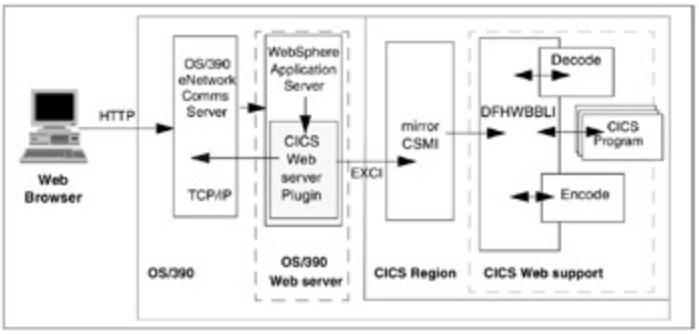


Figure 4: CICS Web support, with the CICS WebServer Plugin

The same facilities within CICS are available using the CICS WebServer Plugin as using a direct connection, but there are a few important differences, which are summarized below:

- The OS/390 Web server and the CICS region must be running within the same OS/390 image or Sysplex since the CICS WebServer Plugin uses the EXCI communication mechanism.
- Only 32 KB of data in the HTTP datastream can be passed to or from the CICS program when using the CICS WebServer Plugin. This is because the EXCI uses a standard CICS COMMAREA on which the restriction of 32 KB applies.
- Security processing can be performed in the OS/390 Web server if using the CICS WebServer Plugin. Either HTTP basic authentication or SSL security can be configured.
- Data conversion is performed in the OS/390 Web server, not in CICS when using the OS/390 Web server.

For further information on using and configuring CWS with the CICS WebServer Plugin, refer to the following manuals:

- *CICS Transaction Server for OS/390 Version 1 Release 3: Web Support and 3270 Bridge* , SG24-5480
- *CICS Internet Guide* , SC34-5445

For information on configuring the OS/390 Web server, refer to:

- *IBM HTTP Server for OS/390 Release 7 Planning, Installing, and Using, Version 5.1* , SC31-8690

1.2.4 3270 Web bridge

The 3270 bridge feature of CICS Web support provides turnkey access to 3270 transactions from the Web. We will refer to this function as the 3270 Web bridge. To implement this solution, you need only reassemble your BMS mapsets and add CICS PROGRAM and TRANSACTION definitions. The resulting HTML is a GUI version of the original 3270 screen; this can be tailored if you want, but you do not need to. Most 3270 transactions will then run unchanged using this technique, though some applications may require modification. These restrictions are documented in chapter 8 of *Revealed! Architecting Web Access to CICS* , SG24-5466. The ease of implementationmakes the 3270 Web bridge

the preferred solution whenever Web access is required quickly, programming resources are limited, or the application has limited use or life expectancy.

The 3270 Web bridge can be used with either the direct connection to CICS or with the CICS WebServer Plugin. Figure 5 illustrates the data flow for a Web browser request using the facilities of the 3270 Web bridge and a CWS direct connection to access a CICS 3270 transaction.

Note that the 3270 bridge feature is only available when using CICS TS V1.2 or a later release.

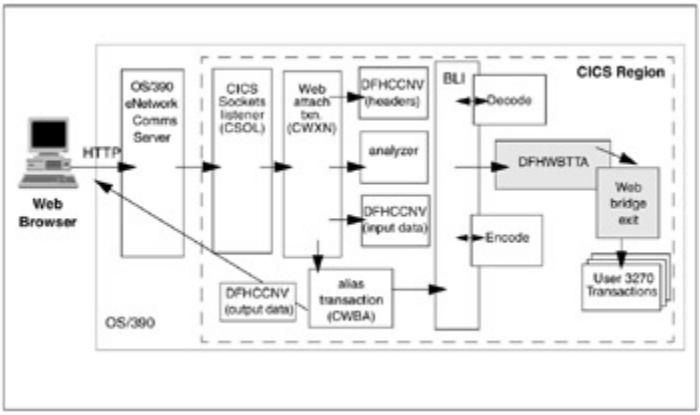


Figure 5: CICS Web support — 3270 Web bridge

The initial data flow is the same as that described in Figure 3 on page 8 for the description of CICS Web support and the BLI. However, instead of invoking the user program, the Web terminal translation program, DFHWBTTA, is invoked by the BLI. DFHWBTTA starts the transaction to be run in the 3270 bridge environment, where it runs in conjunction with the CICS provided Web bridge exit DFHWBLT. A summary of the components of the 3270 Web bridge follows.

DFHWBTTA

This is the Web terminal translation program, it initiates execution of the transaction under the 3270 bridge feature of CICS. DFHWBTTA formats the input in the COMMAREA to the form in which the 3270 transaction named in the Web user's input will expect it, attaches the transaction for execution under the bridge, and waits for it to complete.

DFHWBLT

This is the Web bridge exit and is used to control execution of the target transaction. When the 3270 transaction issues a 3270 RECEIVE, DFHWBLT supplies the input from the DFHWBTTA COMMAREA. When the transaction SENDs, it stores the output there. When the 3270 transaction running under the bridge ends, DFHWBLT notifies DFHWBTTA, which translates the 3270 output from the transaction to the HTML equivalent and then returns to the alias program. The alias now resumes standard CWI processing: It re-invokes the supplied converter program, this time to "encode" the output into HTTP/HTML, invokes DFHCCNV for conversion to ASCII and the proper code page, and returns the response to the Web browser.

Note there are several other sample bridge exits apart from DFHWBLT. These allow invocation from other environments, including MQ, TS, or TD queues, or a CICS Business Transaction Services (CBTS)

environment. Refer to the redbook: *CICS Transaction Server for OS/390 Version 1, Release 3: Web Support and 3270 Bridge* , SG24-5480, for further details.

State management

The program DFHWBST controls the state information required to manage 3270 pseudo-conversations when using the 3270 Web bridge. This information is used by DFHWBTTA and DFHWBLT.

Garbage collection

The program DFHWBGB is responsible for "garbage collection". It runs at an interval controlled by the SIT parameter WEBDELAY and purges state data associated with terminated 3270 Web transactions.

1.3 CICS Transaction Gateway

The CICS Transaction Gateway (CTG) is a set of server based software components that allows a Java program to invoke services in a CICS region. The Java program can be an applet, a servlet, or a custom Java application.

We describe the architecture of using the CTG with applets and servlets, but not applications, since they have no specific architecture.

The CICS Transaction Gateway is available for production use on OS/390, and on the following distributed platforms: AIX, Sun Solaris and Windows NT. It is also available for development use on Windows 95 and Windows 98. A high level summary of how a CICS application can be Web-enabled using the CTG is illustrated in Figure 6 .

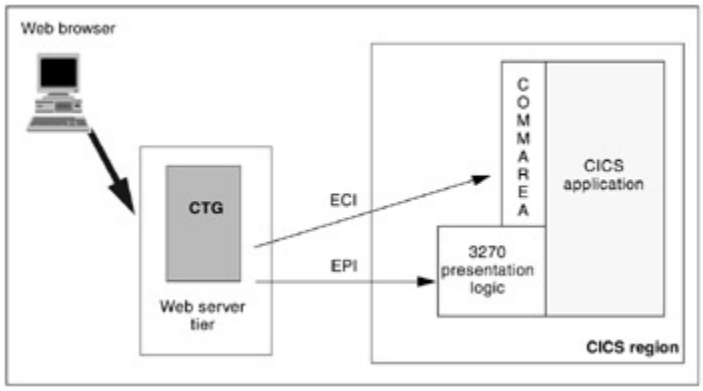


Figure 6: CICS Transaction Gateway

When the CICS Transaction Gateway for OS/390 V3.1 is used, it is supported with CICS TS V1.2 and V1.3. Note, however that if you wish to use the CTG V3.1 with CICS Transaction Server V1.2, the fix for APAR PQ31270 must be applied to CICS Transaction Server. This does not apply to CICS Transaction Server V1.3.

The OS/390 CICS Transaction Gateway, which is the subject of this performance study, consists of the following components:

Java gateway application

- This long-running process is used to accept CTG requests issued from remote Java applications such as applets.

Java class library

- This contains the following components
  - Basic Java methods

    These are used to set up connectivity to a CTG Gateway process or to invoke the underlying CICS Universal Client or OS/390 EXCI.

  - ECI Java methods

    These methods provide access to CICS COMMAREA based programs in a similar fashion to the CICS Universal Client ECI or the OS/390 EXCI.

  - Java beans

    These beans support development of applications from a number of Visual development environments such as Visual Age for Java.

Also, the OS/390 CTG uses the function of the CICS EXCI to communicate with the target CICS region. The function of the EXCI is used in the same way as the CICS Universal Client ECI would be used on a non-OS/390 platform.

In addition, the following components are available on non-OS/390 versions of the CTG:

CICS Universal Client

- The CICS Universal Client provides communication to the CICS server.

EPI Java methods

- These methods provide a Java API to manipulate CICS 3270 based transactions.

Terminal Servlet

- This supplied servlet dynamically converts 3270 output into HTML for display at a Web browser.

Apart from manually coding the CICS CTG Java methods, you can develop a CTG application using the IBM **Common Connector Framework** (CCF) Java Beans. We did not use the CCF in our CTG performance test application; however, IBM's CCF does provides the following:

- A common client programming model for connectors. These interfaces allow VisualAge for Java's Enterprise Access Builder (EAB) for transactions to easily build applets or servlets to access programs or transactions in a CICS region.

- A common infrastructure programming model for connectors, which gives a component

environment, such as WebSphere, a standard view of a connector, and vice versa

When developing an applet or servlet using the CCF CICS connector, the CICSConnectionSpec, CICSCommunication, and ECIInteractionSpec or EPIInteractionSpec classes are used. These classes can be specified in an EAB Command with an input and output (COMMAREA) to invoke a CICS program.

For more information on developing CTG applications using the CCF, refer to the redbook: *VisualAge for Java Enterprise Version 2: Data Access Beans - Servlets - CICS Connector* , SG24-5265.

For product information on using the OS/390 CTG, refer to *CICS Transaction Gateway for OS/390 Administration Version 3.1* , SC34-5528.

For information on configuring the CTG in different scenarios, refer to the redbook *Revealed! CICS Transaction Gateway with More CICS Clients Unmasked* , SG24-5277.

The following sections will now review the major components of the OS/390 CTG and how the architecture is different when using Java applets as opposed to Java servlets.

1.3.1 CICS Transaction Gateway applet architecture

Figure 7 shows an implementation of the CTG applet architecture on OS/390.

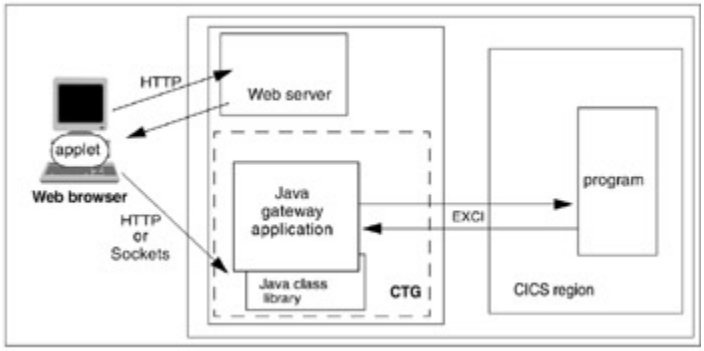


Figure 7: CICS Transaction Gateway applet architecture on OS/390

Web browser

This is a Java enabled Web browser. When a HTML page containing an applet tag is referenced, the applet is downloaded from the Web server. The applet Java methods are then executed in the Web browser JVM, and create a CTG network connection to the Java gateway application. This connection can be one of the following protocols: TCP/IP, HTTP, SSL, or HTTPS.

Web server

The Web server serves up the HTML page, which contains the applet tag for the CTG applet, and also serves this applet to the Web browser.

Java gateway application

This is a long running Java application that receives the remote ECI requests from the applet and, using

the Java Native Interface (JNI), invokes the EXCI to pass the ECI request to the CICS program.

Java class library

These Java methods are used by the applet to open a connection to the Java gateway application, and by the Java gateway application to flow the ECI request to the CICS region.

1.3.2 CICS Transaction Gateway servlet architecture

Figure 8 illustrates the CTG servlet architecture on OS/390.

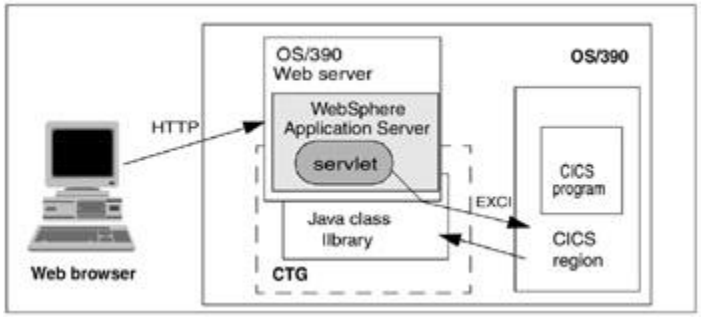


Figure 8: CICS Transaction Gateway servlet architecture on OS/390

Web browser

This is a standard Web browser that can send HTTP requests.

WebSphere Application Server

WebSphere Application Server for OS/390 provides both the OS/390 Web server (IBM HTTP Server) and the servlet engine. The servlet runs within the JVM of the servlet engine, just as an applet runs within a Web browser.

The servlet

The servlet is written using the CTG Java methods and is compiled and deployed ahead of time. It is invoked by a request from the Web browser using either a URL, a HTML FORM, or a HTML server-side include. The servlet uses used the CTG *local:* protocol to invoke the CICS EXCI libraries using the Java Native Interface (JNI). The CTG ECI methods use the EXCI to invoke the CICS program, passing the COMMAREA as input.

Java class library

This Java library contains a set of methods used by the servlet to invoke the EXCI using the facilities of the JNI. The CTG Java methods are invoked within the servlet, and ECI requests are sent from the servlet to the CICS region.

Note that no CTG Java gateway application is usually required in the servlet configuration. The Java gateway application is only required when the CTG Java methods are executed in a JVM remote from where the CTG is installed, as is the case with the applet architecture.

Chapter 2: Performance and capacity planning factors

Overview

In this chapter we will discuss the various system components which contribute to performance bottlenecks. We will also discuss ways to reduce some of these bottlenecks, and tell you where additional information can be found.

What is a performance bottleneck?

A performance bottleneck is the component in a computer environment causing the highest level of contention. This can be a system resource like CPU, memory, disk, the network, the client machine, or the application. There is always a bottleneck, because some component will always be the slowest. The question is whether this bottleneck is a problem to your application.

How to determine a performance bottleneck

There are some general guidelines which should be followed when identifying a performance bottleneck. A standard approach should be used when determining where the performance bottleneck exists. Reviewing CPU, memory, disk I/O, and network I/O, as represented in Figure 9 , is a recommended approach.

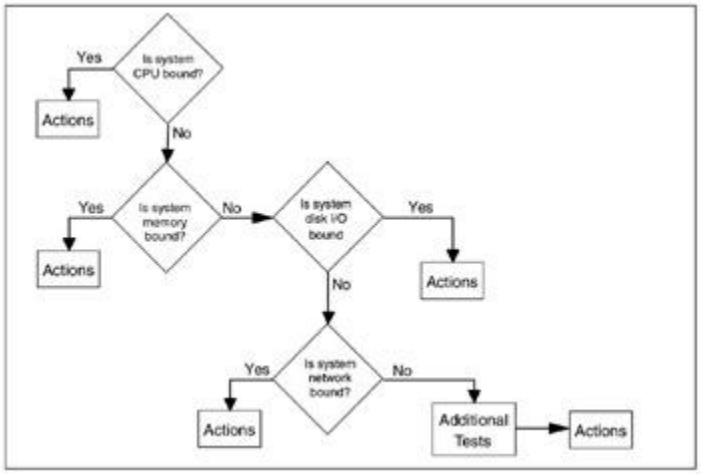


Figure 9: Performance flowchart

Monitoring performance

When attempting to isolate a performance bottleneck, the importance of collecting and analyzing performance data should not be overlooked. Attempting to resolve a performance bottleneck without

actual performance data can lead to making incorrect inferences about the source of a performance bottleneck. Changing the configuration may have no impact on a performance bottleneck without first understanding the source of the bottleneck. Using data collected by RMF, CICS statistics, Tivoli Performance Monitor, and other tools will provide a set of concrete data to be used when isolating a performance bottleneck.

Capturing performance data will also help to measure the affect of configuration changes on performance. Without this data you will not be able to accurately assess the success of performance tuning.

2.1 Hardware components influencing performance

In this section we will focus on the hardware components of a computer system which contribute to performance bottlenecks. Section 2.2 , "Software components influencing performance" on page 26 will address the software components of performance bottlenecks.

2.1.1 CPU

The CPU capacity that is available for use by CICS will have an impact on the performance of a CICS region. In the sections ahead, we discuss some of the impactors to CPU performance.

Number of engines

In its simplest configuration, a central electronic complex (CEC) consists of a single processor (also referred to as a CPU or engine). As workload increases, additional processors may be added to a CEC. In order to take advantage of multiple processors, it must be possible for the workload to be divided into concurrent activities. How CICS TS V1.3 enables the use of multiple processors is discussed in 2.2.1 , "CICS Transaction Server for OS/390" on page 26 later in this chapter.

Cryptographic Coprocessor feature

The Cryptographic Coprocessor feature is a hardware feature available on S/390 processors. It consists of dual cryptographic module chips protected by tamper-detection circuitry and a cryptographic battery unit. It can be used to off-load CPU processing from the main CEC processors when performing cryptographic operations, and as such, can provide a significant reduction in CPU usage for both SSL handshakes and SSL data transmissions. See Chapter 6 , "SSL with CWS" on page 85 for further details.

Total workload

In addition to the CICS workload, the workload of the entire CEC must be taken into consideration when reviewing performance bottlenecks. The reason for a CICS performance bottleneck may be that some other workload is using a higher than anticipated percentage of resources, thus limiting the resources available for the CICS workload. RMF reports will indicate what system tasks are responsible for use of excessive resources.

2.1.2 Storage

Not having enough storage available to a CICS region will result in high paging rates and often short-on-storage (SOS) conditions, both of which can cause slow response times. Some of these issues are

discussed here; additional information can be found in the *CICS Performance Guide* , SC33-1699.

Paging

Paging occurs when there is not sufficient central storage to support all requests for storage within the system. Performance monitor data should be reviewed to determine the paging rate for the CICS region. A paging rate of less than one page-in per second from direct access storage device (DASD) is to be preferred.

Paging between central storage and expanded storage is a fraction of the cost of paging to auxiliary storage (DASD). The page-in operation is more costly than that of a page-out. A page-in operation is processed synchronously by OS/390, which will temporarily halt any other CICS activity within the region, while the requested data is loaded into central storage.

The maximum number of CICS tasks can affect the amount of storage that a CICS region is requesting. The maximum task (MXT) system initialization table (SIT) parameter controls the number of tasks within a CICS region. If paging is a performance bottleneck, a review of the CICS statistics will show the value for MXT and the number of times that a CICS region has reached maximum tasks. By reducing MXT, the demand for storage is less, which may in turn reduce the amount of paging and may increase throughput due to the reduced paging rate.

The *CICS Performance Guide* , SC33-1699, discusses the impact of paging on CICS performance.

CICS storage

CWS uses temporary storage queues to store inbound HTTP requests and outbound responses built by the WEB API. A sample temporary storage queue definition exists in the DFHWEB group. The supplied sample uses main temporary storage in order to reduce the amount of I/O to auxiliary storage. You should consider the placement of temporary storage based upon the storage available to your CICS region.

The Transaction Isolation (TRANISO) SIT parameter also impacts how CICS allocates storage. A CICS region running with TRANSIO=YES will allocate user requested storage above the 16 MB line in 1 MB blocks. This means the amount of storage requested by a CICS region may be very large. This will impact the amount storage requested by a CICS region.

The LE runtime options that are in effect also impact how storage is allocated within a CICS region. LE options such as ALL31 and STACK can have a dramatic impact on storage requirements. For a detailed description of LE options, see *OS/390 LE Installation and Customization Guide* , SC26-4817. The RUWAPOOL, SIT parameter will also impact how CICS allocates LE storage. Additional information on the impact of LE options and RUWAPOOL are found in the *CICS Performance Guide* , SC33-1699.

2.1.3 Disk I/O

Disk I/O can contribute to performance bottlenecks due to the longer response times involved in accessing data from disk than from memory. Ensuring that disk I/O traffic is optimized minimizes the amount of time spent waiting for DASD operations to complete. In addition to monitoring and tuning DASD performance, the use of advanced storage technology, such as IBM's Enterprise Storage Server offer greater levels of performance and scalability which aid in eliminating DISK I/O as a performance bottleneck. Information about the latest storage technology available can be found in:

- http://www.storage.ibm.com

Data set management

The goal of data set management is to minimize DASD operations. When using CICS with VSAM, this can be accomplished by increasing the number of VSAM hiperspace buffers, and by the use of CICS Data Tables, both of which will minimize disk I/O. By minimizing VSAM CI/CA splits and performing DASD subsystem tuning, other DASD operations will be minimized. Ensuring that DASD operations such as channel busy, device busy, and seek times are at appropriate values for your DASD subsystem helps reduce response times. The *CICS Performance Guide* , SC33-1699, has guidelines for DASD tuning.

2.1.4 Network I/O

In the case of Web-enabled CICS transactions, you have two possible classes of network I/O to consider: your private network, and the public Internet.

Your private network

This term *private network* may mean different things, depending on your specific network configuration. It may be an intranet, an extranet, or some other network configuration. But, it is a network over which you have some direct control. Points of potential performance bottlenecks within your network include the CEC network adapter, bridges, routers, and other components within the network infrastructure.

Network adapter

Ensuring that your CEC has ample bandwidth to connect to the network is an important point to review in order to avoid this performance bottleneck. Sizing of the Open Systems Adapter (OSA) should be performed to ensure that adequate bandwidth exists. The OSA is a hardware feature which provides direct connection from a CEC to the LAN. OSA supports a variety of network topologies at different speeds. Refer to *Planning for the System/390 Open Systems Adapter Feature* , GC23-3870, for additional information on the setup of OSA cards.

Network infrastructure

The components of your network are also potential bottlenecks. Ensuring that there is adequate capacity for your CEC and the clients on your network will eliminate this as a bottleneck. The importance of the network to your total performance cannot be stated strongly enough. As you migrate towards a Web-centric environment, away from a 3270 green screen environment, you need to model the impact to the network infrastructure. The 3270 data streams on the network are less traffic-intensive than the Web-based network traffic. Network performance is a unique experience, and every network may behave differently under similar loads. By using tools such as Tivoli, you can determine if some component within your network is a performance bottleneck. You then can develop a plan to address this bottleneck.

The public Internet

It is also possible that the Web-enabled CICS application that you have built is accessed across the *public Internet* as a part of your company's e-business strategy. If this is the case, then you may not have much control over the overall performance of your customer's connection to your application. This does

not, however, remove the Internet as a potential bottleneck.

Your Internet service provider should be able to address your specific needs. For our purposes, we will be assuming that you have sufficient capacity in order to eliminate the Internet as a performance bottleneck.

2.1.5 Client configuration

While the sizing of the client machine is beyond the scope of this redbook, it is a notable part of the total application architecture and should not be overlooked. When isolating performance bottlenecks, the same issues apply to the client machine as are discussed in this chapter. Depending on which approach you use to Web-enable access to CICS, the role of the client machine and the processing requirements for the client machine will vary. A client that is used primarily as a Web browser will have lighter processing demands than a client machine that will be running Java applets. Understanding the role of the client machine within your application architecture will help you to eliminate the client machine as a performance bottleneck.

2.2 Software components influencing performance

In addition to the hardware components which influence performance, as discussed in the last section, software also has an impact on performance.

As part of ensuring that your CICS system and other software is running optimally, you should attempt to keep current on maintenance. While it may not always be possible to be running the latest release of maintenance, there are often significant performance enhancements available through maintenance.

2.2.1 CICS Transaction Server for OS/390

CICS Web support (CWS) is a fully integrated service within CICS TS V1.3. CICS TS V1.3 has expanded the CICS domain structure to include a separate domain for CICS TCP/IP socket requests. CICS TS V1.3 has also extended the API to provide the WEB API and DOCUMENT API, to support the CWS and HTML template processing. See the *CICS Transaction Server, Migration Guide* , GC34-5353 for more information.

Multiple TCBs

Since CICS TS V1.3 has a multi-domain design, it dispatches multiple TCBs, and thus it is able to concurrently utilize multiple processors in a multi- processor CEC. The business logic, however, still all runs within the QR TCB, and therefore is not dispatched across multiple processors. Further details on CICS usage of multiple TCBs and how to use CICS dispatcher statistics to analyze TCB usage is given in 8.3 , "Using too much CPU" on page 146 .

2.2.2 The OS/390 Web server

Tuning the OS/390 Web server for peak performance involves reviewing OS/390 UNIX System Services tuning guidelines. Reducing program loads can be done by ensuring that the UNIX service modules are in LPA and that Web server libraries are in the linklist or LPA. Thus, you will reduce response times for Web requests. Performance can also be improved by using the CacheLocalFile directive or Fast Response Cache Accelerator to pre-load frequently referenced Web pages, or by using

the LE runtime option HEAPPOOLS(ON). It is also important that you do not start the OS/390 Web server from a UNIX shell, otherwise the performance specifications of the UNIX shell will be inherited by the Web server address space.

It is possible to run the OS/390 Web server in three modes:

- Standalone mode is best suited for test environments and a small number of connections.
- Scalable mode works in conjunction with WLM, which can improve OS/390 Web server performance by dynamically dividing work between multiple queues based on WLM settings.
- Multiple mode allows you to run multiple instances of the OS/390 Web Server on different ports.

Refer to http://www.s390.ibm.com/oe/bpxa1tun.html and *IBM HTTP Server for OS/390 Release 7 Planning, Installing, and Using* , SC31-8690, for details about performance tuning the OS/390 Web server.

2.2.3 eNetwork Communications Server

Each subsequent release of eNetwork Communications Server has had significant performance enhancements over the prior release. You should review your current version and maintenance level of TCP/IP before you begin to implement Web-enabled CICS applications. In the latest releases of eNetwork Communications Server, TCP/IP shares services with VTAM, such that CPU time for TCP/IP requests is charged to both the TCP/IP and VTAM address spaces.

Reviewing the *eNetwork CS IP Configuration* , SC31-8513, to ensure that you have selected appropriate TCP/IP tuning parameters, is also important. In the most recent version of TCP/IP, the number of tuning parameters is reduced. Setting the TCP/IP TCPSENDBfrsize and TCPRCVBufrsize to appropriate sizes for the largest data size that you expect to should be reviewed.

The SOMAXCONN parameter controls the number of concurrent connections. This parameter should be reviewed to ensure that it is in line with CICS parameters, such as TCPIPSERVICE BACKLOG.

2.2.4 HTTP

CICS TS V1.3 supports HTTP 1.0 requests and responses and the HTTP 1.0 KeepAlive extension, which offers persistent HTTP connections. Chapter 5 , "CWS with Web-aware presentation logic" on page 65 discusses the affect of using persistent connections. Unpredictable results may occur if you use HTTP 1.1 specific headers.

CICS TS V1.3 has extended the EXEC CICS API to include a new set of WEB commands. These commands are used in CWS Web-aware programs to send, receive, and manipulate HTTP data within a CICS application. The redbook *CICS Transaction Server for OS/390 Version 1, Release 3: Web Support and 3270 Bridge* , SG24-5480, further describes the WEB API and support of HTTP.

2.2.5 Java

It is important that your system is running at the highest maintenance levels available for Java and related Java support. Java support is a rapidly evolving area, and remaining current on support will provide the most efficient performance. LE runtime options can also have a major impact on storage

usage and CPU utilization. The key LE runtime options for Java are STACK, HEAP, and ANYHEAP. Setting the values for these options too small may cause additional GETMAIN of storage, which will also increase CPU consumption. Setting LE runtime options too large will allocate an excessive amount of storage, which may result in SOS conditions. Chapter 7 , "The OS/390 CTG" on page 103 in this redbook, and the *CICS Performance Guide* , SC33-1699, describe Java performance considerations in greater detail.

2.2.6 Client configuration

While the specifics of the client environment are not addressed in this redbook, you should understand what impact your particular client configuration has on performance. Not all software configurations will behave the same way under all circumstances. Be aware of the releases of software and what the impact of migrating from one release to another has on the performance of your application architecture. It is also possible that you will not have complete control over all aspects of the client configuration. Products such as Tivoli Performance Monitor can help with maintaining client configurations.

2.3 Workload management

Once you have set performance goals, Workload Management (WLM) works automatically to maintain those goals. The manuals, *MVS Planning: Workload Management* , GC28-1761, and *CICSPlex: SM Concepts and Planning* , GC33-0786, discuss setting up WLM in detail.

The following benefits are gained through the use of WLM:

- Improved performance through the use of MVS resource management
- Simplified MVS tuning
- The ability to integrate workload balancing for terminal-initiated transactions, non-terminal-initiated transactions, External CICS Interface (EXCI) clients, CICS clients, CICS Web support, CICS Transaction Gateway, IIOP, and started tasks
- The ability to integrate CICS Business Transaction Services processes and activities fully into the workload separation and workload balancing functions
- Optimum performance and response times for a variable and unpredictable workload
- Work routed away from a failing target region to an active target region
- Opportunities for increased throughput and improved performance
- Reduced risk of bottlenecks
- Less operator intervention

2.3.1 OS/390 Sysplex environment

The OS/390 Sysplex environment enables parallel processing, which allows processing on multiple S/390 CECs to occur concurrently.

2.3.2 Workload balancing

2.3.2.1 TCP/IP port sharing

TCP/IP port sharing provides a simple way of spreading workload over multiple CICS regions in one CEC by allowing multiple CICS regions to listen on the same TCP/IP port number.

The TCPIPSERVICE CICS resource definition controls which port a CICS region will listen for incoming requests; this is further described in the *CICS Resource Definition Guide* , SC33-1684-02.

The SHAREPORT parameter of the PORT TCP/IP configuration statement is used to define the names of all of the CICS regions which may listen on a particular port. TCP/IP port sharing requires eNetwork Communications Server in OS/390 Version 2 Release 5 or later. For more information, see *OS/390 eNetworks Communications Server: IP Configuration* , SC31-8513.

2.3.2.2 Dynamic DNS

With dynamic domain name server (DNS), multiple CICS systems are started to listen for requests on the same port, using Virtual IP addresses. The host name in the request is resolved to an IP address by MVS DNS and WLM services. By using dynamic DNS you are able to spread incoming requests across multiple CICS regions that are running anywhere within a sysplex.

Implementing dynamic DNS is discussed in *OS/390 V2R7.0 eNetwork Communications Server IP Configuration* , SC31-8513.

2.3.2.3 SecureWay Network Dispatcher

IBM SecureWay Network Dispatcher manages TCP/IP traffic by allowing you to balance the load across servers of different sizes and different operating systems. The Web site: http://www.ibm.com/software/network/dispatcher/ has additional information about the use of the SecureWay Network Dispatcher.

2.3.2.4 CICSPlex System Manager

CICS TS V1.3 provides extensions to CICSPlex System Manager (CPSM) which supports the dynamic routing of requests for:

- CICS Web support (CWS)
- CICS Transaction Gateway (CTG)
- External CICS interface (EXCI) client programs
- Any CICS client workstation product using External Call Interface (ECI)
- Internet Inter-Object Request Block Protocol (IIOP)
- Any function that issues a CICS LINK request

Dynamic routing provides the ability to balance a workload among multiple CICS regions. *CICSPlex:*

SM Managing Workloads , SC33-1807, describes the implementation of workload balancing using CPSM.

2.4 Capacity planning

Capacity planning is an ongoing activity. Review of performance data and an understanding of the affect of changes to the environment need to be understood so that new workloads can be modeled and their impact on the current environment understood before implementation in a *production* environment. Changing how a business process is performed may stress available system capacity beyond available limits. Capacity planning involves the review of performance data from many disciplines, OS/390, DASD management, network administration, application design, CICS, and other platforms. The redbook *OS/390 MVS Parallel Sysplex Capacity Planning* , SG24-4680, and the *CICS Performance Guide* , SC33-1699, discuss capacity planning in detail.

2.4.1 LSPR ratios

IBM markets a large range of computers, now more usually known as Central Electronic Complexes (CECs), with widely differing processing capacities or "powers". Performing a capacity planning exercise often involves the need to translate estimated or measured performance values from one model of CEC to another. For example, a customer upgrading his machine to a larger model will often want to estimate the cost of running an existing application on the new CEC, and will base this estimate on in-house measured costs on his current machine, and then project or translate them to the proposed new machine. Similarly, a customer adding a new application to an existing machine may have to base his capacity planning estimate on available IBM performance data measured on a different model than his current CEC, and needs a way of coping with the differences in machine in the estimation process.

To facilitate this translation between CEC models, IBM provides the Large System Performance Reference (LSPR) tables. These are accessible on the Internet at http://www.s390.ibm.com/lspr/lspr.html . The tables are updated at regular intervals, and cover IBM, Amdahl, and HDS machines, and OS/390, VM, and VSE operating systems.

The LSPR method, and the tables based on it, operate in the following manner. One CEC model is defined as the LSPR reference or base machine in performance terms. This base machine is rarely changed, and you can expect the same base machine to be used for quite a few years. The base machine is currently defined as the IBM 9672-R15, which is a single processor air cooled machine based on CMOS technology.

IBM has defined, for LSPR purposes, several separate workloads based around each of their principle mainframe software products. For example, there is a typical IMS workload, and a typical TSO based workload, and, of most interest for our purposes, a typical CICS/DB2 workload. Each of these workloads is run on every machine in the LSPR tables, and, in simple terms, a measurement of the amount of CEC processing time required to run each workload is made for each CEC model.

These CEC processing times are then compared to the cost of running the same workload on the base 9672-R15 CEC, and the comparisons are presented in the LSPR tables as a series of indices or ratios. These ratios are in essence, for a given workload, an indication of the relative processing power of the particular CEC, and are a measure of the rate at which it can execute machine instructions, compared with the LSPR base CEC. The LSPR base CEC always takes the ratio value 1.0 for each workload, and all the LSPR table values for all the other CECs are relative to this. A ratio value of greater than 1.0 indicates a more powerful CEC than the base 9672 R15, and a ratio of less than 1.0 indicates a less

powerful CEC. A selected range of the LSPR ratios for the defined CICS/DB2 workload are shown below in Table 1 . Note that the number of processors that a particular CEC model has is given within the LSPR tables in the column marked # *CP* .

Table 1: Selected LSPR ratios for CICS

Processor Model	# CP	CICS/DB2 LSPR
9672-R15	1	1.00
9672-R25	2	1.81
9672-R35	3	2.58
9672-R45	4	3.30
9672-R55	5	4.22
9672-R65	6	4.88
9672-R75	7	5.48
9672-R16	1	2.03

So, for example, the tables indicate that, for the CICS/DB2 workload, the LSPR ratio for a 9672-R25 is 1.81, indicating that the 9672-R25 is a more powerful CEC than the LSPR base machine, the 9672-R15. This is because the 9762-R25 has two processors and the 9672-R15 has one. Thus, theoretically, the R25 is capable of executing twice as many machine instructions per unit time as the R15. However, you will note that for the R25, the LSPR ratio is 1.81 and not 2.0; this is because of a decrease in efficiency involved in the very nature of multi-processing. In general, this reduction in efficiency increases as the number of processors in a CEC increases, and this is reflected in the LSPR table values. So, for the 9672-R55, which is the five processor version of the same CEC series, the CICS/DB2 workload LSPR ration is 4.22 as opposed to 5.

2.4.2 CPU speed considerations

However, when capacity planning with CICS, you must also consider the speed of the individual CPUs used in your CEC. This is because CICS still makes extensive use of a specific TCB, the QR TCB, and it may be that your CICS system is reaching maximum capacity of that TCB, thus limiting your maximum CICS CPU utilization to just one CPU in the CEC. More details on how to do this is given in 8.3 , "Using too much CPU" on page 146 .

To increase the capacity of a single CICS region in this condition, it would be necessary to move to a CEC with a more powerful CPU (for instance, moving from a 9672-R55 to a 9672-R56). Moving to a CEC with more processors, such as from a 9672-R55 to a 9672-R65, may give greater total computational power, but this does not provide a higher individual CPU speed, which would be the limiting performance factor for the CICS region in this situation.

Part 2: Performance analysis

Chapter List

Chapter 3: The 3270 green screen Trader application

Overview

In this chapter we describe the application that will be used in the capacity planning studies presented in subsequent chapters. Like the majority of applications used on CICS systems today, it is written in COBOL and uses the 3270 Basic Mapping Support (BMS) interface of CICS to provide a menu-based user interface for 3270 devices. Such applications are often referred to as legacy applications. The program design employed in such legacy applications is often hierarchical, navigating through levels of menus. Because they were designed to run on monochrome (green characters on a black background) 3270 devices, they were commonly referred to as "green screen" applications.

The huge numbers of CICS COBOL applications developed to run on 3270 devices produced a wide variety of program structures and programming styles. Very often these programs contain a mixture of business logic and 3270 BMS presentation logic. It has been a recommended approach for some time to separate business and presentation logic, particularly because applications developed in this way can be readily used in a client/server environment. It also makes it simpler to extend such applications to exploit access from the Web.

3.1 Introducing the Trader application

Our sample green screen application is called Trader. Trader allows authenticated users to trade shares, that is to buy and sell shares in a given group of companies, as well as obtaining real-time quotes on the value of their current holdings. Trader has been developed as a sample as part of an IBM CICS Web-enablement service offering. Sample code and templates required to Web-enable the Trader using all of the technologies documented in this redbook are available as additional materials from the ITSO Internet site http://www.redbooks.ibm.com . We will be using Trader as our sample application throughout this redbook for our CICS Web-enablement performance study and capacity planning exercises.

Trader is written in COBOL. It uses the VSAM access method for file access and the CICS 3270 BMS programming interface. It is a pseudo-conversational application, meaning that a chain of related non-conversational CICS transactions is used to convey the impression of a "conversation" to the user as he goes through a sequence of screens that constitute a "business transaction". A non-conversational CICS transaction has one input and one output, so no task waits for user input as the user examines a screen and enters responses into it. CICS provides several facilities for passing information about the current state of the business transaction forward from one task to another. The most commonly used is the COMMAREA data structure which can be associated with the terminal.

At each step the application presents a set of options. The user makes a choice, then presses the required key in order to send their selections back to the application. The application performs the necessary

actions based on the user's choice and presents the results together with any possible new options. The application has a strict hierarchical menu structure which allows the user to return to the previous step by using the PF3 key. The application consists of two modules TRADERPL, which contains the 3270 presentation logic, and TRADERBL, which contains the business logic.

3.1.1 Basic application structure

Figure 10 shows a summary of the flow of CICS tasks for our chosen "business transaction" to perform a simple stock update operation. For the ten steps indicated, the following ten separate CICS tasks will run:

1. The initial CICS transaction identifier (TRAD) is entered; this invokes the TRADERPL program, which calls TRADERBL to build a list of companies for use in the next step. TRADERPL returns the signon display.

2. A userid and password is entered and verified. TRADERPL then returns the company selection display.

3. A company is selected, and TRADERPL returns the main options display.

4. Option 1 for a *New Real-Time Quote* is entered. TRADERPL calls TRADERBL, which reads the company and customer files. TRADERPL then returns the real-time quote display.

5. PF3 is pressed to exit back to the main options display, invoking only TRADERPL to send that display.

6. Option 2 for *Buy Shares* is entered, and TRADERPL is invoked, which returns the buy shares display.

7. The number of shares to purchase is entered. TRADERPL calls TRADERBL which reads the company file and updates the stock holding in the customer file. TRADERPL then returns the main *Options* display.

8. Option 1 for a *New Real-Time Quote* is entered (as in step 4). TRADERPL calls TRADERBL, which reads the company and customer files. TRADERPL then returns the real-time quote display.

9. PF3 is pressed to exit back to the main options display.

10. PF12 is pressed; the application terminates by TRADERPL sending a final SEND TEXT message to the screen on completion.

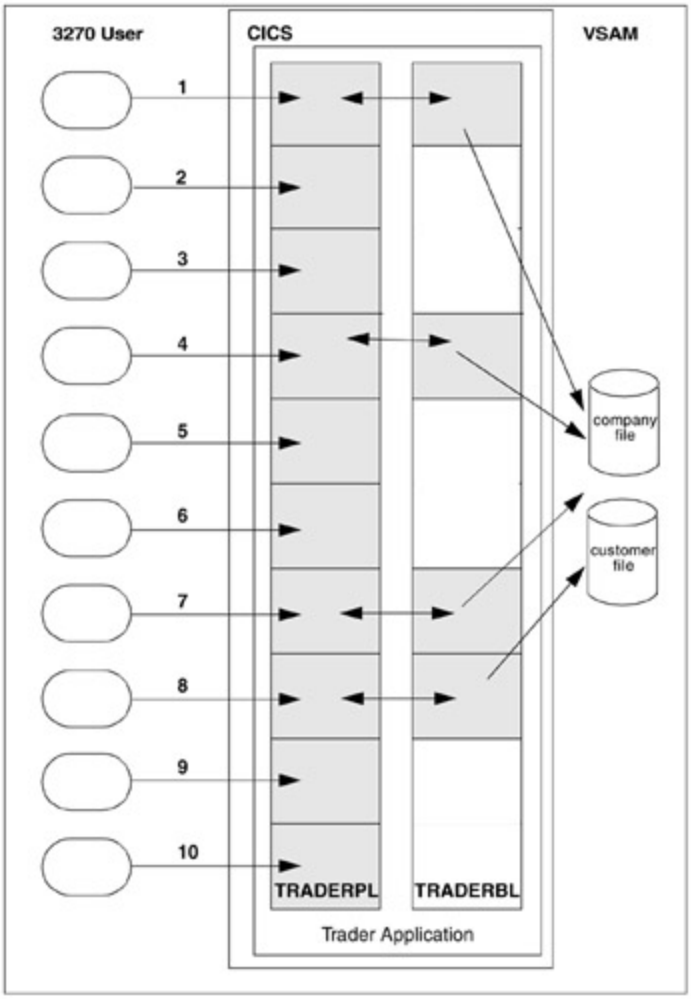


Figure 10: 3270 Trader application summary

3.1.1.1 Detailed application flow

In this section we describe the Trader application in more detail:

1. The program TRADERPL is invoked on a 3270 capable terminal by entering the initial CICS transaction identifier (TRAD). TRADERPL calls TRADERBL, passing an inter-program COMMAREA of 400 bytes. TRADERBL expects the COMMAREA to contain a request type and associated data. There are 3 request types: *Get_Company* to return a company list, *Share_Value* to return a list of share values, or *Buy_Sell* to buy or sell shares. In this step the request type is *Get_Company* .

   When TRADERBL receives a *Get_Company* request, it browses the company file and returns the first four entries to TRADERPL. At this point the user has not entered any request, but the application assumes that a Get_Company request will be following. TRADERPL then sends the *signon display* (T001 shown in Figure 11), which prompts for a userid and password. The list of companies is stored in the COMMAREA associated with the terminal when the TRAD transaction ends, so that it will be available at the next task in the pseudo-conversational sequence.

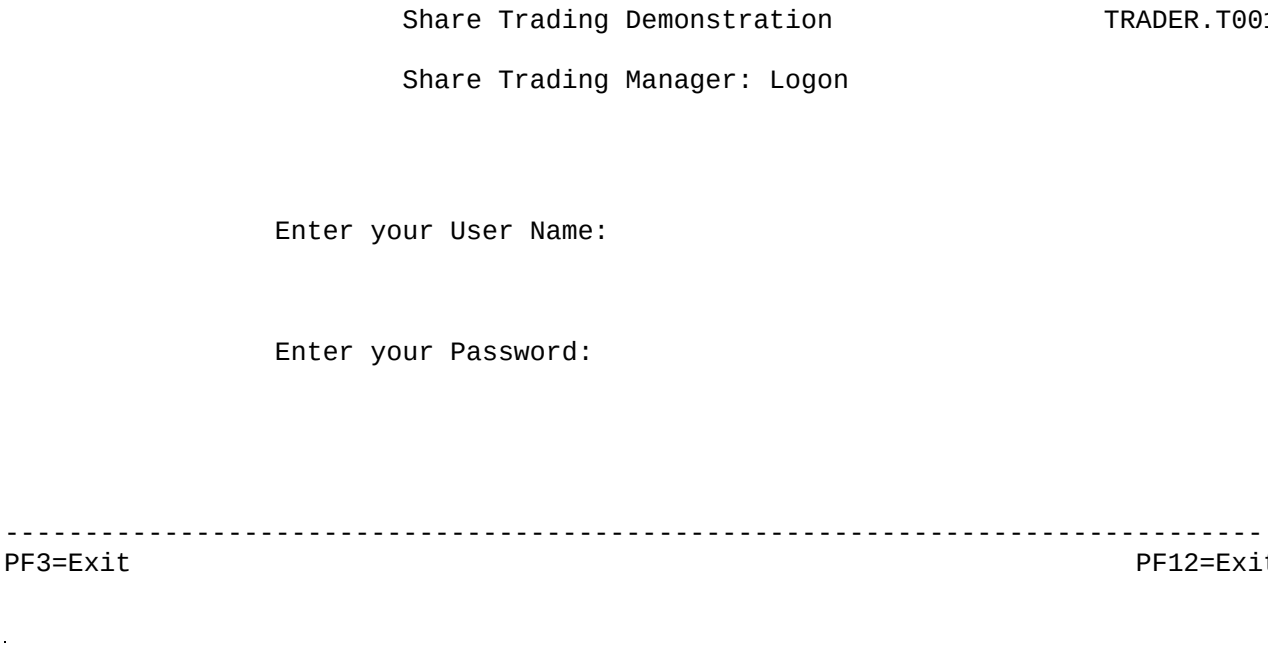


Figure 11: Trader signon display

2. The next transaction invokes TRADERPL, which receives the *signon display* (T001) and the saved COMMAREA from step 1. Using the company data acquired in step 1, TRADERPL sends the *company selection display* (T002, shown in Figure 12), the format of which is shown in Figure 12 . TRADERPL then returns, specifying the next transaction to run and the associated COMMAREA.

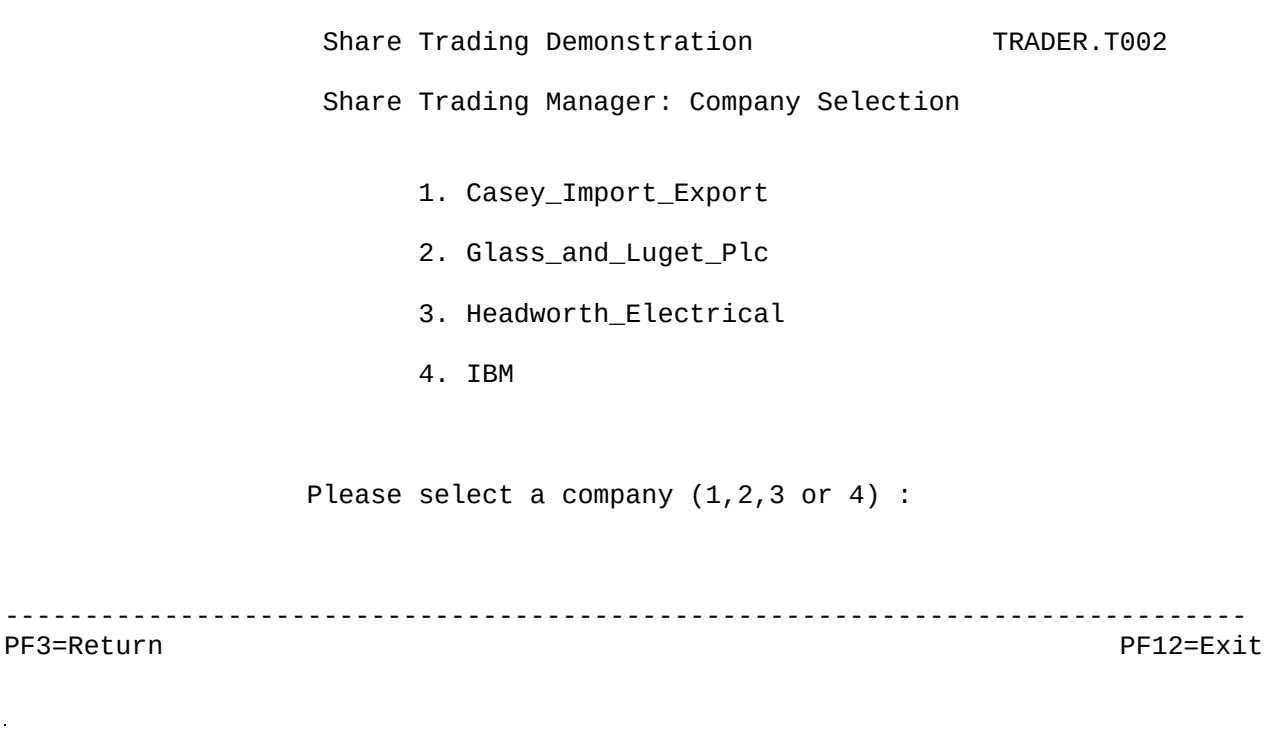


Figure 12: Company selection display

3. The user selects the company to trade from the *Company Selection* display, and presses Enter. The program TRADERPL is invoked and sends the *Options display* (T003, shown in Figure 13) to the terminal. The user can now decide whether to buy, sell, or get a new real-time quote. TRADERPL returns, specifying the next transaction to run and the associated COMMAREA.

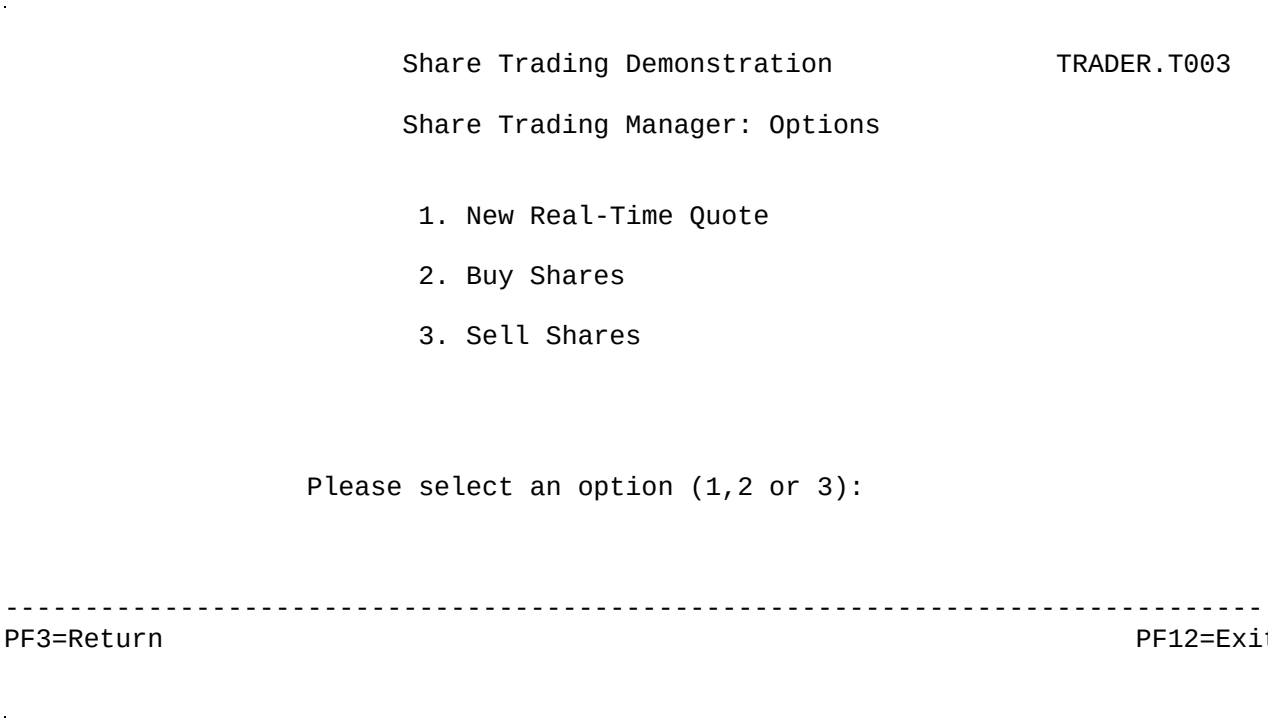
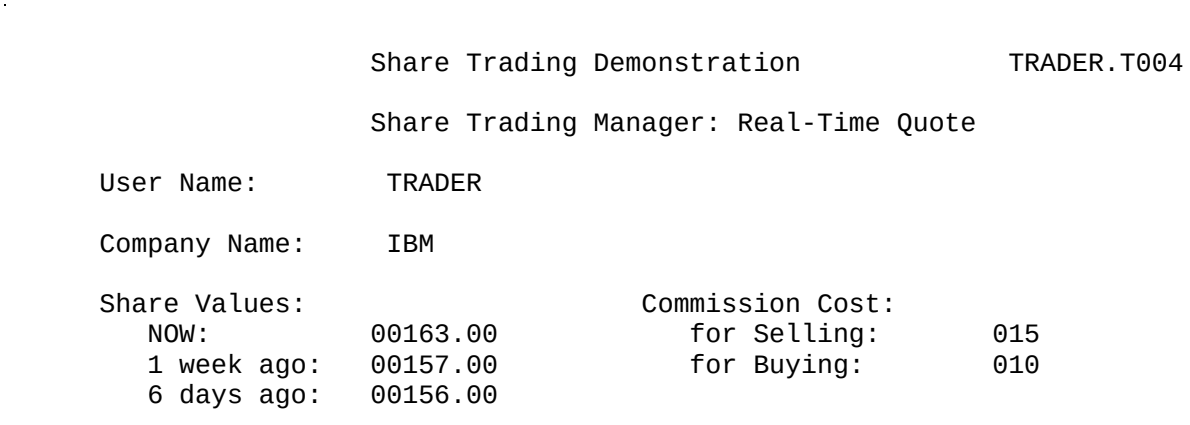


Figure 13: Options menu display

4. In the flow of our business transaction, the user then selects *Option 1* and presses Enter. TRADERPL is invoked and determines that the user's request is a *Share_Value* request type. TRADERPL calls TRADERBL, passing the request type and the company selected earlier. TRADERBL reads the customer file to determine how many shares are held, then reads the company file to determine the price history, and returns the information to TRADERPL. TRADERPL uses this data to build a *Real-Time Quote* display (T004) as illustrated in Figure 14 . This display shows the recent history of share values for the company chosen, the number of shares held with this company, and the total value of these shares. TRADERPL returns, specifying the next transaction to run and the associated COMMAREA data.



```
          5 days ago:   00159.00
          4 days ago:   00161.00
          3 days ago:   00160.00
          2 days ago:   00162.00         Number of Shares Held: 0100
          1 day ago:    00163.00         Value of Shares Held:  000000000.00


--------------------------------------------------------------------------------
PF3=Return                                                              PF12=Exi
```

Figure 14: Real-time quote display

5. The user now presses PF3 to go back to the *options menu* . TRADERPL is invoked and sends the *Options display* (T003) to the terminal (repeating the actions of step 3) and returns, specifying the next transaction to run and the associated COMMAREA data.

6. The user now requires to purchase shares, so selects *option 2* and presses the Enter key. Program TRADERPL receives map T003 and determines that the user wants to buy shares, and sends the *Shares-Buy* display (T005) shown in Figure 15 . TRADERPL returns, specifying the next transaction to run and the associated COMMAREA.

```
                     Share Trading Demonstration              TRADER.T005

                     Share Trading Manager: Shares - Buy


                User Name:     TRADER

                Company Name:  IBM


                Number of Shares to Buy:  100




--------------------------------------------------------------------------------
PF3=Return                                                              PF12=Exi
```

Figure 15: Shares — Buy display

7. Program TRADERPL receives the T005 screen and builds a *Buy_Sell* request COMMAREA which is passed to program TRADERBL. TRADERBL reads the company file and then performs a READ for UPDATE and REWRITE to the customer file to update the customers share holdings. The success of the request is returned to TRADERPL in the COMMAREA, and TRADERPL sends the *Options display* (T003) reporting the successful buy to the user. TRADERPL returns, specifying the next transaction to run and the associated COMMAREA.

8. Next the user checks his shareholdings by repeating step 4.

9. The user returns to the options screen by repeating step 5.

10. The business transaction is completed by the user pressing PF12, which performs a SEND TEXT to write a message to the terminal reporting the session is complete. TRADERPL then executes the final RETURN command. No COMMAREA is specified because the pseudo-conversation is over and there is no conversation state data to retain.

3.1.2 Application characteristics influencing performance

Let us now look at the different characteristics of the Trader application influencing performance. The Trader application is modular and well structured, in that the presentation logic (3270 and BMS commands) is in a separate module to the business logic. Thus we can examine the factors influencing presentation logic costs and business logic costs separately.

3.1.2.1 Presentation logic

These are the factors that will affect CPU usage in the presentation logic:

Number of network I/O operations

- The number of network I/O operations is related to the number of SENDs and RECEIVEs (both BMS and native 3270 commands). For our particular business transaction sequence illustrated in Figure 11 on page 40 , we have nine pairs of BMS maps received and sent, and a BMS RECEIVE with a 3270 SEND in the final transaction. These costs will be partly incurred in CICS and partly in VTAM.

State management

- Running multiple pseudo-conversational transactions requires a degree of state management by CICS. State data, covering the "state" of the terminal and any user-specific data areas (commonly in a COMMAREA or in a CICS Temporary Storage queue) is stored in memory managed by CICS. Thus an increasing number of users will require an increasing amount of memory to be allocated. This memory is primarily stored in the CICS extended dynamic storage areas (EDSA) and should be considered when configuring the SIT EDSALIM option for the CICS region.

3.1.2.2 Business logic

Now we will look at the factors affecting business logic CPU usage.

Business logic CPU usage

- The business logic in the Trader application is that portion of the application that gets the information, and processes changes that the user requests, including file access and update. In our example it can be easily quantified using CICS monitoring facilities. CICS monitoring data can be used to determine the CPU utilization of each CICS task. The presentation logic of each task is very similar, and therefore this part of CPU cost is essentially constant across all tasks. It equals the cost of a task that does not execute any business logic, such as step 2. Hence we can determine the cost of the business logic in each step, by subtracting the cost of a task that does no business logic, from the total for a task that does execute business logic.

Number of disk I/O operations

- An increase in transaction rate may create excessive demands on the I/O subsystem and it may not be able to match the rate of increase of requests. If this happens, CICS will be unable to service higher transaction rates as tasks wait for a response from the I/O subsystem. When running many user requests in parallel, there will of course be an increasing number of I/O operations to files. The efficiency of performing I/O operations may decrease as the rate of requests increases, due to the limited bandwidth inherent in any physical I/O device, and to the serialization and locking required when updating recoverable resources.

Serialization characteristics (enqueue/dequeue)

- If the Trader application uses a recoverable file, then the update operation results in an implicit enqueue/dequeue. Under an increased load this could lead to an I/O bottleneck, as transactions queue waiting to update the file.

3.2 Measured CPU usage

In order to understand the CPU cost of running the 3270 version of the Trader application, we undertook a number of CPU measurements to get a baseline from which to estimate the delta costs of different methods of Web-enabling the Trader application.

First we measured the CPU usage for running one Trader business transaction, as described in Figure 10 on page 39 . This was undertaken using CICS monitoring. The results are shown in Table 2 .

Table 2: CPU costs from CICS monitoring for 3270 Trader application

CICS task	Presentation logic (CPU ms)	Business logic (CPU ms)	Total (CPU ms)
1	0.8	4.1	4.9
2	0.8	0.0	0.8
3	0.8	0.0	0.8
4	0.8	4.1	4.9
5	0.8	0.0	0.8
6	0.8	0.0	0.8
7	0.8	4.8	5.6
8	0.8	4.1	4.9
9	0.8	0.0	0.8
10	0.8	0.0	0.8
Totals	**8.0**	**17.1**	**25.1**

All numbers represent CPU milliseconds consumed by the CICS address space when running on an IBM 9672-R55 processor. The business logic component is effectively the path-length in program TRADERBL and the presentation logic is that in TRADERPL

From these numbers, we can see that when using the 3270 version of Trader, the majority of the CPU cost (68%) occurs in the business logic, and these costs are dominated by the CICS tasks that perform

file I/O operations.

With the scalability offered by CICS, these costs should increase in a linear fashion when running many user sessions in parallel. To verify the scalability of the Trader application, a workload consisting of instances of the Trader business transaction sequence was generated using the Teleprocessing Network Simulator (TPNS); and the CPU consumed by CICS, VTAM, and the overall total were measured using RMF monitoring. RMF monitoring records the CPU charged to each address space, along with the total used in the whole OS/390 system.

These costs are documented in Table 31 on page 170 and illustrated graphically in Figure 16 . The figures plotted are the % usage of a single R55 CPU with a maximum of 500% available. The CPU usage for the CICS and VTAM address spaces, along with the total CPU of the OS/390 system are plotted. In Figure 17 we plot the CPU cost in ms per transaction, against increasing workloads, in order to illustrate the scalable nature of 3270 CICS transactions.

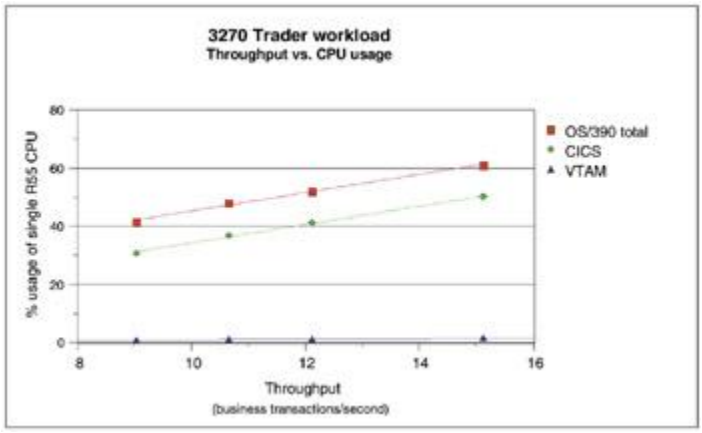


Figure 16: 3270 Trader workload, throughput vs. CPU usage

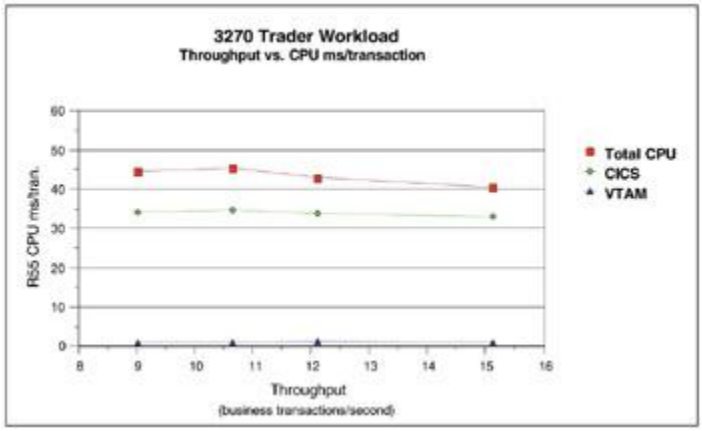


Figure 17: 3270 Trader workload, throughput vs. CPU ms/transaction

From Figure 17 it can be seen that the Trader workload scales very efficiently, and the CPU cost per transaction actually falls slightly as the throughput increases. This is due to the efficiencies gained at higher throughputs. It can also be seen that the proportion of CPU time spent in VTAM is consistently very low (approximately 2% of the total CPU used on the OS/390 system).

Using the plot in Figure 16 we produced a linear fit equation to calculate the CPU cost of the Trader application based on a given throughput.

A linear fit equation is of the form (y = k1 * x + k2). It predicts the value of y (in our case, CPU usage) based on the value of x (in our case throughput) and two constants, k1 and k2. The constant k1 is an indication of the slope and k2 the y-axis intercept. The degree of fit is reported by the R-square value, a value of 1.0 indicating a perfect fit. We use several linear fit equations throughout this study, all of which were produced using the *Series Trend* function in Lotus 1–2–3.

The linear equation for predicting the CPU cost of the 3270 Trader application is given in Figure 18 along with the predicted cost for a throughput of 10 business transactions per second. The R-square value for this equation was 0.994. Note that we will continue to use a throughput of 10 business transactions per second in all our capacity planning estimations later in this redbook.

Total CPU used in OS/390 system when running Trader:

```
Total CPU ms = (31.5 * throughput) + 137
```

Thus at a throughput of 10 business transaction/second:

```
Total CPU ms = (31.5 * 10) + 137 = 452 CPU ms
```

Throughput = business transactions per second

Figure 18: Linear equations for 3270 Trader CPU usage

You should note that the figures reported by CICS monitoring (Table 2 on page 45) for one Trader business transaction (25.1 CPU ms) are considerably less than the CPU usage per transaction in Figure 17 on page 47 (approximately 35 CPU ms). This is because the figures for CICS monitoring do not include general overhead of running the CICS region, just the individual costs associated with invoking a specific program.

Of this total 452 ms for running Trader using the 3270 interface, we can calculate how much should be allocated to the different OS/390 components. We do this by using the relative proportions reported for each component in our test measurements, as found in Table 31 on page 170 . The throughput of 10.6 business transactions/second was chosen, as it is the closest to our defined rate of 100 CICS tasks per second. This calculation is illustrated in Table 3 .

Table 3: CPU percentage breakdown for Trader via 3270 Web bridge

Component	Percentage of total per component	CPU usage for 10 business transactions (CPU ms)
CICS total	76.9%	348
VTAM	10.0%	10
OS/390 other	20.7%	94

Total	-	452

3.3 Trader performance

Using the results of our performance tests from Table 3 on page 48 we have plotted the CPU usage for each component when running the 3270 Trader application. This is shown in Figure 19 ; the figures plotted are CPU ms on an 9672-R55, for running 10 invocations of the Trader business transaction. Thus 10 Trader business transactions equate to 100 CICS tasks when using 3270 green screens.

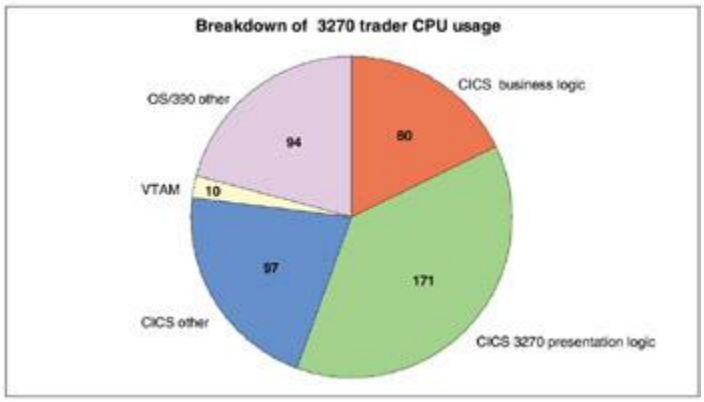


Figure 19: Breakdown of CPU usage for 3270 Trader application

Chapter 4: CWS with the 3270 Web bridge

Overview

In this chapter we discuss the Web-enabling of the Trader application using the 3270 bridge function of CICS Web support (CWS). We will refer to this function as the "3270 Web bridge". We then present a set of performance studies of a simple 3270 test application Web-enabled via the 3270 Web bridge, and go on to use this information to perform capacity planning for Web-enablement of the Trader application.

4.1 Converting the Trader application

The 3270 Web bridge allows for the Web-enablement of existing CICS 3270 applications with little or no change to the original 3270 based application. In the case of the Trader application, no changes were required to the presentation or business logic, as all the commands used were compatible with the restrictions imposed by the 3270 Web bridge. For further details on what changes may be necessary to an application, refer to *Revealed! Architecting Web Access to CICS* , SG24-5466.

4.1.1 Basic application structure

The Trader application consists of seven BMS maps. All BMS maps were converted to HTML templates by reassembling the BMS source with the BMS TEMPLATE option provided as part of CWS. This provides a basic HTML version of the original green screen; further customization can be carried

out to provide a more modern graphical user interface (GUI). Any such customization is unlikely to have a significant impact on performance, as the underlying application design will remain unchanged.

The flow of the 3270 Web-bridge-enabled Trader application is illustrated in Figure 20 and described below. It is essentially the same as the 3270 green screen version of Trader, since the 3270 Web bridge allows you to Web-enable your 3270 application with little or no modification.

1. The initial transaction is invoked through the 3270 Web bridge from a Web browser using a URL of the form http://myhost/cics/cwba/dfhwbtta/trad . This invokes the CWS module DFHWBTTA, which starts the TRAD transaction under a 3270 bridge environment. CICS creates a virtual 3270 terminal called a 3270 bridge facility, and the 3270 transaction then executes under the control of the Web bridge exit (DFHWBLT), unaware of the fact that the 3270 bridge facility is an emulated rather than a real 3270 terminal. TRADERPL calls the TRADERBL module in order to read the customer file. Then TRADERPL outputs the signon map, which is converted to HTML by CICS using a pre-generated HTML template.

2. The signon HTML page is sent back to CICS, and TRADERPL is invoked. The HTML version of the company selection display is sent to the Web browser.

3. A company is selected, and TRADERPL returns the main options display.

4. A *New Real-Time Quote* is selected, and TRADERPL calls TRADERBL, which reads the company and customer files, and then returns the real time quote display.

5. PF3 is selected to exit back to the main options display, invoking only TRADERPL.

6. Option 2 for *Buy Shares* is selected, and TRADERPL invoked, which returns the buy shares display.

7. The number of shares to purchase is selected. TRADERPL calls TRADERBL, which reads the company file and updates the stock holding in the customer file. TRADERPL then returns the main options display.

8. A *New Real-Time Quote* is selected (as in step 4). TRADERPL calls TRADERBL, which reads the company and customer files. TRADERPL then returns the real time quote display.

9. PF3 is selected to exit back to the main options display.

10. PF12 is selected, and the application terminates by TRADERPL, sending a final SEND TEXT message to the screen on completion.

As with the 3270 version of Trader, we will define these ten CICS tasks as constituting a single business transaction.

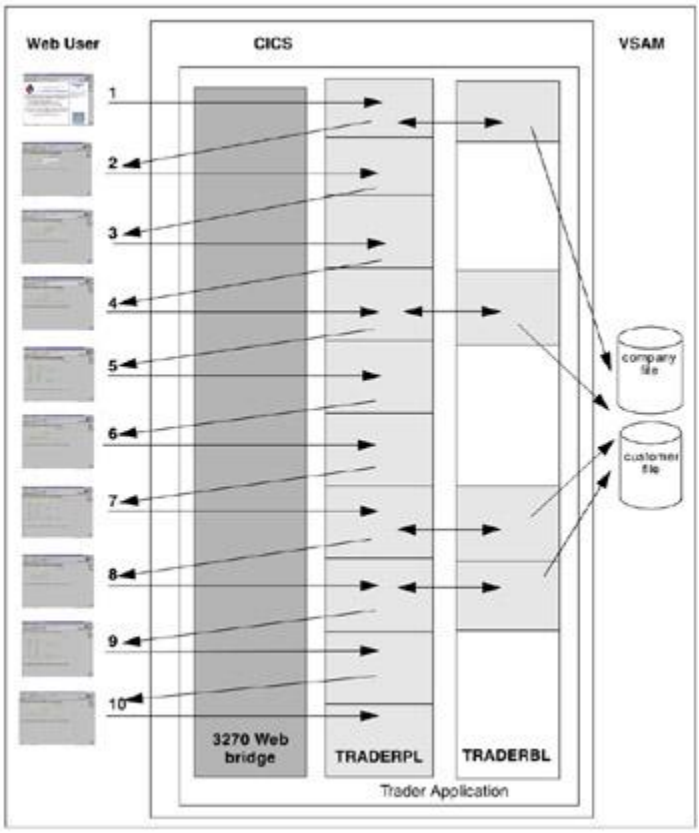


Figure 20: 3270 Web bridge Trader application flow

4.1.2 Application characteristics influencing performance

In this section we will discuss the factors which impact the performance of applications using the 3270 Web bridge.

Business transaction flow

The flow of the Trader application is identical when Web-enabled through the 3270 Web bridge, to the flow of Trader as a 3270 green screen application. When using the 3270 Web bridge to Web-enable an application, there are several management functions that impact the CPU usage of the application.

One of the key differences between Trader as a 3270 application and Trader as a Web-enabled application is how application state is maintained. In a 3270 environment, state data is naturally maintained using the CICS terminal. This allows the application to "know" its location within a pseudo-conversational chain and to store or pass data between different tasks in the pseudo-conversation. In contrast, the Internet is a stateless environment. Thus there is no permanent connection established between a Web browser and CICS. There is also no real 3270 device with which to associate session data, since the transaction is run under the control of a "3270 bridge facility". Instead, the 3270 bridge uses state tokens in hidden HTML fields to keep data for one user separate from others.

3270 bridge facility management

The 3270 Web bridge is responsible for managing these virtual 3270 devices. It does this using 3270 bridge facilities, which are created at the start of a pseudo-conversation and destroyed at the end. The

3270 bridge facility looks to the underlying application like a true 3270 device, including the ability to have associated state data, such as the next transaction identifier and a COMMAREA. The 3270 Web bridge uses the state tokens to associate the correct 3270 bridge facility with the correct user when new input arrives.

The 3270 Web bridge assumes that the user is beginning a new business transaction if the request does not carry state tokens from a previous interaction. The 3270 Web bridge regards the end of a pseudo-conversational chain as the absence of a "next transaction identifier" on the last CICS task. The SIT keep-time parameter, configured using the WEBDELAY keyword, tells CICS how long to keep a 3270 bridge facility that remains inactive, so that if the user loses connectivity (or interest) before the end of the pseudo-conversational chain, the 3270 bridge facility is not retained indefinitely.

Impact of pseudo-conversational chain length

The length of a pseudo-conversational chain within an application can affect CPU usage significantly. If the user is permanently held within the pseudo-conversation, then the state data and 3270 bridge facility are held continuously. This results in less work for the 3270 bridge garbage collector and shorter pathlengths within the 3270 bridge facility and state data management routines.

3270 bridge garbage collection

Garbage collection is the CICS management routine which is responsible for purging control blocks associated with Web state data. For each 3270 bridge facility created, the 3270 bridge maintains Web state data within CICS storage. As more Web state data is managed by the CPU usage associated with CWBG, the garbage collection transaction, will also be higher.

CWBG is started periodically. When it runs, it calls the CWS State Manager, which runs through the chain of Web state blocks destroying unused or timed out blocks, and flagging blocks that haven't been used for the time-out period to get destroyed on the next cycle. The frequency of garbage collection is controlled through the WEBDELAY SIT parameter, which is discussed below.

WEBDELAY(time_out,keep_time)

WEBDELAY is a CICS System Initialization Table (SIT) parameter which controls 3270 bridge facility time-out and application state data keep-time.

- ***Time_out*** is the maximum time, in minutes, that a CICS task running under a 3270 bridge facility is allowed to remain in a terminal wait state before being timed out.

- ***Keep_time*** is the amount of time, in minutes, during which application state data is maintained. Keep-time also controls the frequency of garbage collection.

Setting the WEBDELAY parameters to low values is advisable if the transaction rate is high and the number of CICS tasks within a business transaction is low. This avoids potential performance degradation caused by large amounts of 3270 bridge facility and state data being managed. However, setting WEBDELAY too low may cause bridge facilities and state data to be timed out before a business transaction has completed. In all our tests with the 3270 Web bridge, we set WEBDELAY to its lowest setting of (1,1). This gave good results in our environment, and the delay of one minute was greater than the think time in any of our Web client test scripts. Refer to A.2.2 , "CICS Web support with the 3270 Web bridge" on page 164 , for full details of our test configuration.

Persistent HTTP connections

The use of persistent HTTP connections (often termed KeepAlive), whereby subsequent HTTP sessions can reuse the underlying TCP/IP socket connection, will aid the performance of CWS applications. Support for persistent HTTP connections is enabled within CICS by using the SOCKETCLOSE keyword on the TCPIPSERVICE definition. Support is enabled with the OS/390 Web server using the directives PersistTimeout and MaxPersistRequest . The time-out period is counted from the receipt of the last HTTP datastream from each Web browser. Note that the Web browser client must also support persistent connections, and this includes an HTTP application proxy server if one is used.

We used an HTTP connection time-out of 10 seconds in all our 3270 Web bridge tests; this was greater than the think time in any of our test scenarios, and so allowed a pseudo-conversational chain to re-use the same TCP/IP socket connection. However, you should note that enabling persistent connections has the affect that each Web attach transaction (CWXN) remains long running until the time-out expires or the Web browser client closes the connection. This will require a higher number of CICS tasks to be running in your CICS region, and you should balance this against the performance benefits. We tested the effect of persistent HTTP connections in our Web-aware tests, which are detailed in Chapter 5 , "CWS with Web-aware presentation logic" on page 65 .

HTML templates

The placement of HTML templates is controlled through a DOCTEMPLATE CICS resource definition and has a potential impact on performance. The fastest load times for these HTML templates can be achieved by storing them as CICS load modules. These modules are managed like other loaded CICS programs and may be flushed out by program compression when storage is constrained. For more information on how to store HTML templates as CICS load modules, see the redbook, *CICS Transaction Server for OS/390 Version 1 Release3: Web Support and 3270 Bridge* , SG24-5480.

SEND TEXT

The CICS SEND TEXT command is a relatively costly command when executed through the 3270 Web bridge, as compared to using BMS. The reason for this is that the data stream contained in the SEND TEXT is translated between 3270 and HTML character-by-character as it is sent to the 3270 bridge facility. The CPU overhead associated with each SEND TEXT is thus greater than the CPU usage of a BMS commands. BMS commands are less CPU intensive because they use pre-generated HTML templates which can be cached in memory as CICS load modules.

4.2 Performance tests using the 3270 Web bridge

In our performance tests we used a simple BMS test application running under the 3270 Web bridge. This program consisted almost entirely of 3270 presentation logic, and thus was not the same as a real life application such as Trader, which is likely to spend more time in business logic than presentation logic. In the following section we present our testing methods and results when using our simple BMS test application. We then go on to detail a capacity planning methodology, and show you how to use our results to estimate the CPU usage when Web-enabling a real life application such as Trader.

4.2.1 Test environment

The test environment was equipped with sufficient hardware (processor, memory, DASD, network bandwidth) to eliminate any constraints. The operating system was OS/390 v2.7 together with CICS

Transaction Server V1.3. Full details of the software levels and parameters in effect during testing are listed in Appendix A "Test environments" on page 161 . The test system hardware configuration is illustrated in Figure 21 .

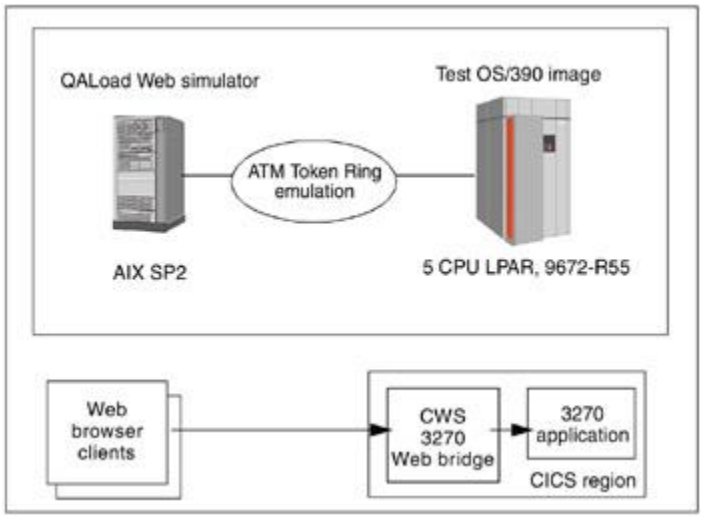


Figure 21: 3270 Web bridge test environment

4.2.2 Test methodology

For the 3270 Web bridge tests in this chapter, Web browsers were simulated using the Compuware QALoad product. These were run from two nodes of an AIX SP2 connected via Token Ring emulation over an ATM network to the S/390 processor, as illustrated in Figure 21 on page 57 . The think time was set to different values and the workload allowed to settle before a five minute measurement interval was sampled using the OS/390 RMF feature. This process was repeated for different think times to obtain results for five throughput rates from approximately 15 up to 100 Web requests per second. All the tests used 128 simulated Web browser clients.

Our 3270 test application was a simple BMS 3270 application. It consisted of a pair of CICS 3270 transactions which sent and received BMS maps in a pseudo-conversational mode. The BMS map contained some identifying header information and two 50 byte data fields. The program contained virtually no business logic, and as such, was only designed to test BMS data transmissions.

This workload was run in both a continuous, and a non-continuous, 3270 pseudo-conversation. In the non-continuous pseudo-conversation there are two CICS tasks in every pseudo-conversation. The first task sends a BMS map, and then initiates the second task using the RETURN TRANSID command. The second task receives the BMS map, issues a final SEND TEXT command, and then terminates. Thus, during this test, there is a continuous cost of creating and destroying 3270 bridge facilities as pseudo-conversations start and stop.

In the continuous pseudo-conversation, the second task was modified to issue a RETURN TRANSID command for the first transaction, such that the pseudo-conversational chain never finishes. Since a 3270 bridge facility is created on the first transaction in the pseudo-conversation and not destroyed until the end of the pseudo-conversation, there were no bridge facilities created/destroyed for the duration of the measurement, which was taken once the workload had settled.'

All our tests with the 3270 Web bridge used a CWS direct connection; it is also possible to use the CICS

WebServer Plugin in conjunction with the 3270 Web bridge, as described in 1.2.4 , "3270 Web bridge" on page 12 . If you wish to use the CICS WebServer Plugin, you should refer to Chapter 5 , "CWS with Web-aware presentation logic" on page 65 , where we give details of our performance measurements using the WebServer Plugin with Web-aware presentation logic.

4.2.3 Test results

In this section we present a summary of the performance measurements of our simple test BMS transaction using the 3270 Web bridge to illustrate the important points from the data. All the actual test data can be found in Appendix B "Performance data" on page 169 . Refer to Table 32 on page 171 and Table 33 on page 171 .

We did not report transaction response times in our test results, but IBM internal measurements have shown significant improvements in the 3270 Web bridge response time in CICS TS V1.3 as compared CICS TS V1.2, due to the restructuring of CWS in CICS TS V1.3.

Figure 22 illustrates the CPU usage for our test of a non-continuous pseudo-conversation. Figure 23 illustrates how the total OS/390 CPU usage varied between the non-continuous pseudo-conversation and the continuous pseudo-conversation scenarios. In both graphs, the figures plotted are the % usage of a single R55 CPU with a maximum of 500% available.

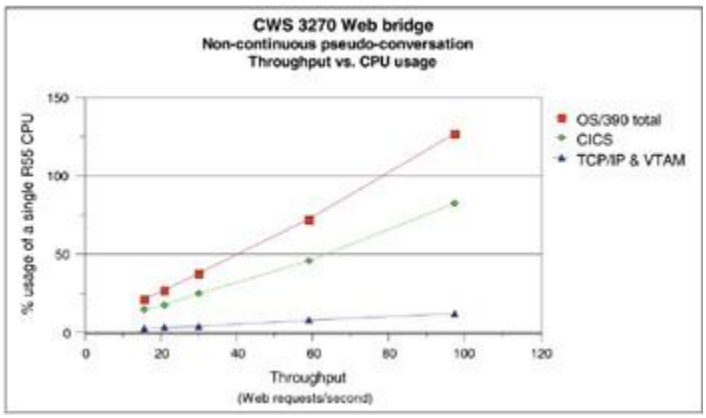


Figure 22: 3270 Web bridge, non-continuous pseudo-conversation

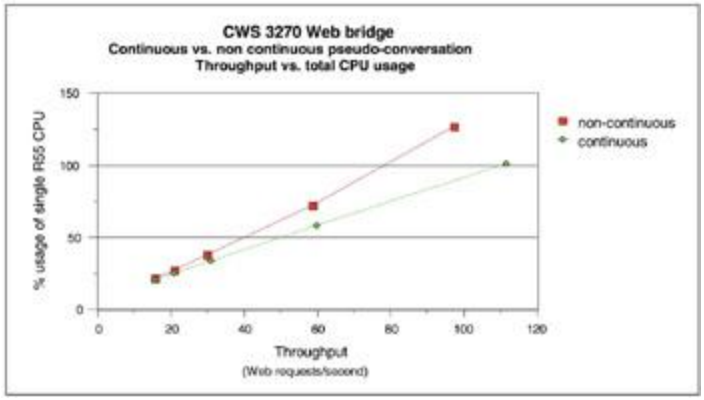


Figure 23: 3270 Web bridge, continuous vs. non-continuous pseudo-conversation

Figure 23 illustrates that the CPU usage associated with a 3270 Web-bridge-enabled transaction is primarily within the CICS address space. The CPU usage associated with TCP/IP becomes a decreasing portion of the total CPU usage as the throughput increases.

The higher CPU usage of a non-continuous over a continuous conversation can be clearly seen in Figure 23 . Since both workloads sent and received the same amount of data, the higher cost of a non-continuous pseudo-conversation is attributable to the increased overhead of managing bridge facilities and state data when using a non-continuous pseudo-conversation.

4.3 Capacity planning for the 3270 Web bridge

In this section we use the results of our previous performance tests to create a capacity planning methodology for estimating the CPU usage of a Web-enabled CICS application using the 3270 Web bridge. We then use this methodology to estimate the CPU usage when the Trader application is Web-enabled using the 3270 Web bridge. We also present the results of a test to confirm this capacity planning estimate.

4.3.1 Capacity planning methodology

Using our the results of our performance tests for the 3270 Web bridge, we have calculated a general increase formula for Web-enablement using the 3270 Web bridge. This formula uses the length of the 3270 pseudo-conversational chain as a key factor and provides a different increase, depending on the length of the pseudo-conversational chain. This formula has been subject to separate validation using several different 3270 workloads with differing amounts of screen data, and has been found to give good results. The formula is documented in Figure 24 .

Continuous pseudo-conversation:

```
New total CPU ms = Original 3270 total CPU ms + (throughput * 8.54)
```

Non-continuous pseudo-conversation:

```
New total CPU ms = Original 3270 total CPU ms + (throughput * 11.1)
```

Total CPU = all CPU consumed in OS/390 LPAR in one second

Throughput = CICS tasks per second

Figure 24: 3270 Web bridge general increase formulae

We will use the continuous pseudo-conversation formula for estimating the Trader workload, since Trader has ten CICS tasks in one business transaction, which is a relatively high number. If you are in

doubt about which formula to use, we would advise using the non-continuous pseudo-conversation, as this will give more margin for error.

4.3.2 Capacity planning estimate

Applying the general increase formula in Figure 24 on page 61 to Trader, we anticipate the increase represented in Table 4 for running 10 Trader business transactions per second via the 3270 Web bridge. The original costs of running the Trader application in a 3270 environment were calculated using the linear equations in Figure 18 on page 48 .

Table 4: Estimated CPU increase for Trader via 3270 Web bridge

Old total (CPU ms)	Throughput (tasks/sec.)	3270 Web bridge general increase	New total (CPU ms)
452	**100**	**8.54**	**1306**

Of this total 1306 ms for running Trader using the 3270 Web bridge, we can estimate how much should be allocated to the different OS/390 components. We do this by first deducting the known cost of 171 ms for the business logic in TRADERBL, and then using the relative proportions reported for each component in our test results. We used our results from a non-continuous pseudo-conversation in Table 32 on page 171 . A throughput of 111.8 Web requests/second was chosen, as it is the closest to our defined rate of 100 CICS tasks per second (or 10 business transactions per second). This calculation is illustrated in Table 5 .

Table 5: CPU percentage breakdown for Trader via 3270 Web bridge

Component	Percentage of total per component	CPU usage for 10 business transactions component (CPU ms)
CICS TRADERBL	-	171
CICS other	81.8%	928
TCP/IP & VTAM	15.2%	173
OS/390 other	3%	34
Total	-	1306

4.3.3 Confirming our estimate

In order to quantify our capacity planning estimate, we actually measured the CPU usage of the Trader application Web-enabled using the 3270 Web bridge. We determined from CICS monitoring data that a single business transaction using persistent HTTP connections consumed, on average, 93 CPU ms within the CICS address space. This included the cost of the CWBA (alias) and CWBG (garbage collection) transactions. Our estimation, documented in Table 5 on page 62 , shows a usage of 171+928 = 1099 CPU ms per 10 business transactions that is allocated to CICS, which equates to 101 CPU ms per individual business transaction. This measured value of 93 CPU ms is 8% less than our estimate of 101 CPU ms. This indicates that our capacity planning methodology gives good results for the 3270 Web bridge.

4.4 Trader performance comparison

Using our capacity planning estimate in Table 5 on page 62 , we have compared the CPU usage of the Trader application running via the 3270 Web bridge to the original costs of the 3270 version. This is illustrated in Figure 25 . The figures plotted are CPU ms on an 9672-R55, for running 10 invocations of the Trader application.

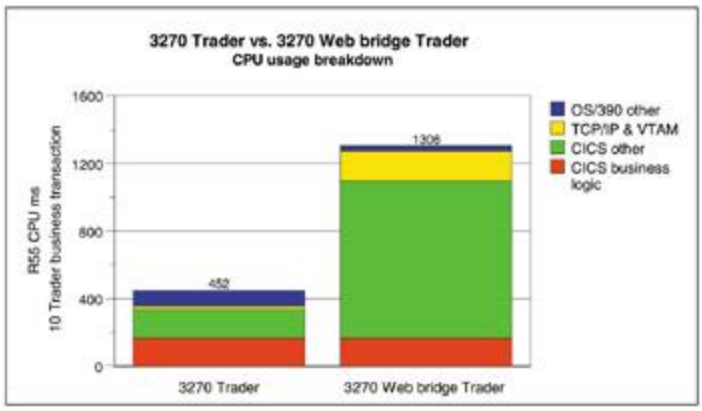


Figure 25: Capacity planning estimates for Trader via 3270 Web bridge

This graph illustrates that when using the 3270 Web bridge, the cost of the business logic portion of the application remains constant, but the cost of the presentation logic (*CICS other*) increases approximately five fold. This is due to the high overhead of emulating and managing the 3270 environment within CICS. This additional CPU usage would mean that on our 9672-R55 processor, the CICS region CPU usage would theoretically exceed 1000 CPU ms or 100% of one CPU. However, this is greater than the maximum capacity of a single CICS region. Solutions to this situation are discussed further in 8.3 , "Using too much CPU" on page 146 .

In summary, the ease of implementation of a 3270 Web bridge solution needs to be balanced against the relatively high CPU cost of such a solution. Alternative non-3270 based Web-enabling solutions are discussed next in Chapter 5 , "CWS with Web-aware presentation logic" on page 65 and Chapter 7 , "The OS/390 CTG" on page 103 .

Chapter 5: CWS with Web-aware presentation logic

Overview

In this chapter we summarize how to provide Web access to the business logic of the Trader application, using CICS Web support (CWS) together with new Web-aware CICS presentation logic. We then present a set of performance studies of for various laboratory workloads and go on to use these figures to perform capacity planning for Web-enablement of the Trader application.

5.1 Converting the Trader application

Two programming tasks are required when Web-enabling the Trader application using CWS and Web-aware presentation logic. This is in contrast to the CWS 3270 Web bridge solution, which requires little or no programming.

First, we must separate the 3270 presentation logic from business logic in the application. This is easy to do in the Trader application, because the business logic and presentation logic are isolated in separate modules, TRADERBL and TRADERPL, respectively. In many legacy CICS applications, this is not the case, and separating the business and presentation logic may require extensive re-engineering. It is, however, an essential part of using this and other CICS Web-enabling techniques and offers several benefits which should become clear throughout this chapter. Figure 26 illustrates this required division of presentation and business logic.

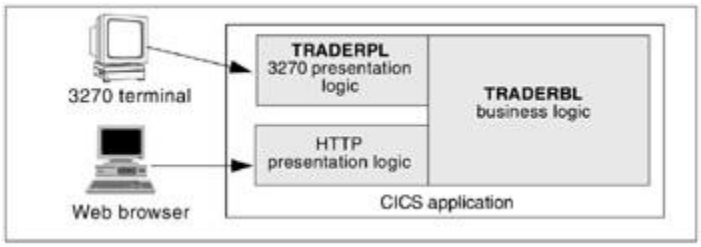


Figure 26: Separation of business logic and presentation logic

The second programming task is to supply the HTTP Web-aware presentation logic, which will have to perform two new functions:

- Interpret the browser input and, when we need a business function, convert it to the COMMAREA format expected by TRADERBL.
- Produce responses in HTML, including converting output returned in the COMMAREA from the called business logic (TRADERBL).

This Web-aware presentation logic is best implemented in a new HTTP based presentation module, just as TRADERPL was the 3270 based presentation module for 3270 devices. This module can either be a specific HTTP presentation logic module, or it can be implemented in the converter routine that the CWS can invoke. We have chosen the second option and put the new logic into our convertor which is called TRACERCV. The convertor is invoked by the CWS business logic interface (BLI) and uses a COMMAREA to pass data to the business functions in TRADERBL.

To create the HTTP based presentation logic for an application such as Trader, there are two fundamentally different CICS programming techniques:

- The CICS WEB API used together with the DOCUMENT API
- COMMAREA manipulation and the CWS HTML template manager

In releases of CICS TS prior to V1.3, the only choice was to use COMMAREA manipulation and the CWS HTML template manager to manually build HTML. The new WEB and DOCUMENT APIs supplied in CICS TS V1.3 greatly ease this task and also overcome the 32 KB limit on the size of HTTP messages that could previously be passed by means of the CICS COMMAREA.

5.1.1 Basic application structure

In this section we describe how the Trader application was Web-enabled via the facilities of CWS. The new Web-aware presentation logic was implemented in a converter module called TRADERCV, and the HTTP data streams manipulated using the CICS WEB and DOCUMENT API. This converter can be used via a CWS direct connection or the WebServer Plugin.

The flow of CICS tasks in one business transaction is illustrated in Figure 27 and documented below.

1. CWS receives the initial HTTP GET for TRADERPL from the browser, with the converter program TRADERCV specified in the request. A CWXN Web attach transaction is started and handles all further HTTP requests for this business transaction, using a persistent HTTP connection. TRADERCV is invoked, which builds the *signon page* HTML using the CICS DOCUMENT and WEB API and sends it back to the browser.

2. The Web browser does an HTTP POST of the signon form. The request is passed to TRADERCV, which calls TRADERBL passing a COMMAREA as input. TRADERBL verifies the userid and password, reads the company and customer files and returns the result to TRADERCV via the COMMAREA. TRADERCV returns the *company selection* HTML page.

3. The Web browser does a POST of the company selection form. TRADERCV receives this data, and calls TRADERBL via the COMMAREA. TRADERBL browses the company and customer files, builds the *buy-sell quote page* , and returns the quote to the browser.

4. The Web browser does an HTTP POST of the completed *buy-sell quote page* with the number of shares to buy. The request is passed to TRADERCV, which calls TRADERBL passing a COMMAREA. TRADERBL updates the share holdings in the customer file, and calculates the value of the updated holdings. TRADERCV sends the *buy-sell quote page* with the value of the new share holdings.

5. The Web browser does an HTTP POST when the user clicks on the *End Trader* radio button. TRADERCV returns the *Trader Complete* page.

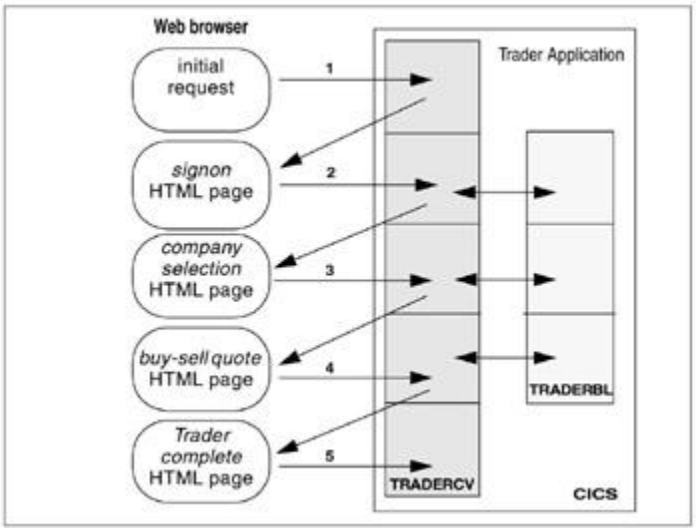


Figure 27: Trader application flow using CWS and Web-aware presentation logic

Comparing these flows to that of the 3270 Trader application described in Chapter 3.1.1 , "Basic application structure" on page 38 , it is clear that the number of CICS tasks in one business transaction has been reduced from ten to five. This was enabled by the removal of the dependency on the hierarchical 3270 menu system, and the implementation of a new presentation layer.

These flows were analyzed using CICS tracing and the data sizes measured. The results are summarized in Table 6 , and will be used later in our capacity planning calculations.

Table 6: HTTP datastream sizes when using Trader via CWS

Step	HTTP method	Bytes received	Bytes sent
1	GET	261	2007
2	POST	453	1923
3	POST	458	2406
4	POST	449	2406
5	POST	456	1342

5.1.2 Application characteristics influencing performance

There are a number of characteristics of an application that affect its cost in terms of system resources and hence its performance. For an application using CICS Web support, the following principal factors can be identified.

Size of datastream

For both incoming and outgoing HTTP datastreams, the CPU cost in CICS, TCP/IP, and VTAM (eNetwork Communications Server), and the OS/390 Web server, will all increase as the size of the HTTP message increases. Note that in more recent versions of OS/390 eNetwork Communications Server, CPU usage for TCP/IP is allocated to both the TCP/IP and VTAM address spaces. The general principal, as in all communications tuning, should be reduce the amount and frequency of data transmitted, and to make sure packet sizes match throughout the network.

Outgoing HTTP messages (from CICS) containing HTML may often contain comments put in by well-meaning HTML application programmers. These are never presented on the browser screen, but nevertheless are transmitted across the network from the Web server to the Web browser, sometimes constituting a significant percentage of the data. To reduce outgoing datastream size, datastreams should be created with the minimum number of HTTP components, usually just the HTTP level, a suitable error code, a message, a content type, and a content length header.

Inbound datastream size can be significantly affected by different Web browsers, which will send different amounts of HTTP header data, much of which aren't really essential. There is little you as the application programmer can do about this, but you should be aware of the fact when testing or analyzing your applications.

HTTP presentation logic

Within CICS, the Web-aware HTTP presentation logic can either be coded using the WEB API or by using COMMAREA manipulation and the HTML template manager. Further details on the difference in

CPU usage between these techniques are given in Figure 31 on page 74 . This HTTP presentation logic can either be implemented within the encode and decode functions of the CWS converter, or a can be placed in a separate HTTP presentation logic module. In terms of performance, the only significant difference between these two methods is the number of EXEC CICS LINKS required to call the presentation logic, the converter design having the most.

Persistent HTTP connections

The usage of persistent HTTP connections from the Web browser to the Web server can give a significant performance advantage. The support of persistent HTTP connections in CWS has already been discussed in "Persistent HTTP connections" on page 56 , and the same principles apply to a Web-aware design as when using the 3270 Web bridge. We analyzed the effect of persistent HTTP connections in our Web-aware tests, which are detailed in later in this chapter.

State data

A typical CICS business transaction is composed of several short running CICS tasks, this applies equally to Web based business transactions as it does to a traditional 3270 legacy application. To enable continued processing of a business transaction, it is usually necessary to store some "state" data within the CICS region. This state data can be stored within CICS temporary storage queues (TSQs) using the facilities of the supplied CWS sample state management program (DFH$WBST). Like all CICS TSQs, this information can be stored within CICS memory or on physical DASD. Obviously, storing large amounts of state data within CICS memory will impact the storage requirements of your CICS region, but will also give better performance than data stored on physical DASD. To optimize performance, you should aim to be conservative both with the amount of state data stored within CICS and the number of CICS tasks that constitute your Web business transaction.

5.2 Performance tests using the CWS and Web-aware presentation logic

In the following section we present our testing methods and results for a range of measurements of CPU usage for HTTP data transfers using simple Web-aware CICS applications.

5.2.1 Test environment

The system parameters in effect during our testing are listed in A.2.3 , "CICS Web support with Web-aware presentation logic" on page 165 . These parameters are not necessarily recommended for all environments, but were found to give good results in our circumstances. The hardware environment is illustrated in Figure 28 , and also documented in A.1 , "Hardware environment" on page 161 .

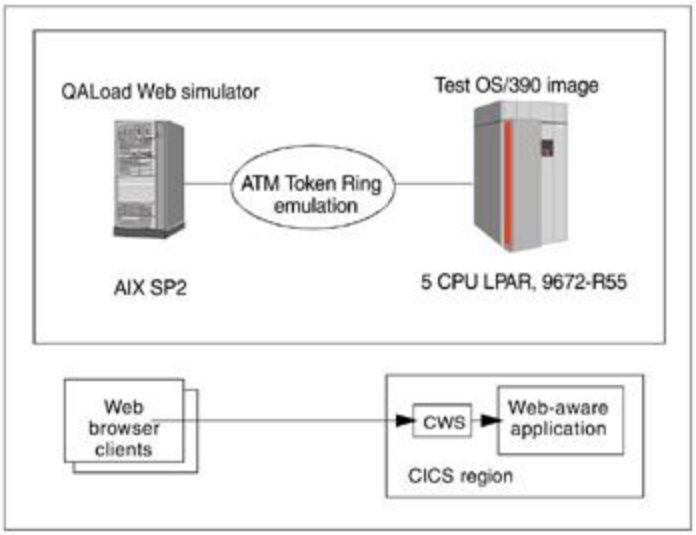


Figure 28: CWS test environment

We used the same series of HTTP data transfer tests, in two different environments.

- Direct connection from the Web browser to CWS using the CICS Sockets listener.
- Indirect connection from the Web browser to CWS, using the OS/390 Webserver and the CICS WebServer Plugin.

The only difference between these tests was in the route of the HTTP data stream and the means by which CICS handles the HTTP datastream, the CICS application and the Web browser workload setup being identical.

5.2.2 Test methodology

For the CWS tests in this chapter, Web browsers were simulated using the Compuware QALoad product. These were run from two nodes of an AIX SP2 connected via Token Ring emulation over an ATM network to the S/390 processor, as illustrated in Figure 28 on page 70 .

A range of five throughputs from approximately 20 to 190 Web requests per second were achieved by varying the think time of the simulated Web browsers within the QALoad tool. The number of Web users was set to 200 for the CWS direct connection environment and 70 when using the CICS WebServer Plugin. The workloads were allowed to settle before a five minute measurement interval was sampled using OS/390 RMF.

All our tests used a Web-aware CICS application design, where the HTTP manipulation was performed within the code of the test program. Four slightly different applications were used, two for testing sending and receiving of data via the CICS WEB and DOCUMENT APIs, and two for sending and receiving data via COMMAREA manipulation and the HTML template manager. The sending of data by CICS was tested using HTTP GET requests from the Web browser, and the receiving of data was tested using HTTP POST requests.

The principal quantifiable cost associated with Web-aware presentation logic in a CICS application will be the cost of sending and receiving the HTTP datastream. To quantify this cost, we performed a set of

measurements to determine the CPU cost of sending and receiving different size HTTP data from a CICS Web-aware application. We measured datastream sizes of 100 bytes, 5 KB, 15 KB, 32 KB, 33 KB and 50 KB. However, the tests using the WebServer Plugin were limited to a maximum of 32 KB, due to the limiting size of COMMAREAs when using the External CICS Interface (EXCI). All tests were run with and without persistent HTTP connections to quantify the savings of doing so.

We also ran a set of 5 KB HTTP data transfer tests using an application written with the old COMMAREA manipulation programming technique; this was done to enable a performance comparison of the new WEB API technique and the COMMAREA manipulation technique.

5.2.3 Test results

In this section we present a summary of the performance measurements of our tests using CWS with Web-aware applications to illustrate the important points from the data. The full set of results are documented in B.3 , "CWS with Web-aware presentation logic" on page 172 .

We have not reported transaction response times in our test results, but IBM internal measurements have shown significant improvements in the 3270 Web bridge response time in CICS TS V1.3 compared CICS TS V1.2 due to the internal restructuring of CWS.

In Figure 29 we show the CPU usage for increasing throughputs for a 5 KB send workload, using a direct connection and the WEB API, with persistent HTTP connections. The figures plotted are the percentage usage of a single R55 CPU, with a maximum of 500% available. Throughput is defined as the number of Web requests per second, and measured in CICS Web-aware tasks per second.

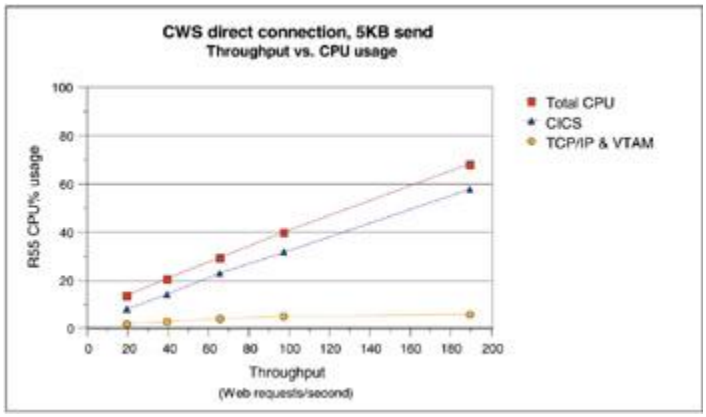


Figure 29: CPU usage of 5 KB send using CWS direct connection

In Figure 30 we plot the same results, but using the CICS WebServer Plugin. Note that the CPU usage for the Web server includes the CPU used by the CICS WebServer Plugin as well as the Web server itself, since the WebServer Plugin runs within the Web server address space.

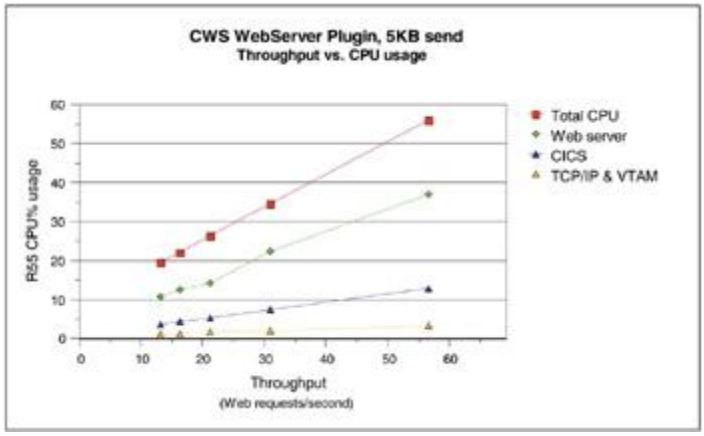


Figure 30: CPU usage of 5 KB byte send using CWS WebServer Plugin

From these two graphs, the following generalizations can be made which hold true across all data sizes and connection techniques:

- The total OS/390 CPU usage is lower when using the CWS direct connection as opposed to the CICS WebServer Plugin. This is to be expected, given the more complex pathlength involved when using the WebServer Plugin as compared to a direct connection.

- The CICS CPU usage is lower per call when the WebServer Plugin is used, as opposed to a direct connection. This is because the WebServer Plugin replaces a considerable proportion of the function that otherwise occurs in CICS when using a direct connection.

- The CPU usage by eNetwork Communication Server (TCP/IP and VTAM) is a small percentage of the overall total CPU usage.

An additional set of tests was run using a COMMAREA manipulation style application. The purpose of this was to assess if there was any significant difference in CPU cost between using the new CICS WEB API and the old COMMAREA manipulation technique to build the HTTP datastream. The average CPU ms per Web request over all the throughputs tested are displayed in Figure 31 , and the full set of results can be found in Table 59 on page 180 and Table 60 on page 180 . The data plotted is the total OS/390 CPU cost and the CICS CPU cost in ms per Web request. Tests were conducted for 5 KB sends and receives using a CWS direct connection.

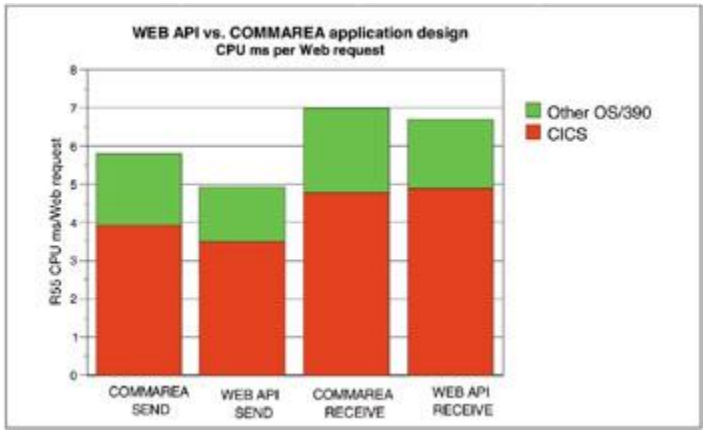


Figure 31: CWS HTTP data transfers, COMMAREA vs. WEB API application design

These results show that there are relatively minor differences in CPU utilization between the WEB API and COMMAREA manipulation application techniques, but do indicate that sends are somewhat cheaper than receives. The ease of use of the new WEB API provided in CICS TS V1.3 over the old COMMAREA manipulation technique is likely to the be the overriding factor in deciding which technique to use for new CICS Web-aware applications.

Next we analyzed the cost for different size data transfers using the WEB API. We averaged the total OS/390 CPU cost per Web request for the whole range of throughputs measured and then compared these for each data size, both for sends and receives. The results are illustrated in Figure 32 on page 75 for a direct connection and in Figure 33 on page 76 for the WebServer Plugin. The figures plotted are the average total OS/390 CPU ms per Web request for the different data sizes. The actual data for the direct connection measurements can be found in Table 35 on page 174 to Table 60 on page 180 , and the data for the WebServer Plugin measurements in Table 62 on page 182 to Table 69 on page 184 . Note that we were unable to measure the CPU usage for receives with a non-persistent HTTP connection through the CICS WebServer Plugin due to time constraints.

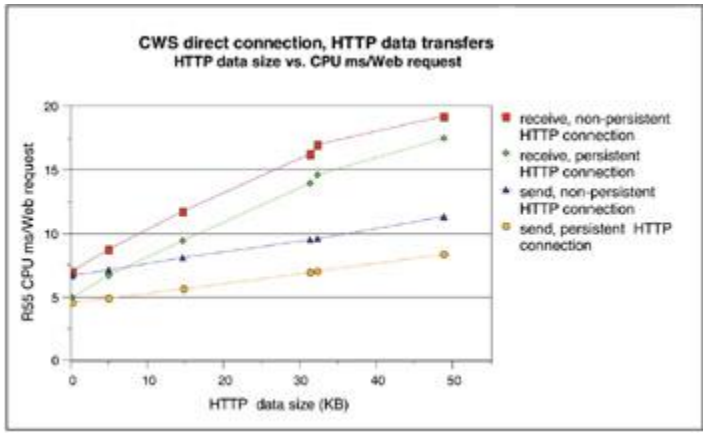


Figure 32: CPU usage for HTTP data transfers using CWS direct connection

The figures for the CWS direction connection and the WebServer Plugin both demonstrate good scalability for HTTP data transfers. The main observations from these figures are as follows:

- There is significant cost associated with a minimal data transfer (the null cost), and this cost is likely to be the dominant cost for small data transfers (less than 10 KB).
- The null cost is considerably higher if using non-persistent HTTP connections as compared to persistent HTTP connections, but the amount of additional CPU consumed per byte is about the same.
- Sends are significantly cheaper than receives.
- The figures for 50 KB receives showed a slight decrease in cost per byte over the smaller data sizes.

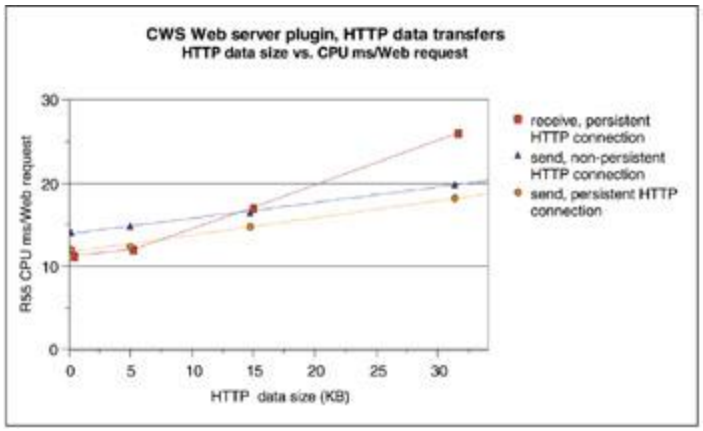


Figure 33: CPU usage for HTTP data transfers CWS and WebServer Plugin

Using the plotted data we were able to obtain good linear fit equations relating CPU usage per Web request to data size; these are used later in our capacity planning estimation. It should be noted that these costs are only the average CPU cost per request. Analysis of the data showed that the cost per Web request tends to decrease as throughput increases, and this effect was more pronounced when using the WebServer Plugin.

5.3 Capacity planning for a CWS Web-aware application

In this section we use the results of our previous performance tests to create a capacity planning methodology for estimating the CPU usage of a Web-enabled CICS application using the CWS with new Web-aware presentation logic. We then use this methodology to estimate the CPU usage when the Trader application is Web-enabled using the CWS with new Web-aware presentation logic.

5.3.1 Capacity planning methodology

Our Web-enabled Trader application has five CICS tasks in one business transaction. Three of these tasks invoke the CICS Trader business logic module TRADERBL. The HTTP presentation logic is written using the WEB API, and the size of the data streams sent and received are documented in Table 6 on page 68 . We shall assume that persistent HTTP connections are configured. Thus we can calculate the basic costs of the application based on the original business logic costs plus the HTTP data transmission costs.

The costs of the business logic in TRADERBL are already documented in Table 2 on page 45 , and we shall re-use this data, taking into account that only three calls are made to the business logic in our Web-enabled trader application, as opposed to four when using the original 3270 version. We shall calculate the HTTP data transmission costs using linear fit equations relating CPU ms per request to size of the HTTP data stream. These were produced from the graphs in Figure 32 on page 75 and Figure 33 on page 76 . The equations are listed in Figure 34 . The R-square values for the CWS direct connection equations were all greater than 0.99, and the R-square values for the WebServer Plugin equations were all greater than 0.98. Note that since the figures for 50 KB receives showed a small decrease in cost per byte over the smaller data sizes, they were excluded from the linear fits.

When using a CWS direct connection:

send, persistent HTTP connection

```
Total OS/390 CPU ms per Web request = 4.52 + (0.078 * data KB)
```

send, non-persistent HTTP connection

```
Total OS/390 CPU ms per Web request = 6.65 + (0.093 * data KB)
```

receive, persistent HTTP connection

```
Total OS/390 CPU ms per Web request = 5.11 + (0.289 * data KB)
```

receive, non-persistent HTTP connection

```
Total OS/390 CPU ms per Web request = 7.13 + (0.301 * data KB)
```

When using CWS and the CICS WebServer Plugin:

send, persistent HTTP connection

```
Total OS/390 CPU ms per Web request = 11.7 + (0.206 * data KB)
```

send, non-persistent HTTP connection

```
Total OS/390 CPU ms per Web request = 13.9 + (0.189 * data KB)
```

receive, persistent HTTP connection

```
Total OS/390 CPU ms per Web request = 10.2 + (0.492 * data KB)
```

Figure 34: Equations for CPU usage per Web request based on HTTP data size

5.3.2 Capacity planning estimate

The costs for one Web-enabled Trader business transaction consists of five separate Web requests or CICS tasks as shown in Table 7 .

Table 7: Breakdown of costs in CWS Web-enabled Trader

CICS task	HTTP method	Data received (bytes)	Data sent (bytes)
1	GET	261	2007
2	POST	453	1923
3	POST	458	2406
4	POST	449	2406
5	POST	456	1342

5.3.2.1 CWS direction connection estimation

To estimate the CPU usage at each step when using a direct connection, we use the linear equations given in Figure 34 on page 77 , relating throughput to the size of the HTTP datastream. To this we add the known cost of the business logic for Trader as given in Table 2 on page 45 .

For the first CICS task in the Trader business transaction, we use the cost for a non-persistent HTTP connection, since the HTTP connection must first be established. For the next four CICS tasks in the business transaction, we use the cost for persistent HTTP connections. At each step there is a relatively small amount of data sent to CICS from the Web browser. We do not factor this into our estimates, as doing so proved to be of small consequence. Similarly, we do not take into account that in the test measurements, there is a small amount of data sent for the receive tests, and a small amount of data received for the send tests. The calculation of the CPU usage at each step is illustrated in Table 8 .

Table 8: CPU usage per Web request with CWS and direct connection

Step	Linear equation	Data sent (bytes)	CWS (CPU ms)	TRADERBL (CPU ms)	Total (CPU ms)
1	Total CPU ms = 6.65 + (0.093 * data KB)	2007	6.8	0	6.9
2	Total CPU ms = 4.52 + (0.078 * data KB)	1923	4.7	4.1	8.7
3	Total CPU ms = 4.52 + (0.078 * data KB)	2406	4.7	4.1	8.8
4	Total CPU ms = 4.52 + (0.078 * data KB)	2406	4.7	4.8	9.5
5	Total CPU ms = 4.52 + (0.078 * data KB)	1342	4.6	0	4.6
Totals			**25.5**	**13.0**	**38.5**

We can now calculate the CPU usage required to run the Web-enabled Trader at a throughput of 10 business transactions per second (or 50 Web requests per second) as follows:

CPU Usage for Trader

Total CPU ms = 38.5 * 10 = 385 CPU ms

Of this total 385 CPU ms, we can calculate how much should be allocated to the different OS/390 components. We do this by first deducting the known cost of 171 ms for the business logic in TRADERBL, and then calculating the percentage breakdown for the individual components. We do this by using the relative proportions reported for each component in our 5 KB test measurements with persistent HTTP connections, as found in Table 36 on page 174 . The throughput of 39.57 transactions per second was used, as it is the closest to our defined rate of 50 Web requests per second (or 10

business transactions per second). This calculation is illustrated in Table 9 .

Table 9: CPU percentage breakdown for CWS direction connection

Component	Percentage of total per component	CPU usage for 10 business transactions (CPU ms)
CICS TRADERBL		130
CICS other	77.3%	197
TCP/IP & VTAM	13.4%	34
OS/390 other	9.3%	24
Total		385

5.3.2.2 WebServer Plugin estimation

To estimate the CPU consumption at each step when using the CICS WebServer Plugin we will use a similar methodology to that used previously in 5.3.2.1 , "CWS direction connection estimation" on page 78 , but instead use the appropriate linear equations for the CICS WebServer Plugin from Figure 34 on page 77 . This calculation is illustrated in Table 10 .

Table 10: CPU usage per Web request with CWS WebServer Plugin

Step	Linear equation	Data SENT (bytes)	CWS CPU ms	TRADERBL CPU ms	Total
1	Total CPU ms = 13.9 + (0.189 * data KB)	2007	14.3	0	12.0
2	Total CPU ms = 11.7 + (0.206 * data KB)	1923	12.1	4.1	13.9
3	Total CPU ms = 11.7 + (0.206 * data KB)	2406	12.2	4.1	14.0
4	Total CPU ms = 11.7 + (0.206 * data KB)	2406	12.2	4.8	14.7
5	Total CPU ms = 11.7 + (0.206 * data KB)	1342	12.0	0	9.7
Total			**62.8**	**13.0**	**75.8**

Since each Trader business transaction comprises five Web requests, we can calculate the CPU usage to run the Web-enabled Trader at a throughput of 10 business transactions per second (or 50 Web requests per second) as follows:

CPU Usage for Trader

Total CPU ms = 75.8 * 10 = 758CPU ms

Of this total 758 CPU ms, we can calculate how much should be allocated to the different OS/390 components. We do this by first deducting the known cost of 171 ms for the business logic in TRADERBL, and then calculating the percentage breakdown for the individual components. We calculate the breakdown for the individual components by using the relative proportions reported for each component in our 5 KB test measurements with persistent HTTP connections, as found in Table 63 on page 182 . The throughput of 56.49 transactions per second was used from these figures, as it is the closest to our defined rate of 50 Web requests per second (or 10 business transactions per second). This calculation is illustrated in Table 11 .

Table 11: CPU percentage breakdown for CWS WebServer Plugin

Component	Percentage of total per component	CPU usage for 10 business transactions (CPU ms)
CICS TRADERBL		130
CICS other	23%	144
TCP/IP & VTAM	6%	38
Web server	64%	402
OS/390 other	7%	44
Total		758

5.3.3 Confirming our estimate

In order to quantify our capacity planning estimate, we measured the CPU usage of the Trader application using a direct CWS connection with new Web-aware presentation logic, implemented in the CWS converter.

We determined from CICS monitoring data that one single business transaction using persistent HTTP connections consumed, on average, 58 CPU ms within the CICS address space. Our estimation documented in Table 9 on page 80 shows a usage of 130 + 197 = 327 CPU ms per 10 business transactions that is allocated to CICS, which equates to 33 CPU ms per individual business transaction. At a higher throughput the actual cost per transaction will decrease, thus reducing this difference. Even so, our measured value of 59 CPU ms is still 26 CPU ms higher than our estimate of 33 CPU ms.

On investigation, the reason for this difference is thought to be because of the design of the new Web-aware presentation logic program TRADERCV. TRADERCV does significantly more than simply replace the BMS RECEIVE MAP and SEND MAP in Trader, with WEB RECEIVE and WEB SEND calls.

Within TRADERCV there are a number of calls to the CICS-supplied state management program (DFH$WBST) to keep application state data across related Web browser requests based on a token passed between the Web browser and CICS. This state management program stores this state data using the facilities of CICS Temporary Storage Queues (TSQ). The presentation logic in the converter, TRADERCV, also makes extensive use of the CICS DOCUMENT API to build HTML pages before they are sent using the WEB API commands. Neither the state management program nor the

DOCUMENT API are heavy CPU users, but when their cost is added to a simple transaction, it appears to have a significant effect which is not factored into the results of our simple estimate based on data transmission costs alone.

5.4 Trader performance comparison

Using our capacity planning estimates for a CWS direct connection in Table 9 on page 80 , and for the CICS WebServer Plugin in Table 11 on page 81 , we have compared the CPU usage of the Trader application running as a Web-aware application to the original costs of the 3270 version. This is illustrated in Figure 35 . The figures plotted are CPU ms on an 9672-R55, for running 10 invocations of the Trader business transaction.

Note that 10 Trader business transactions equate to 50 Web requests or CICS tasks when using CICS Web support, and 100 CICS tasks when using the 3270 green screens.

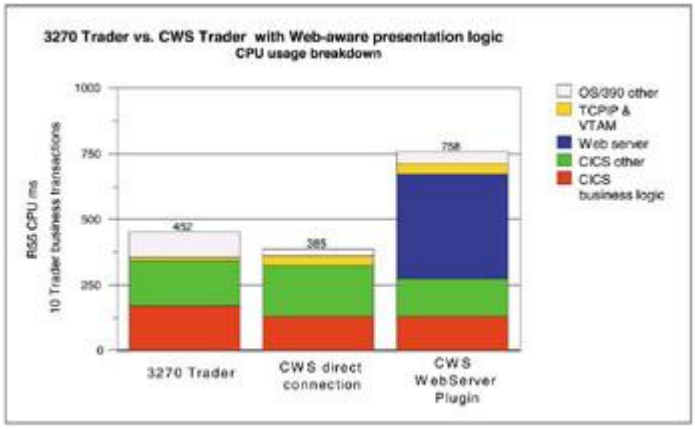


Figure 35: Capacity planning estimates for Trader via CWS

It should be borne in mind when comparing these figures that the costs are based on the average cost of transferring HTTP data over the range of throughputs measured. This cost will decrease as throughput increases, thus reducing the overall CPU usage at higher throughputs; this effect appears to be more pronounced when using the WebServer Plugin than when using a direct connection to CICS Web support.

The estimate also does not include the costs of any HTTP presentation logic apart from the basic cost of building and transmitting the HTTP data stream. In our test to verify our estimation (5.3.3 , "Confirming our estimate" on page 81), we found that the presentation logic costs in our CICS Web-aware version of the Trader application were significantly more than our estimated cost based on transmission of the HTTP data stream.

Chapter 6: SSL with CWS

Overview

In this chapter we summarize how to provide Web access to the business logic of the Trader application, using CICS Web support (CWS) together with new Web-aware CICS presentation logic. We then

present a set of performance studies for various laboratory workloads, and go on to use these figures to perform capacity planning for Web-enablement of the Trader application.

In this chapter we first give a brief overview of the Secure Sockets Layer (SSL) protocol and what the implications are of using it to secure your CICS Web application. We then present laboratory performance figures for using SSL with CICS Web support (CWS) to access CICS applications, both via a direct connection and using the CICS WebServer Plugin. We then go on to use these figures to perform capacity planning for Web-enablement of the Trader application.

Sources of further information on SSL and CICS Web security are:

Chapter 5 ,"TCP/IP Security Overview" of the *TCP/IP Tutorial and Technical Overview* , GG24-3376 (redbook)

Chapter 6, "CWS Security" of the *CICS Transaction Server for OS/390 Version 1 Release 3: Web Support and 3270 Bridge* , SG24-5480 (redbook)

6.1 SSL overview

Since the Internet is so popular and easy to access, it immediately raises security concerns when used as the infrastructure for any sort of electronic communication. A recent U.K. newspaper article stated the findings of the Credit Card research group as follows: "Consumers who pay for goods over the Net are 20 times more likely to fraud than if they pay at a till or over the telephone". (*Guardian Weekly, September 16, Volume 161, No. 12)*

It is generally wise to consider the Internet as a non-secure network, implying that data sent could be read by any person, and that the Web site you are accessing is only that which it claims to be if you have good reason to believe so. The SSL security protocol was designed to address both of these issues.

SSL is a security protocol that was developed by Netscape Communications Corporation, along with RSA Data Security, Inc. SSL provides an addition to the standard TCP/IP socket API that has security implemented within it. Hence, in theory, it is possible to run any TCP/IP application in a secure way without changing the application. In practice, SSL is only widely implemented for HTTP connections as the HTTPS protocol.

The SSL protocol is composed of two layers, the SSL Handshake Protocol and the SSL Record Protocol. The SSL Handshake Protocol provides a protocol for initial authentication of the server and optionally the client, and for the exchange of secret encryption keys to be used by the Record Protocol. The SSL Record Protocol sits below the TCP/IP sockets protocol and provides a means for transferring data using a variety of predefined cipher and authentication combinations.

An HTTPS connection is the protocol used for transmitting HTTP datastreams over SSL connections. A HTTPS connection is initiated by the client Web browser using a special URL that commences ***https*** : instead of ***http:*** . This will establish a secure connection between the Web browser and Web server via SSL. The following chain of events occurs during this process and is illustrated in Figure 36 . Note that this is a highly simplified version of SSL. In reality, it contains numerous other details that counter different types of attack.

1. The client sends a connection request with a *client hello* message, the content of which includes:
    - SSL version number

- A random number
- List of cryptographic options supported by the client (cipher suites)

2. The server evaluates the parameters sent by the *client hello* message and replies with its own *server hello* message. This includes the following information:
   - Server X.509 certificate containing the server's public key
   - SSL version number
   - Session ID
   - Cipher to be used
   - Optional request for a client certificate

3. The client authenticates the server certificate and returns a message containing a random number called the *pre-master secret* key, which is encrypted using the server's public key. If requested, a client certificate with a certificate verify message is also sent.

4. The server decrypts the clients message containing the *pre-master secret* key using the server's private key. The server switches to the cipher specification selected by the client and authenticates the client certificate if requested. The server replies with a finished message.

5. Both client and server generate a master key using a hashing process involving the *pre-master secret key* and random numbers exchanged previously. This is then used to generate secret session keys for the subsequent secret key data encryption.

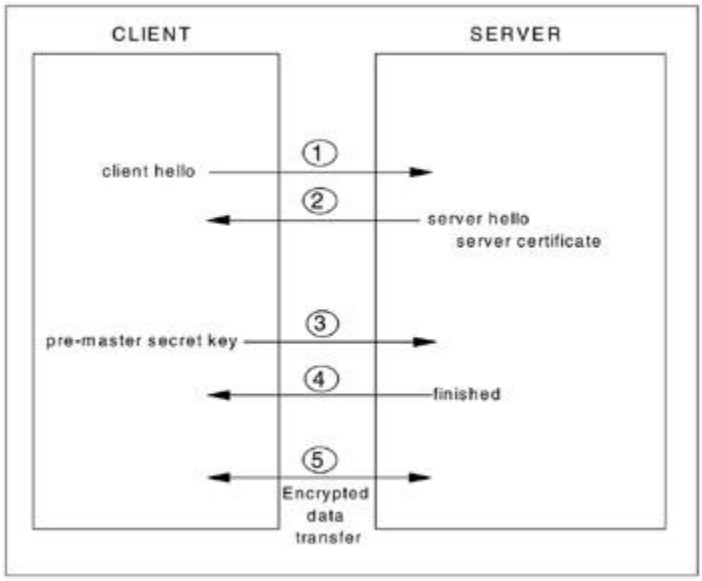


Figure 36: SSL handshake process

The SSL protocol combines the benefits of public/private key (asymmetric) cryptography with those of secret key (symmetric) cryptography. The SSL handshake phase uses public/private key cryptography to authenticate the server (and optionally the client) and to distribute a shared secret key. This secret key is

then used for the encryption of all subsequent transmitted data, and offers the benefit of being much less CPU intensive than public/private key cryptography.

The encryption of data will always have a performance impact; however, using SSL on S/390, this can be minimized in several ways:

- Usage of the S/390 Cryptographic Coprocessor Feature

  The Cryptographic Coprocessor Feature can be used to reduce the CPU costs of SSL data transmission when using the DES or triple DES ciphers, and SSL handshaking when using the RSA PKCS#1cipher. In order to use this hardware feature, the OS/390 Integrated Cryptographic Service Facility (ICSF) has to be installed and operational. ICSF provides a cryptographic application programming interface.

- SSL session ID re-use

  An SSL session can be resumed when a client makes a new HTTP connection; this is achieved by passing a previous session ID to the server for re-use. This is termed "session ID re-use", and the handshake involved is termed a null handshake, as opposed to the full handshake usually incurred. The processing costs of a null handshake are considerably less than those of a full handshake, since the session ID does not have to be re-generated.

- Choice of cipher suites

  SSL offers a choice of different ciphers, and these will have different CPU requirements. Also, most ciphers offer different levels of security by using different length keys. Key length may affect CPU usage of the cipher.

- The use of persistent HTTP connections

  Use of a persistent HTTP connection, whereby a subsequent HTTP connection re-uses a previously opened persistent TCP/IP socket connection, ensures that after the initial SSL handshake, no other handshake is performed until the persistent HTTP connection is broken, which will usually only occur when the HTTP connection is timed out by the server.

The different types of SSL handshakes can be defined as "full", "null", or "none", and will occur when using HTTP as follows:

1. A full handshake will be performed when the client initially establishes the HTTPS connection, since there is no session ID in the client hello message. A full handshake will also be performed when the server decides a submitted session ID is not valid for re-use.

2. A null handshake will be performed when the client establishes an HTTPS connection and includes a session ID for the session to be resumed, and the server decides it is valid for re-use.

3. No handshake will be performed when a new HTTPS request is received from a Web browser via a previously established persistent HTTP connection.

A schematic logic diagram of which SSL handshake is used in which set of circumstances is shown in Figure 37 .

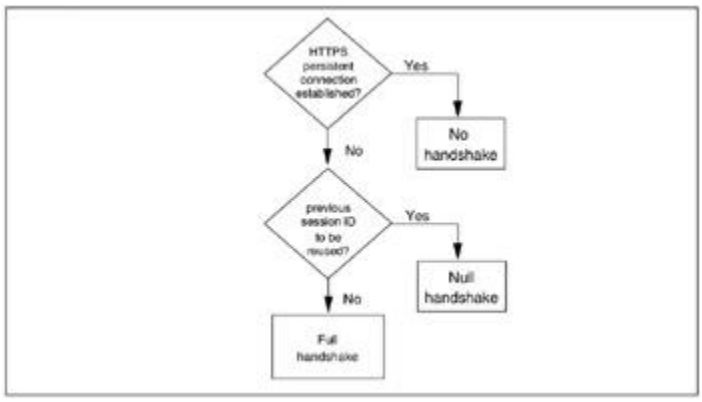


Figure 37: Types of SSL handshakes

For further information on the SSL protocol with examples of SSL handshaking, go to http://developer.netscape.com/tech/security/index.html . To read about client authentication, go to http://home.netscape.com/eng/ssl3 .

In OS/390 V2.7 the SSL protocol was made available as an externalized and integral component of the operating system. CICS TS V1.3 can utilize this support to implement HTTPS connections when establishing a direct connection from the Web browser to the CICS Sockets listener in CICS Web support. The CICS Transaction Gateway (CTG) for OS/390, and the WebSphere application server for OS/390, also utilize OS/390 SSL support. This allows your Web browser clients to communicate securely over the Internet with your target CICS region using either the CICS Web support, CTG applets, or CTG servlets.

6.2 Performance tests using SSL with CWS

In this section, we first present measurements for various types of SSL handshakes, and then we present measurements for encrypted data transmission via CWS SSL support. We investigate the results of using different strength server keys, and we show the cost of using selected ciphers for data encryption, as well as the advantages of using the S/390 Cryptographic Coprocessor Feature for SSL handshaking and SSL data transmission.

6.2.1 Test environment

The test environment was equipped with sufficient hardware (processor, memory, DASD, network bandwidth) to eliminate any constraints. The operating system was OS/390 V2.7, together with CICS Transaction Server V1.3, IBM HTTP Server V5.1, and several PTFs relating to SSL support. Full details of the software levels and parameters in effect during testing are listed in A.2.4 , "CWS with SSL" on page 166 . The test system hardware configuration was the same as that used in Chapter 5 , "CWS with Web-aware presentation logic" on page 65 and illustrated in Figure 28 on page 70 . Additionally, the S/390 Cryptographic Coprocessor Feature was enabled; this consists of dual cryptographic module chips protected by tamper-detection circuitry and a cryptographic battery unit. These coprocessors were dedicated to performing encryption operations for SSL.

6.2.2 Test methodology

For the CWS SSL tests in this chapter, Web browsers were simulated using the Compuware QALoad

product. These were run from two nodes of an AIX SP2 connected via Token Ring emulation over an ATM network to the S/390 processor, as illustrated in Figure 28 on page 70 .

A set of tests were run using a CWS direct connection and using the CICS WebServer Plugin. A range of five throughputs from approximately 15 to 60 Web requests per second were achieved by varying the think time of the simulated Web browsers within the QALoad tool. The number of Web users was set to 70 in all cases. The workload was allowed to settle before a five minute measurement interval was sampled using OS/390 RMF.

The test CICS application was a simple Web-aware CICS program, written in assembler. It used the WEB and DOCUMENT APIs to send a variable number of bytes as specified by the client. All requests used HTTP GETs to invoke the CICS program.

For the SSL handshake measurements, an HTTP GET request was used to request that the CICS program just return 1 byte of data. The use of persistent HTTP connections and SSL session IDs was controlled to test the following SSL handshakes.

- Full handshake using a 1024 bit server key
- Full handshake using a 512 bit server key
- Full handshake using a 1024 bit server key and the S/390 Cryptographic Coprocessor Feature enabled
- Full handshake with client certificates using a 1024 bit server key
- Full handshake with client certificates using a 1024 bit server key and the S/390 Cryptographic Coprocessor Feature enabled
- Null handshake using a 1024 bit server key
- Null handshake using a 512 bit server key

The server key length refers to the size of the public/private key pair used by the server. The size of the server key is specified in the server certificate.

For the SSL data transmission measurements, the same test program was used as in the SSL handshake tests, except the amount of data returned was modified. All the data transmission measurements utilized persistent HTTP connections such that no SSL handshaking was performed for the period of measurement. The CPU usage of workloads transmitting 1 byte, 8 KB and 16 KB were measured using the following ciphers

- RC4-MD5 (40 bit and 128 bit)
- Triple DES
- Triple DES together with S/390 Cryptographic Coprocessor Feature enabled

All the tests with the S/390 Cryptographic Coprocessor Feature and with SSL client certificates were only carried out using a CWS direct connection. The OS/390 Web server does support usage of SSL

client certificates and the Cryptographic Coprocessor Feature, but we did not have time available to test these configurations.

6.2.3 Test results

In this section we present a graphical summary of the SSL performance measurements, in order to highlight the important points and to provide the necessary information to perform our capacity planning estimate. A more detailed performance comparison of the different CICS Web technologies can be found in 8.2 , "Analysis of results" on page 132 .

SSL handshake results

In Figure 38 we show the total OS/390 CPU% usage for our SSL handshake tests, when using the CWS direct connection. The figures plotted are the percentage usage of a single R55 CPU, with a maximum of 500% available. Unprocessed results for the measurements can be found in Table 76 on page 187 through Table 82 on page 189 , in section B.4.1 , "SSL handshakes with a CWS direction connection" on page 187 . The figures marked " *with crypto* " were measured with the Cryptographic Coprocessor Feature enabled.

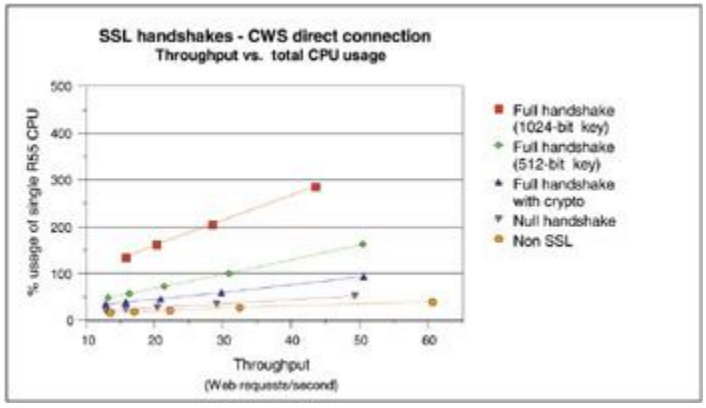


Figure 38: SSL handshakes — CWS direct connection

All handshakes utilized HTTP non-persistent connections; the *non-SSL* handshake is the cost of establishing a non-persistent HTTP connection without SSL. The null handshake costs were found to be the same if using a 512 or 1024 bit server key, and similarly, the full handshake costs with the Cryptographic Coprocessor Feature enabled were found to be the same when using a 512 or a 1024 bit server key; in both cases, only the results for the 1024 bit server key are plotted.

In Figure 39 we show the total OS/390 CPU% usage for our SSL client certificate tests with a CWS direct connection. The figures plotted are the percentage usage of a single R55 CPU, with a maximum of 500% available. All the tests with client certificates used a 1024 bit server certificate. Unprocessed measurements can be found in Table 83 on page 189 and Table 84 on page 189 , in section B.4.2 , "SSL data transmission with a CWS direction connection" on page 190 .

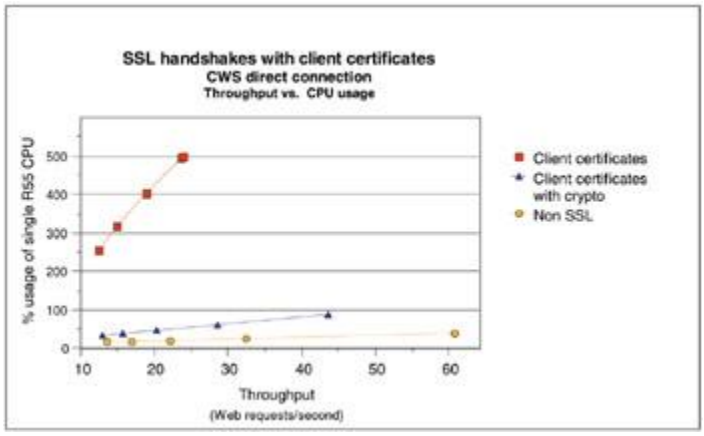


Figure 39: SSL handshakes, client certificates

Figure 39 illustrates the high CPU cost of client authentication, but also shows how the Cryptographic Coprocessor Feature greatly reduces this cost. Usage of the Cryptographic Coprocessor Feature reduced the CPU cost for client authentication to about the same cost as a full handshake with a 1024 bit key using the S/390 Cryptographic Coprocessor Feature, as shown in Figure 38 on page 92 .

We did not measure SSL client certificates or usage of the Cryptographic Coprocessor Feature with the OS/390 Web server and the CICS WebServer Plugin. These features are supported in this configuration, but we did not have the time available to test them.

In Figure 40 we show the total OS/390 CPU% usage for our SSL handshake workloads, using CWS with the CICS WebServer Plugin. The figures plotted are the percentage usage of a single R55 CPU, with a maximum of 500% available. Unprocessed measurements can be found in Table 100 on page 195 through Table 103 on page 196 , in section B.4.3 , "SSL handshakes with the CICS WebServer Plugin" on page 195 . Note that the Cryptographic Coprocessor Feature was not enabled in these measurements.

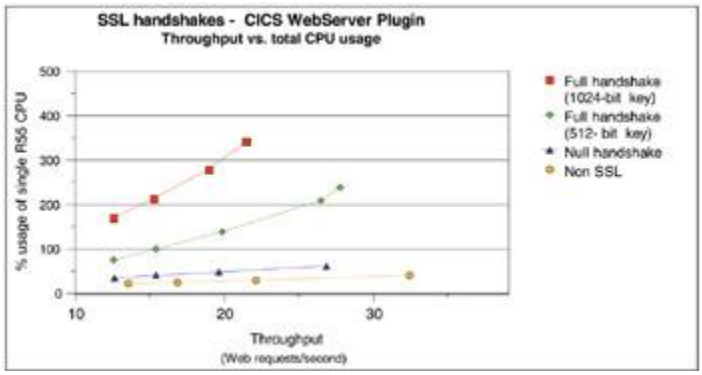


Figure 40: SSL handshakes — WebServer Plugin

Figure 40 shows the same pattern as observed in our previous tests, that using the CICS WebServer Plugin uses somewhat more CPU than using a direct connection to CWS. This difference in CPU usage when using the CICS WebServer Plugin is discussed in 5.2.3 , "Test results" on page 72.

The results for our SSL handshake tests illustrate several important points:

- The Cryptographic Coprocessor Feature reduces the CPU cost of the full handshakes with 1024 and 512 bit keys to the same level, which is around 210% of the cost of establishing a non-SSL connection.

- The Cryptographic Coprocessor Feature also reduces the CPU cost of the 1024 bit full handshake with client certificates to the same level as without client authentication.

- When the Cryptographic Coprocessor Feature is not used, the smaller 512 bit server key reduces the CPU cost of the full handshake by about three fold.

- The null form of the handshake reduces the CPU cost of the SSL handshake to around 140% of the cost of establishing a non-SSL connection.

SSL data transmission results

In Figure 41 we show the total OS/390 CPU% usage for the different SSL data transmission ciphers at a range of throughputs for both the CWS direct connection and the CICS WebServer Plugin. 8 KB of data were sent from a Web-aware program using the WEB API. The actual data for these measurements can be found in Table 88 on page 191 through Table 99 on page 194, and Table 104 on page 197 through Table 106 on page 197, in section B.4 , "CWS with SSL" on page 186. The figures marked *triple DES + crypto* used the Cryptographic Coprocessor Feature; this can be used with either a CWS direct connection or the CICS WebServer Plugin.

The illustrated test results for the RC4-MD5 cipher are simplified because the results were found to be the same if using a 40 bit (international) or a 128 bit (US domestic) key. This is because they both pass a 16 byte key length into the encryption algorithm. The difference is that 40 bit encryption uses "salted" (unencrypted random data) as part the of key-block used to generate the 16 byte key. This reduces the strength of the encryption, not the path length.

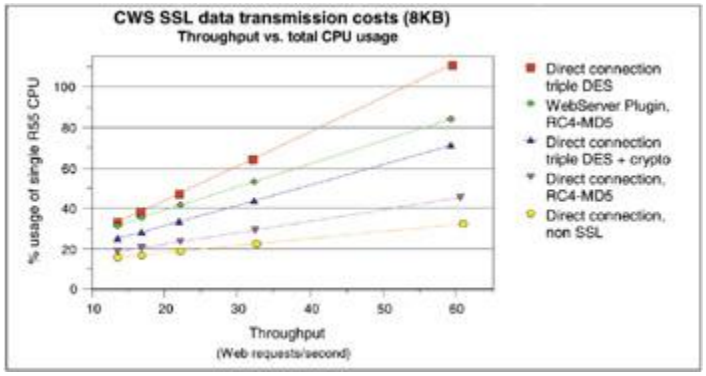


Figure 41: CPU usage for 8 KB SSL data transmissions

The graph highlights the following points:

- The Cryptographic Coprocessor Feature provides a reduction of CPU usage for SSL data transmission of about 50% when using the triple DES cipher.

- The RC4-MD5 cipher (40 or 128 bit) uses less CPU for data transmission than the triple DES cipher.

- The CPU cost of SSL data transmission is significantly less expensive than the cost of SSL handshaking, as reported in our "SSL handshake results" on page 92.

6.3 Capacity planning for SSL with CWS

In this section we will now perform an estimation of the OS/390 CPU usage for the Trader application when Web-enabled via the CWS, using a CWS direct connection and Web-aware presentation logic secured with the CWS SSL support.

SSL CPU estimation

The data in this chapter should only be used in conjunction with CICS Web support and CICS TS V1.3; it should not be used to estimate CPU usage for any other IBM products which may use different implementations of SSL.

6.3.1 Capacity planning methodology

We have already estimated the CPU costs for Web-enabling the Trader application using CWS and a direct connection to a Web-aware program; this can be found in section 5.3.2.1 , "CWS direction connection estimation" on page 78 . To estimate the CPU usage of the same scenario but with SSL, we need to calculate the delta cost for the SSL handshakes and the delta for SSL data encryption.

We will assume that persistent HTTP connections are being used in our application. Thus each Trader business transaction will incur one full SSL handshake on the first HTTP request in the business transaction, and the subsequent encryption cost of data sent from CICS to the Web browser.

Note that this is a simplistic model, and it is possible that an application such as Trader could incur greater or indeed lesser costs. The circumstances where CPU costs could be less are as follows:

- The persistent HTTP connection does not expire across the lifetime of several business transactions.

  If the persistent HTTP connection time-out value does not expire (defined in the SOCKETCLOSE parameter of the CICS TCPIPSERVICE definition), then a subsequent HTTPS request from a previously attached Web browser will not incur any SSL handshake costs).

- The persistent HTTP connection is broken, but the SSL session ID time-out value has not expired.

  If the SSL time-out value (specified in the CICS SIT as SSLDELAY) has not expired when a request for a subsequent HTTPS connection from a previously attached Web browser is received, then a null handshake will be performed, as opposed to a full handshake.

The circumstances where CPU costs could be greater are as follows:

- More Web browser clients are connected to one CICS region than can be supported by the number

of CICS SSL TCBs.

Once the number of attached SSL clients exceeds the number of defined SSL TCBs in one CICS region, then subsequent HTTP requests will "steal" the least previously used SSL TCB. Since there is a one-to-one affinity between an HTTPS session and an individual SSL S8 TCB, then TCB stealing will cause Web browsers that send a subsequent HTTPS request to CICS to incur the additional cost of an SSL null handshake and the creation of a new HTTP connection. It is possible to spread larger numbers of Web browsers across multiple CICS "Web Owning" regions by using TCP/IP port sharing or TCP/IP dynamic DNS to workload-balance HTTP or HTTPS requests across multiple CICS regions. When using CWS and the WebServer Plugin, the "TCB stealing" situation does not occur, due to the different internal design of the OS/390 Web server SSL support.

- Client certificates are used.

  If SSL client certificates are used, then further CPU costs may be incurred, both within the SSL routines in the server (CICS) and on the Web client. As shown in our tests, the costs of SSL handshaking can be minimized using the S/390 Cryptographic Coprocessor Feature.

- A large amount of data is received as well as sent.

  Trader receives only a very small amount of data, and the encryption cost of the HTTP headers is already included in our SSL data encryption measurements. However, for applications that receive large amounts of data, this cost should be factored into the capacity planning estimate.

However, we will assume none of these conditions apply to Trader, and that there are sufficient CICS SSL TCBs to support persistent HTTP connections from all attached Web browsers.

SSL handshake delta

To calculate the CPU delta of the SSL full handshake with a 1024 bit server key using hardware cryptography, we shall calculate the average CPU cost per request for this handshake and subtract this from the average CPU cost for establishing a non-SSL, non-persistent HTTP connection. This calculation is illustrated in Table 12 . The averages were calculated from the *CPU ms/request* figures in Table 76 on page 187 and Table 79 on page 188 .

Table 12: SSL handshake delta

	Non-SSL (CPU ms)	**SSL full handshake (CPU ms)**	**Delta per SSL full handshake (CPU ms)**
Average CPU ms per request	9.7	22.4	12.7

SSL data transmission delta

Similarly we can calculate the delta SSL cost for the data transmission. We shall calculate the average CPU cost per request for the 8 KB data transmission and subtract from this the average CPU cost per request for a non-SSL data transmission. This calculation is illustrated in Table 13 . The averages were calculated from the data in Table 86 on page 190 and Table 89 on page 191 .

Table 13: SSL data transmission delta

	Non-SSL data transmission (CPU ms)	RC4-MD5 data transmission (CPU ms)	Delta for RC4-MD5 8 KB data transmission (CPU ms)	Delta per KB (CPU ms)
Average CPU ms per request	8.5	10.6	2.1	0.3

6.3.2 Capacity planning estimate

In this section we will now perform an estimation of the total OS/390 CPU usage for the Trader application when Web-enabled via CWS, and a direct connection using CICS SSL support. See Table 14 . We will assume minimal SSL costs are incurred as follows:

- CWS direct connection.
- SSL handshake: 1024-bit server key utilizing the OS/390 Cryptographic Coprocessor.
- SSL data transmission: RC4-MD5 cipher with a 40 or 128 bit key.
- No TCB stealing occurs within the CICS region.
- A persistent HTTP connection is used for the duration of the business transaction.

The data sizes used in the trader application are already documented in Chapter 6 , "SSL with CWS" on page 85 , along with the estimated CPU Usage for Trader at a throughput of 10 business transactions per second. We will re-use this data and add to it the cost of a full SSL handshake on the first request as given in Table 12 , and the subsequent SSL encryption costs from Table 13 .

Table 14: CPU usage per Web request with SSL and a CWS direct connection

Step	Data SENT (bytes)	CWS (CPU ms)	Handshake (CPU ms)	Data transmission (CPU ms)	TRADERBL (CPU ms)	Total (CPU ms)
1	2007	6.9	12.7	0.6	0	20.2
2	1923	4.6	0	0.6	4.1	9.3
3	2406	4.7	0	0.6	4.1	9.4
4	2406	4.7	0	0.6	4.8	10.1
5	1342	4.6	0	0.4	0	5.0
Total		**25.5**	**12.7**	**2.8**	**13.0**	**54.0**

Since each Trader business transaction comprises five Web requests, we can use this figure of 54 CPU ms to calculate the CPU cost of running the Web-enabled trader application at a throughput of 10 business transactions per second (or 50 Web requests per second) as follows:

CPU cost of Trader with SSL

```
Total CPU ms = 54 * 10 = 540 CPU ms
```

Looking at the SSL cost for the same 10 business transactions, we can now calculate:

CPU cost of SSL for Trader

```
SSL CPU ms = (12.7 ms + 2.8 ms) * 10 = 155 ms
```

of this 155 ms

```
SSL handshake% = 127/155 = 82%
SSL data transmission%= 28/155 = 18%
```

If we now deduct the known cost of 130 CPU ms for the CICS business logic in TRADERBL from this 540 CPU ms to give 410 CPU ms, we can estimate how much should be allocated to the different OS/390 components. We do this by using the relative proportions reported for each component in our CWS 8 KB SSL data transmission figures given in Table 89 on page 191 , using the throughput of 60.57, which is the closest to our defined rate of 50 Web requests per second. This calculation is illustrated below in Table 15 .

Table 15: CPU percentage breakdown for CWS direct connection with SSL

Component	Percentage of total per component	CPU usage for 10 business transactions (CPU ms)
CICS TRADERBL		130 ms
CICS other	74%	303 ms
TCP/IP & VTAM	10%	41 ms
OS/390 other	16%	66 ms
Total		540 ms

6.4 Trader performance comparison

Using the figures in Table 15 , we have compared the cost of the CWS Web-enabled Trader application with the cost of the SSL version; this is illustrated in Figure 42 . We assume the use of a 1024-bit server key utilizing the OS/390 Cryptographic Coprocessor, the RC4-MD5 cipher. The figures plotted are the total CPU ms used on an 9672-R55, for running 10 Trader business transactions (which equates to 50 Web requests or CICS tasks).

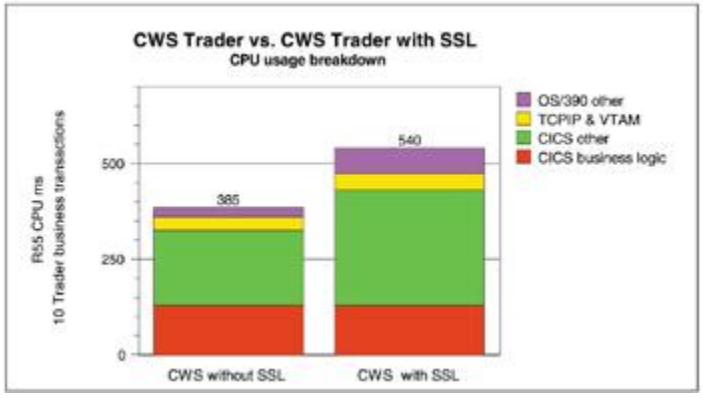


Figure 42: Capacity planning estimates for Trader via CWS with SSL

The graph illustrates the cost of enabling SSL security with CWS using a direct connection. The largest proportion of this cost is incurred in the CICS address space; thus there is a resulting increase in the "CICS other" CPU usage. Note that the SSL costs represented are minimal SSL costs, and you should refer to 6.3.1 , "Capacity planning methodology" on page 96 for further information on how the SSL costs could be different for your particular environment. It would also be possible to use SSL security if using the CICS WebServer Plugin. We do not present data for this configuration, but in this case, the additional CPU cost would be incurred in the Web server address space.

Chapter 7: The OS/390 CTG

Overview

In this chapter we first discuss how to Web-enable the Trader application using the OS/390 CICS Transaction Gateway (CTG). We then present the results of our performance studies of CTG applets and servlets using simple test applications. We use the results of these performance tests to build a capacity planning methodology for estimating the CPU usage when using the OS/390 CTG. Lastly we calculate the CPU usage of the Trader application if it were to be Web-enabled using the OS/390 CTG.

CTG V3.1

Version 3.1 of the CTG has implemented significant performance enhancements over version 3.03 of the CTG and its predecessor the CICS Java Gateway v2. However, because of this, if the OS/390 CTG V3.1 is used with CICS Transaction Server V1.2, it requires the fix for APAR PQ31270 to be applied to CICS. This does not apply if using CICS Transaction Server V1.3

7.1 Converting the Trader application

In this section we discuss the Trader application and how to convert it from a legacy 3270 application to a modern Java-based application using the CTG. Refer to Chapter 3 , "The 3270 green screen Trader

application" on page 37 for more details on the Trader application. This task is eased because the original Trader application has separate business and presentation logic. The CICS business logic in the program TRADERBL can be invoked directly using the CTG External Call Interface (ECI) Java methods.

The CTG provides the ability for Java client programs to access CICS in three different architectures; applets, servlets, or stand-alone Java applications. We will discuss the applet and the servlet options, as there is no specific architecture for Java applications. Refer to 1.3 , "CICS Transaction Gateway" on page 14 for a description of the applet architecture and the servlet architecture.

7.1.1 Basic application structure

We now give a brief overview of how the application structure would look if the Trader application was Web-enabled, using Java applets and Java servlets.

Using the applet architecture

If a Java applet architecture was used to Web-enable the Trader application, the presentation logic would be implemented within the Java applet, from which ECI calls would be made to the business logic within CICS. The flow of requests in one Trader business transaction is illustrated in Figure 43 and explained below.

1. An HTTP request is sent from the Web browser to the Web server for an HTML page containing a tag for the CTG Java applet.

2. The Web server returns the HTML page with the embedded applet tag.

3. The browser requests download of the specified applet.

The applet is invoked within the JVM of the Web browser and now runs the rest of the application; this would be as follows:

4. The applet opens a network connection to the CTG Java gateway application on OS/390 using the JavaGateway.open() method.

5. The applet builds an HTML page, and the user enters his userid and password into the presented display. The applet constructs an ECI request with a COMMAREA of 372 bytes containing the userid and the password. Then the applet, using the JavaGateway.flow() method, calls the TRADERBL program in CICS passing the COMMAREA. The ECI request is flowed to the CTG Java gateway application, which passes it on to CICS using the External CICS Interface (EXCI) protocol. The CICS business logic program TRADERBL returns the company list in the COMMAREA, which is passed back to the applet by the CTG.

6. The applet constructs an ECI request with a COMMAREA of 372 bytes containing the company selection. Then the applet, using the JavaGateway.flow() method, calls the TRADERBL program in CICS passing the COMMAREA. TRADERBL returns the quote in the COMMAREA.

7. The applet constructs an ECI request with a COMMAREA of 372 bytes containing the number of shares to buy. Then the applet, using the JavaGateway.flow() method, calls TRADERBL passing this COMMAREA. TRADERBL returns the number of shares bought in the COMMAREA. The

applet then updates the quote and displays it.

8. The applet closes the connection to the CTG Java gateway application using the JavaGateway.close() method.

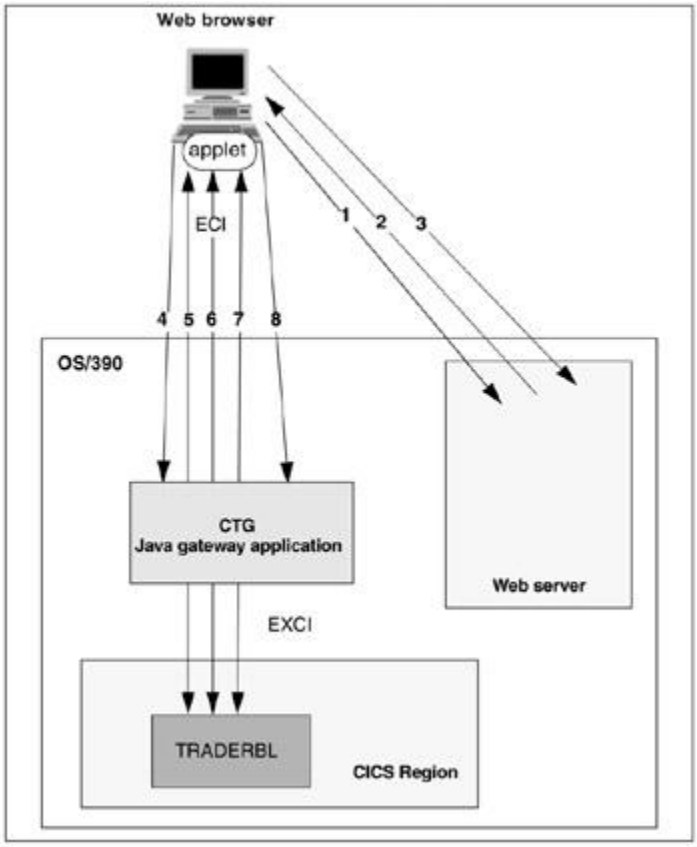


Figure 43: Trader application flow using the CTG applet architecture

It is important to note that the number of calls to the CICS business logic per Trader business transaction is now three, and thus the number of CICS tasks per business transaction is also three.

Using the servlet architecture

If a Java servlet architecture was used to Web-enable the Trader application, the presentation logic would be part of the servlet or a Java Server Page (JSP), while the business logic remains unchanged inside the CICS application. A servlet is controlled by and runs within the JVM of the servlet engine such as WebSphere Application Server.

The basic structure of the Trader application Web-enabled using a servlet architecture is illustrated in Figure 44 , and the flows would then be as follows:

1. An HTTP request is sent from the Web browser to the Web server for the relevant servlet.

2. The Web server invokes the servlet.

The servlet is now in control and runs the rest of the application. We assume the servlet was loaded at

the start-up of the Web server, and also that the local connection to the CTG was established at that time. Further processing in case of the Trader application would be as follows:

3. The servlet builds an HTML page for the signon display and sends this to the Web browser.

4. A userid and password is entered on the HTML page and the Web browser sends this to the servlet. The servlet uses the CTG ECI methods to build an ECI request. Then the servlet, using the JavaGateway.flow() method, calls the CICS program TRADERBL, passing a COMMAREA. TRADERBL returns the company list in the COMMAREA, and the servlet formats the company list display and sends the HTML to the Web browser.

5. A company is selected. The servlet reads the company selection, and uses the CTG Java methods to build an ECI request with a COMMAREA containing the company selection. Then the servlet, using the JavaGateway.flow() method, calls TRADERBL, passing this COMMAREA. TRADERBL returns the quote in the COMMAREA, and the servlet formats the quote display and sends the HTML to the Web browser.

6. The option for buy shares is entered. The servlet reads the buy share option, and builds a COMMAREA containing the number of shares to buy. Then using the JavaGateway.flow() method, the servlet calls TRADERBL, passing the COMMAREA. TRADERBL returns the number of shares bought in the COMMAREA to the servlet; the servlet then builds an HTML page and sends this to the Web browser.

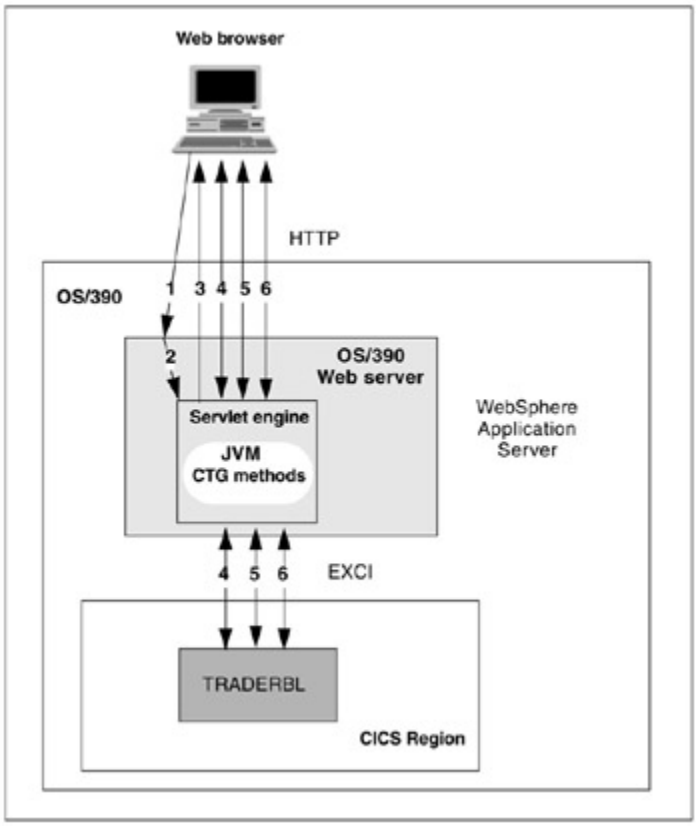


Figure 44: Trader application flow using the servlet architecture

It is important to note that the number of calls per business transaction to the CICS business logic is now

only three, and that the number of Web requests per business transaction is now four.

7.1.2 Performance considerations

We will now describe the major issues which are likely to affect the performance of CTG applet and servlet designs.

7.1.2.1 Using the applet architecture

When using the applet architecture, all the new Java presentation logic will be executed in the Web browser. The following characteristics should be considered:

Client CPU usage

The performance of the Java Virtual Machine (JVM) on the Web browser will affect the performance of the applet solution, since the new presentation logic is implemented within the JVM on the client machine's Web browser. This will not impact the server CPU usage and will not be considered in our studies.

CTG thread usage

The CTG Java gateway application, which is used for the applet architecture, is itself a sophisticated multi-threaded Java application. It can handle multiple requests simultaneously and has a set of properties (configured in the CTG.INI file), to allow requests to be queued and timed-out if necessary. Within this file two pools of threads are can be configured, the ConnectionManager threads and the worker threads. For each connected applet client, one ConnectionManager thread is used in the Java gateway application, and is held until the client issues a disconnect using the JavaGateway.close() method. In order for an ECI call to be performed via an allocated ConnectionManager thread, a thread must be allocated from the worker thread pool for the duration of the ECI request. This relationship is summarized in Figure 45 .

Thus the ConnectionManager threads limit the maximum number of connected Java applets, while the worker threads limit the number of concurrent ECI calls that can be issued by these attached clients. The initial and maximum numbers of these ConnectionManager and worker threads are set in the CTG.INI file. Requests can be timed out if a ConnectionManager or worker thread does not become available within a specified time, or if the gateway detects that a client is idle or is not responding. Further details are given within comments in the CTG.INI file.

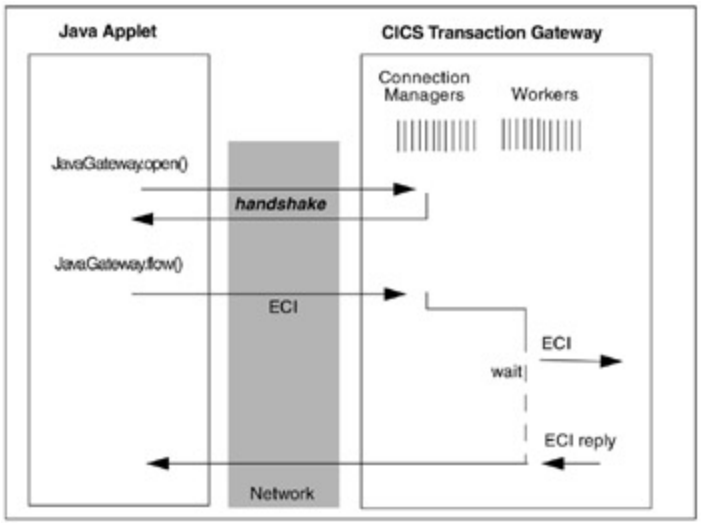


Figure 45: CTG threading model

Network I/O

The bandwidth of the network is of primary importance, and the network protocol used to connect from the Web browser to the CTG can also be an important factor influencing overall performance and scalability of the solution. If this network runs over a public network such as the Internet, then you may not be able to control the bandwidth or availability of this network. The performance of the network will be affected by:

- The size of the applet downloaded from the Web server
- The size of the data passed in an applet ECI COMMAREA
- The number of ECI requests made per business transaction

The CICS COMMAREA is passed across the network from the applet via the CTG Java gateway application to the CICS server and back. You should always try to design your application so it has the minimum number of data flows from the Web client through to the CICS server. It is also possible both to truncate or compress the data flowed through the network from the applet to the CTG Java gateway application; further details are discussed in 8.2.5 , "CICS Transaction Gateway" on page 140 .

7.1.2.2 Using the servlet architecture

When using the servlet architecture, the new Java presentation logic will be executed in the OS/390 Web server address space. The following characteristics should be considered.

Server processing

The servlet architecture places more workload on the S/390 running the Web server as compared to the applet architecture, because the presentation logic runs within the servlet. Thus the performance of the OS/390 JVM and the OS/390 Web server are key ingredients in the performance of the servlet architecture. For hints and tips in this context, refer to the *IBM WebSphere Troubleshooter for OS/390* , which you can find at: http://www.s390.ibm.com/nc/wsphere.html

Java design

The design of the Java logic in your servlet will be a key factor in the overall performance of a servlet solution, since the presentation logic is implemented within the servlet. One of the key factors in the performance of your Java presentation logic is likely to be the cost of manipulating datastreams. Thus in your Java logic you should reduce the amount of parsing of the CICS COMMAREA. Also, all our performance used the basic Java classes provided by the CTG. If you decide to use the Common Connector Framework (CCF) CICS classes as provided by Visual Age for Java, you should quantify any additional costs involved since the CCF classes use a higher level of abstraction than the CTG basic Java classes.

CTG connection re-use

The connection from the servlet to the CTG is created by using the open() method of the JavaGateway constructor. When designing a servlet this should usually be a "local" connection to give the best performance. The CTG local protocol signifies that the CTG will use the Java Native Interface(JNI) to invoke procedures in the local EXCI shared library provided by CICS.

Since servlets run within multiple threads of the servlet JVM engine, a servlet design is multi-threaded. These multiple threads can re-use the CTG connection created by the open() method of the JavaGateway constructor. For best performance you should ensure that this connection is initialized just once in the servlets init() method, and then re-used during the life of the servlet. A good example of how to implement a multi-threaded servlet with the CICS Transaction Gateway is described in the CICS Support Pack CA89 at http://www.software.ibm.com/ts/cics/txppacs , and further details are given in the redbook *Revealed! Architecting Web Access to CICS* , SG24-5466.

GUI design

If you want to build a complex HTML GUI for your Web users, then you should consider that, in this case, the servlet architecture may cause a large increase in network traffic. This is because every HTML page is built by the servlet and has to be sent from the Web server to the Web browser in every interaction.

Network I/O

When using the servlet architecture there are two different network transmissions: one between the Web browser and the servlet, and the other between the Web server and CICS. The flow from the Web browser to the servlet is across a network and should be reduced as much as possible. The flow from the servlet to CICS will be cross memory or cross coupling facility, and so is of less concern.

However, you can use this design to your advantage by implementing some new business logic in the servlet which can make multiple calls to CICS before building the HTML presentation output. This would enable you to reduce the flows from the Web browser to the Web server, and may make the servlet architecture attractive as an Internet solution. Note that the CTG setCommareaOutboundLength() method is not designed for servlet usage, since this only affects the data stream from the Java application to the CTG.

7.2 Performance tests using CTG Java applets

In the following section we show the results of our performance tests of Java applets and the OS/390

CTG. You should be aware that the test scenarios and applications used were simplified in order to quantify the configuration under analysis; the application tested was not a real life application such as Trader.

7.2.1 Test environment

The test environment was equipped with sufficient hardware (processor, memory, DASD, network bandwidth) to eliminate any constraints. The operating system was OS/390 V2.7. We used the CICS Transaction Gateway for OS/390 V3.1, together with CICS TS V1.3, JDK V1.1.8, WebSphere Application Server V1.1, and the OS/390 IBM HTTP Server V5.1. The test environment is illustrated Figure 46 and full details of the software levels and parameters in effect are listed in A.2.5 , "CICS Transaction Gateway" on page 168 .

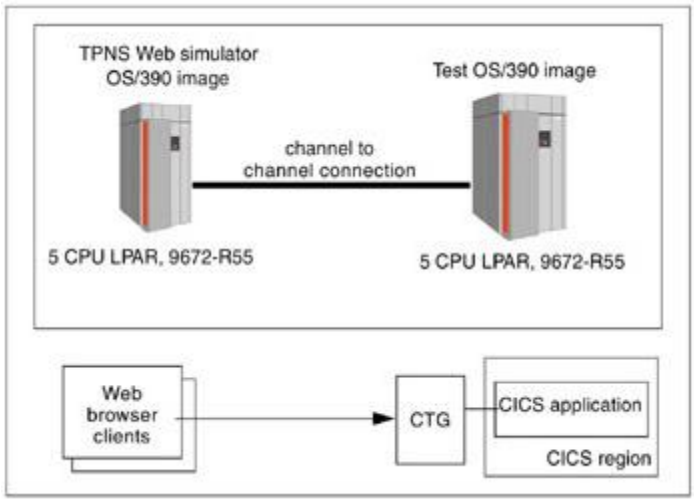


Figure 46: CTG applet test environment

7.2.2 Test methodology

The applet workloads were emulated using TPNS; this was achieved by capturing the network flows of a sample CTG Java applet and then replaying them at different throughputs. The TPNS driver was running on a separate 9672-R55 processor within the sysplex so as not to interfere with the test OS/390 image.

The think time was set to different values, and the workload allowed to settle before a five minute measurement interval was sampled using the OS/390 RMF feature. This process was repeated for different think times to obtain figures for five throughput rates from approximately 30 up to 100 Web requests per second. All our applet tests used 500 simulated Web browser clients.

Our applet tests used a simple CICS COMMAREA based application. This application was a minimal application that merely modified and returned the COMMAREA sent by the client. Note that the complete COMMAREA was transmitted from the applets, through the CTG Java gateway application, into CICS, and back again. You may be able to significantly reduce network I/O by using methods to truncate the COMMAREA, refer to "Applet data transmission" on page 140 for further details.

The CTG supports four network protocols for connectivity from an applet to the CTG Java gateway application, TCP/IP sockets, HTTP, and secure versions of these, namely SSL and HTTPS. We used

only the TCP/IP and HTTP protocols in our tests, and you should quantify the additional costs of using SSL or HTTPS if you have a need to use these. Note that our tests with SSL in Chapter 6 , "SSL with CWS" on page 85 only apply to CICS Web support.

We ran a wider range of tests using the TCP/IP protocol and analyzed the effect of the following variables:

- The cost of opening the network connection from the applet to the CTG Java gateway application
- Re-using the connection from the applet to the CTG across multiple ECI calls
- Increasing the COMMAREA size in ECI requests from 100 bytes to 16KB
- Workload balancing using multiple CTG Java gateway applications

Note that our applet measurements do not include any CPU usage when downloading the applet from the Web server to the Web browser.

7.2.3 Test results

In this section we present a graphical summary of the performance measurements, in order to highlight the important points and to provide the necessary information to perform a capacity planning estimate of an application such as Trader. Comparison of the results of the different Web technologies can be found in 8.2 , "Analysis of results" on page 132 .

First of all, we analyzed the cost of ECI calls using different network protocols from the applet to the CTG Java gateway application.

Using the CTG HTTP protocol

For this situation, we measured the CPU cost of sending data using the HTTP protocol. The results are shown in Figure 47 . The figures plotted are the percentage usage of a single R55 CPU, with a maximum of 500% available. The size of the COMMAREA for these measurements was 100 bytes and the connection from the applet to the CTG Java gateway application was not re-used, that is, the cost of each ECI call includes the cost of opening and closing the HTTP connection from the applet to the CTG Java gateway application. Refer to Table 115 on page 202 for the detailed set of measurement data.

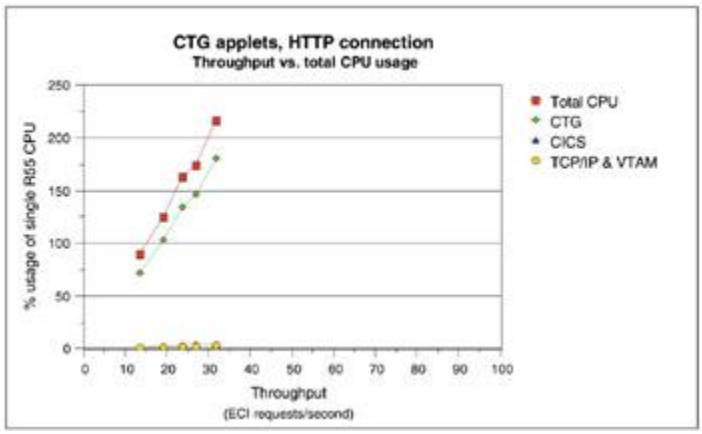


Figure 47: CPU usage of CTG applets, with an HTTP connection

These figures show that the majority of the CPU cost is incurred in the CTG address space, and that the cost within CICS is and TCP/IP is minimal.

Using the CTG TCP/IP protocol

In Figure 48 we show a set of measurements illustrating the CPU% usage on an R55 for an ECI workload, when using the TCP/IP protocol from the applet to the CTG Java gateway application. The figures plotted are the percentage usage of a single R55 CPU, with a maximum of 500% available. The size of the COMMAREA for these measurements was 100 bytes, and the connection from the applet to the CTG Java gateway application was not re-used. Refer to Table 107 on page 198 for the detailed measurement data.

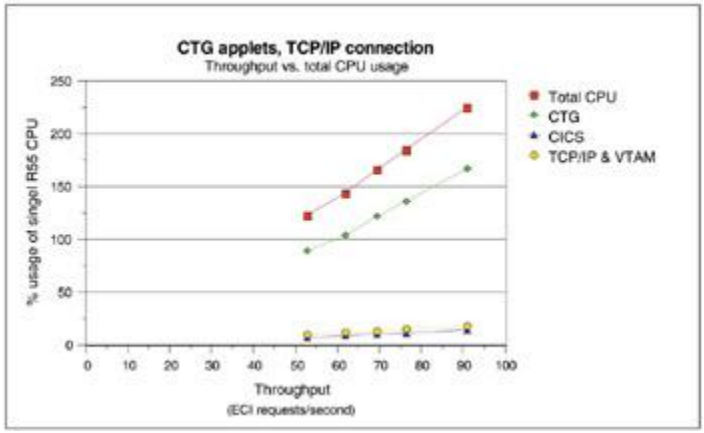


Figure 48: CPU usage of CTG applets, with a TCP/Ip connection

Comparing these measurements with the TCP/IP protocol to those with the CTG HTTP protocol in Figure 47 on page 114 , it is can be seen that when using the TCP/IP protocol, approximately 2.8 times less CPU per call is used, and that this reduction is found principally in the CTG address space.

Cost of making the applet TCP/IP connection

A set of measurements was conducted to understand the CPU usage when opening and closing the TCP/IP connection from the client applet to the CTG Java gateway application. This event is triggered in the CTG applet code using the JavaGateway.open() and JavaGateway.close() methods. Measurements were compared for ECI requests that did and did not re-use the TCP/IP connection from the applet to the CTG Java gateway application. The total OS/390 CPU% usage for these measurements is illustrated in Figure 49 . The figures plotted are the percentage usage of a single R55 CPU, with a maximum of 500% available. Refer to Table 108 on page 199 , Table 107 on page 198 and Table 115 on page 202 for the detailed measurement data.

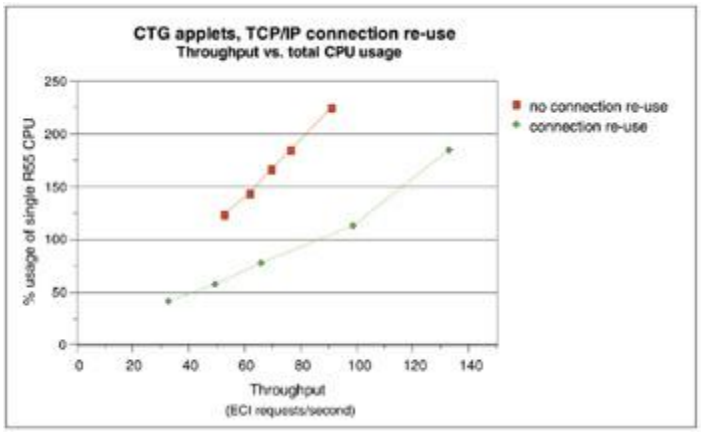


Figure 49: CPU usage of CTG applets making TCP/IP connection

This graph shows the efficiency of re-using the CTG applet connection when making multiple ECI calls from the applet to CICS. Using these figures we are able to calculate the CPU cost of opening and closing a CTG applet TCP/IP connection. To do this we calculated the average CPU cost per ECI request for the workload that re-used the connection, and subtracted this from the average CPU cost per ECI request for the workload that did not re-use the connection. This gave us a figure of 10 CPU ms to open and close a CTG applet TCP/IP connection, which we will use later in our capacity planning methodology.

Increasing COMMAREA size, and the TCP/IP protocol

The principal quantifiable factor affecting CPU usage after having made the connection from the applet to the CTG Java gateway application will be the amount of data transmitted in the COMMAREA when making an ECI call. We measured CPU costs for COMMAREA sizes varying from 100 bytes to 16KB bytes in our tests. In Table 16 we have calculated the average cost over our different throughputs for each ECI COMMAREA size; this data is plotted in Figure 50 . The actual measurements for these results can be found in Table 108 on page 199 through Table 113 on page 200 . All these measurements were conducted with one CTG Java gateway application and 500 clients, and re-used the CTG TCP/IP connection.

Table 16: CPU cost per ECI call with increasing COMMAREA sizes

ECI COMMAREA (bytes)	Average total CPU per ECI call
100	12.4 ms
1,000	14.3 ms
2,000	18.0 ms
4,000	19.6ms
8,000	21.8 ms
16,000	27.4 ms

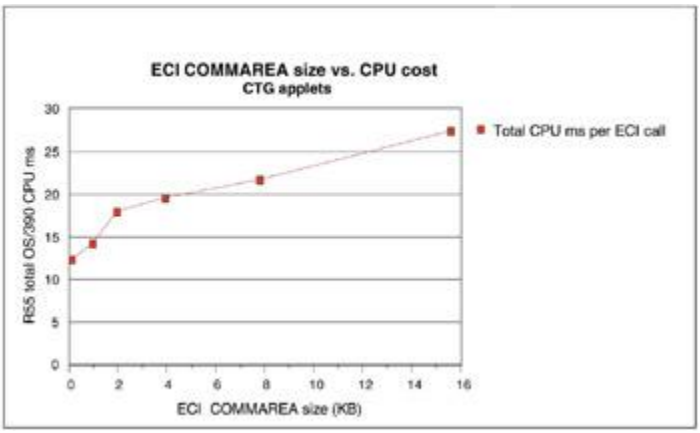


Figure 50: CPU cost of varying CTG applet ECI COMMAREAs

Multiple CTG address spaces and the TCP/IP protocol

In Figure 51 we show a set of measurements conducted using multiple CTG Java gateway application address spaces. Refer to Table 114 on page 201 for the detailed measurement data. In this scenario we spread the workload across four CTG address spaces using the OS/390 eNetwork Communications Server TCP/IP port sharing feature. This allows multiple address spaces to listen on the same port number, thus providing for inbound IP requests to be workload balanced across these address spaces. This has the affect on the CTG of reducing the number of threads used per address space. The figures plotted are the percentage usage by all four CTG address spaces of a single R55 CPU, with a maximum of 500% available. The COMMAREA size used was again 100 bytes, and the TCP/IP connection was re-used.

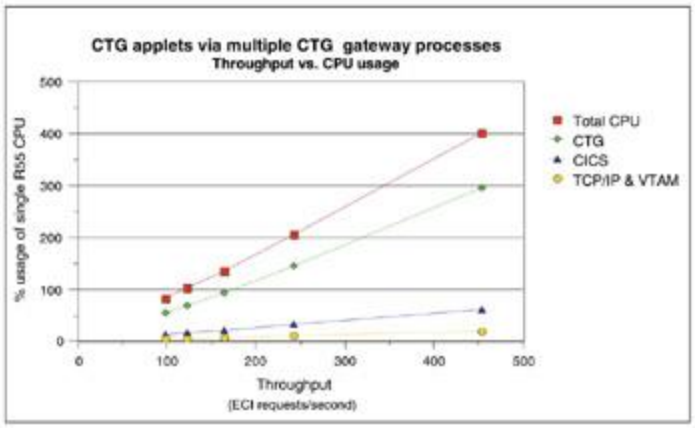


Figure 51: CPU usage of CTG applets using multiple CTG address spaces

By comparing the measurements in Figure 51 for multiple CTG address spaces to those with just one address space (Figure 48 on page 115), it can be seen that using multiple address spaces greatly increases the scalability of the CTG. This is due to the fact that reducing the number of threads per address space reduces the CPU cost per ECI call and thus increases the overall efficiency of the CTG, allowing higher throughputs to be reached.

7.3 Capacity planning for CTG Java applets

In this section we use the results of our previous performance tests to create a capacity planning methodology for estimating the CPU usage of a Web-enabled CICS application using the OS/390 CTG applets. We then use this methodology to estimate the CPU usage when the Trader application is Web-enabled using CTG applets.

7.3.1 Capacity planning methodology

As illustrated in Figure 43 on page 105 , our Web-enabled applet design for Trader has an initial call to open the connection from the applet to the CTG Java gateway application, followed by three ECI calls to the TRADERBL CICS application, using a COMMAREA size of 372 bytes — thus giving a throughput of 30 CICS tasks per second for our defined 10 business transactions per second.

The presentation logic will be implemented in the Java applet on the Web browser client, and as such is not included as part of our capacity planning estimation.Thus the total OS/390 CPU costs per business transaction running the Java applet Web-enabled Trader will be:

1. Cost of one request to open and close the applet CTG TCP/IP connection

2. Cost of three 372-byte COMMAREA ECI calls which re-use the TCP/IP connection

3. Cost in CICS of the requests to the business logic in TRADERBL

The cost of opening and closing a TCP/IP connection from a Java applet to the CTG are already known from the data in Figure 49 on page 116 as 10 CPU ms per request.

The CPU cost for transmitting a given amount of data in an ECI COMMAREA can be calculated from the data in Figure 50 on page 117 , by extrapolating from the two closest measured COMMAREA sizes. We did not plot a linear fit equation for this data, since it can be seen that the costs do not increase in a linear fashion.

The CPU cost of invoking the business logic in TRADERBL are already documented in Table 2 on page 45 . This cost will be 13 CPU ms per business transaction, since when using our CTG applet architecture, only three calls are made to the CICS business logic

7.3.2 Capacity planning estimate

Using our capacity planning methodology we can estimate the OS/390 CPU usage when Web-enabling the Trader application via the applet architecture:

1. Cost to open/close the applet CTG TCP/IP connection:

   10 CPU ms per request

2. Cost of three 372 byte COMMAREA ECI calls:

   3 * (12.4 + (((372-100)/(1000-100)) * (14.3-12.4))) = 39 CPU ms

3. Cost in CICS of TRADERBL:

13 CPU ms

CPU = total CPU consumed in OS/390 R55 LPAR Throughput is the number of ECI or Web requests per second

Thus the total is 10 + 39 + 13 = 62 CPU ms per business transaction, for running Trader using a CTG applet architecture on an R55 processor. Hence we can calculate the cost of running Trader at our designated throughput of 10 business transactions per second to be 62 * 10 = 620 CPU ms.

Of this total 620 CPU ms for running Trader using CTG Java applets, we can estimate how much should be allocated to the different OS/390 components. We do this by first deducting the known cost of 130 ms for the business logic in TRADERBL, and then using the relative proportions reported for each component in our test results. We used our results from the 1000 byte TCP/IP test found in Table 109 on page 199 . A throughput of 30.37 Web requests per second was chosen, as it is the closest to our defined rate of 10 business transactions per second (or 30 CICS tasks per second). This is illustrated in Table 17 .

Table 17: CPU percentage breakdown for CTG applet Trader

Component	Percentage of total per component	CPU usage for 10 business transactions (CPU ms)
CICS TRADERBL	-	130 ms
CICS other	9.7%	48 ms
TCP/IP & VTAM	3.9%	19 ms
CTG Java gateway application	64.8%	317 ms
OS/390 other	21.6%	106 ms
Total		620 ms

7.4 Performance tests using CTG Java servlets

In the following section we show the results of our performance tests of Java servlets and the OS/390 CTG. You should be aware that the test scenarios and applications used were simplified in order to quantify the configuration under analysis; the application tested was not a real life application such as Trader.

7.4.1 Test environment

The test environment was equipped with sufficient hardware (processor, memory, DASD, network bandwidth) to eliminate any constraints. The operating system was OS/390 V2.7. We used the CICS Transaction Gateway for OS/390 V3.1, together with CICS TS V1.3, JDK V1.1.8, WebSphere Application Server V1.1, and the OS/390 IBM HTTP Server v5.1. The test environment is illustrated in Figure 52 on page 121 , and full details of the software levels and parameters in effect are listed in Appendix A "Test environments" on page 161 .

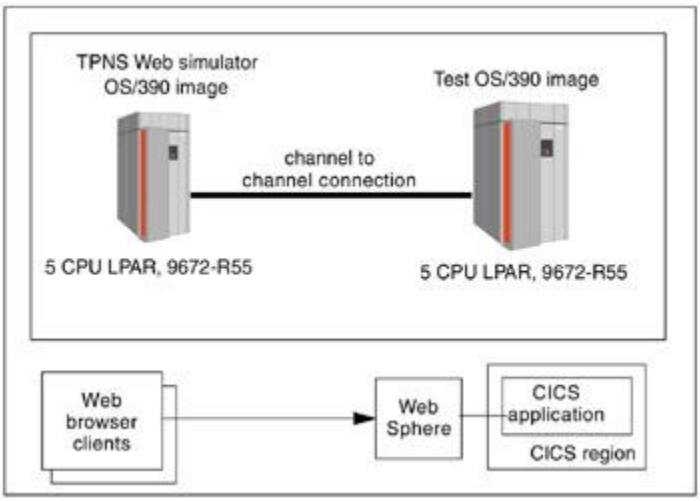


Figure 52: CTG servlet test environment.

7.4.2 Test methodology

The Web browser workloads were emulated using TPNS, this was achieved by capturing the network flows of a sample CTG Web browser client and then replaying them at different throughputs. The TPNS driver was running on a separate 9672-R55 processor within the sysplex so as not to interfere with the OS/390 test image.

A range of five throughputs from approximately 30 to 100 Web requests per second were achieved by varying the think time of the simulated Web browsers within TPNS. The number of Web users was set 100 for the servlet tests. The workload allowed to settle before a five minute measurement interval was sampled using OS/390 RMF.

The application running in the CICS region was a minimal application, that is, the application received a short COMMAREA (of 39 bytes), changed the last byte, and returned it. The reason for choosing such a minimal application and small COMMAREA size, was that we wanted to show the amount of CPU usage for invoking a CICS application from the Java servlet environment.

We ran tests to determine the costs of the following quantifiable components when using CTG Java servlets:

- Creation of the HTTP connection
- Basic servlet cost
- Cost of an ECI call from within the servlet

Our servlet tests used a very simple Java servlet that sent back a minimal HTML reply to the HTTP GET method used to invoke the servlet. We did not use Java server Pages (JSP), Visual Age Java(VAJ), or the Common Connector Framework(CCF) in the development of our servlet, and if you do so you should quantify any such additional costs incurred.

7.4.3 Test results

In this section we present a graphical summary of the performance measurements, in order to highlight the important points and to provide the necessary information to perform a capacity planning estimation of an application such as Trader.

OS/390 servlet JVM performance

You should be aware that new versions of the Web-enablement connectors (OS/390 Java Development Kit, WebSphere Application Server, and CTG) are constantly being developed by IBM, each release of which has historically shown significantly improved performance. The numbers presented here for OS/390 CTG Java servlets are merely a snapshot in time, with expectation for continued improvements in future releases. Refer to http://www.s390.ibm.com/java for the latest details.

Servlet using the ECI

First we analyzed the cost of CTG ECI calls to a simple CICS application from a servlet. This is illustrated in Figure 53 on page 123 , detailed data for these measurements are shown in Table 121 on page 204 . The figures plotted are the percentage usage of a single R55 CPU, with a maximum of 500% available. The Web clients used persistent HTTP connections to communicate with the OS/390 Web server. This graph shows good scalability at the workloads measured, and you can see that the majority of the CPU used is incurred in the Web server address space, since this is the process that serves the HTML pages and runs the JVM and the CTG Java methods.

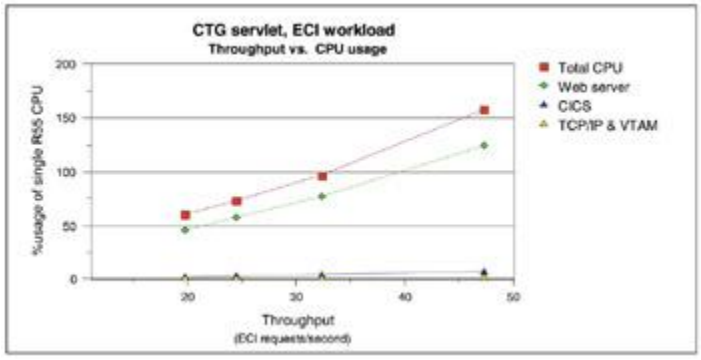


Figure 53: CPU usage of CTG servlets

Servlet with no ECI call

Next we analyzed the cost of the same servlet but without the ECI call to CICS, in order to determine the delta cost within the servlet of invoking the EXCI to pass the COMMAREA to CICS. The resulting total CPU usage along with the CPU usage when invoking CICS is illustrated in Figure 54 ; the raw data for these measurements can be found in Table 121 on page 204 and Table 123 on page 205 . The figures plotted are the percentage usage of a single R55 CPU, with a maximum of 500% available. The Web clients used persistent HTTP connections to communicate with the OS/390 Web server.

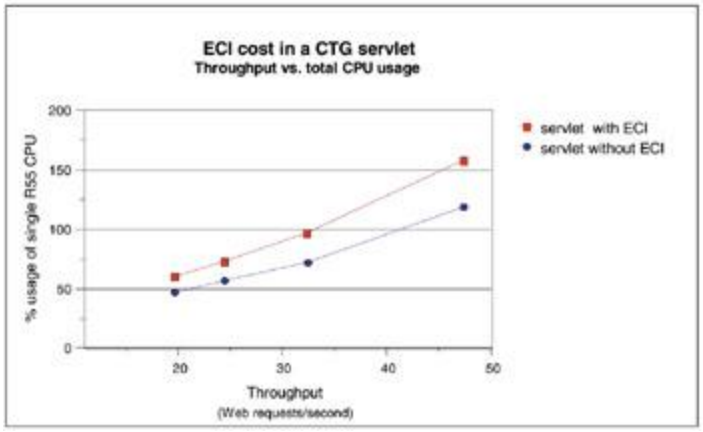


Figure 54: CPU usage of servlets with and without the CTG

The graph shows that there is a significant cost associated with calling CICS from a servlet. This is 24% of the cost of our test servlet or, on average, 8 CPU ms per Web request. This cost is unlikely to increase significantly as the COMMAREA size increases, since the CTG uses the EXCI protocol to pass data to the CICS region. The EXCI utilizes the CICS MRO protocol to pass data to CICS, either via cross memory communication if the CTG and CICS region are within the same CEC, or via an S/390 coupling facility if the CICS region is in a different CEC in the Parallel Sysplex. Both of these communication mechanisms should have minimal costs.

Persistent HTTP connections

Next we analyzed the cost of the non-ECI servlet but measured the increase when persistent HTTP connections were not used, in order to determine the saving of using persistent HTTP connections over non-persistent HTTP connections. The resulting total CPU usage is illustrated in Figure 55 ; the figures plotted are the percentage usage of a single R55 CPU, with a maximum of 500% available. The raw data for these measurements can be found in Table 123 on page 205 and Table 124 on page 205 .

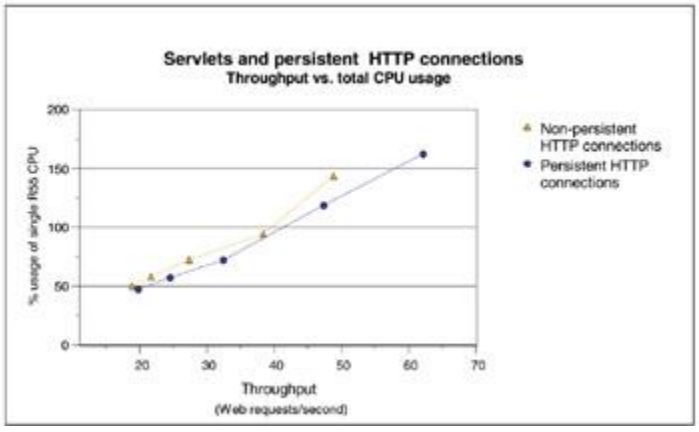


Figure 55: CPU usage of servlets with persistent HTTP connections

The graph shows that there is a saving associated with use of persistent HTTP connections of, on average, 10% of the cost of invoking the servlet, or 2.7 CPU ms per call. We will use this figure later in our capacity planning estimation. The data also suggests that at higher throughputs the usage of persistent HTTP connections provides better scalability, since there is a marked increase in CPU usage

for the last data point with non-persistent HTTP connections.

7.5 Capacity planning for CTG Java servlets

In this section we use the results of our previous performance tests to create a capacity planning methodology for estimating the CPU usage of a Web-enabled CICS application using servlets with the OS/390 CTG. We then use this methodology to estimate the CPU usage when the Trader application is Web-enabled using CTG servlets.

7.5.1 Capacity planning methodology

As illustrated in Figure 44 on page 107 , our Web-enabled servlet design for Trader has one initial call to the servlet to build the signon page, and then three calls to the servlet which invoke the business logic in the TRADERBL application — thus giving a throughput of 40 Web requests per second, and 30 CICS task per second, for our defined 10 business transactions per second.

The CPU costs of invoking the business logic in TRADERBL are already documented in Table 2 on page 45 . This cost will be 13 CPU ms per business transaction, since only 3 calls are made to the CICS business logic. The presentation logic will be implemented in the Java servlet, but the cost of this is not included as part of our capacity planning estimation, as the costs are indeterminate. These costs should be quantified and factored into any servlet capacity planning estimation.

Thus the total OS/390 CPU costs per second for running the Web-enabled Trader application at a throughput of 10 business transaction per second are:

1. Cost of initial request to invoke a servlet with no ECI call using a non-persistent HTTP connection.

2. Cost of three requests which invoke the ECI using persistent HTTP connections.

3. Cost in CICS of the requests to the business logic in TRADERBL.

We can calculate the cost of the first call using the average of the *CPU ms/request* from our data in Table 124 on page 205 *(Servlets, non-persistent HTTP connection, no ECI)* . This is 27 CPU ms.

The cost of the next three requests which invoke the ECI can be calculated using the average of the *CPU ms/request* from our data in Table 121 on page 204 *(Servlets, persistent HTTP connection, ECI)* . This is 32 CPU ms per request.

The CPU costs of invoking the business logic in TRADERBL are already documented in Table 2 on page 45 . This cost will be 13 CPU ms per business transaction, since when using our CTG servlet architecture only three calls are made to the CICS business logic.

7.5.2 Capacity planning estimate

Using our capacity planning methodology we can estimate the OS/390 CPU usage for one Trader business transaction. We will use a throughput of 40 Web requests per second, since this is the throughput we wish to achieve.

1. Cost of initial request to invoke a servlet with no ECI call using a non-persistent HTTP connection:

   27 CPU ms

2. Cost of three requests which invoke the ECI using persistent HTTP connections:

   32 * 3 = 96 CPU ms

3. Cost in CICS of the requests to the business logic in TRADERBL:

   13 CPU ms

Thus the total is 27 + 96 + 13 = 136 CPU ms, for one Trader business transaction, and hence we can calculate the cost of running Trader at our designated throughput of 10 business transactions per second to be 136 * 10 = 1360 CPU ms.

Of this total 1360 ms, we can estimate how much should be allocated to the different OS/390 components. We do this by first deducting the known cost of 130 ms for the business logic in TRADERBL, and then using the relative proportions reported for each component in our test results. We used our results for *servlets, persistent HTTP connection, ECI* found in Table 121 on page 204 . A throughput of 47.37 Web requests per second was chosen, as it is the closest to our defined rate of 10 business transactions per second (or 40 Web requests per second). This calculation is illustrated in Table 18 .

Table 18: CPU percentage breakdown for CTG servlet Trader

Component	Percentage of total per component	CPU usage for 10 business transactions (CPU ms)
CICS TRADERBL		130
CICS other	4.6%	57
Web server	78.5%	966
TCP/IP & VTAM	1.5%	18
OS/390 other	15.4%	189
Total		1360

7.6 Trader performance comparison

Using the figures in Table 17 on page 120 and Table 18 on page 128 , we have compared the cost of Web-enabling the Trader application using a CTG Java applet architecture and a CTG Java servlet architecture. This is illustrated in Figure 56 .

The figures plotted are CPU ms on an 9672-R55, for running 10 invocations of the Trader business transaction. Thus 10 Trader business transactions equate to 30 Web requests or CICS tasks when using applets, 40 Web requests but only 30 CICS tasks when using servlets, and 100 CICS tasks when using 3270 green screens.

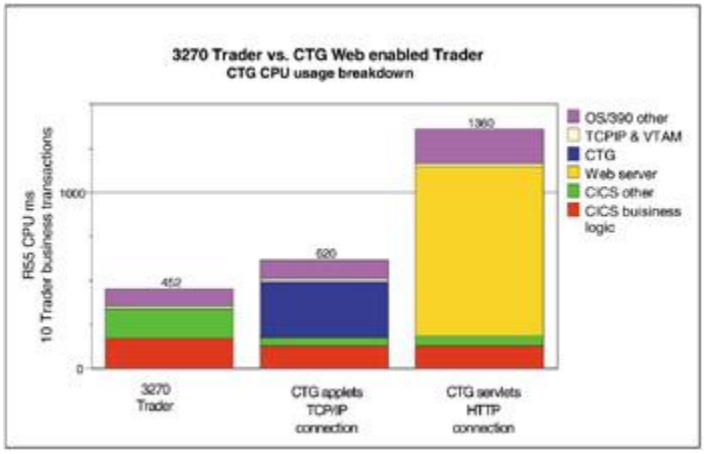


Figure 56: CPU usage comparison for Trader via CTG

It should be borne in mind when comparing these figures that the Java servlet architecture is fundamentally different from the applet architecture because the presentation logic is implemented within the Java servlet which runs on OS/390.

The CTG applets figures assume usage of the CTG TCP/IP protocol and re-use of the TCP/IP connection as discussed in "Cost of making the applet TCP/IP connection" on page 116 . We also do not factor in any savings that workload balancing would give when using multiple CTG Java gateway application address spaces, as found in "Multiple CTG address spaces and the TCP/IP protocol" on page 118 .

Chapter 8: Conclusions and recommendations

Overview

The objective of this redbook is to help you understand the performance impact of Web-enabling your CICS-based applications, and to provide the necessary information to perform capacity planning estimation. The redbook *Revealed! Architecting Web Access to CICS* , SG24-5466 explains the choices available to you and helps you decide which is the best solution to choose. There are many factors influencing this choice, but having considered which technical solution to adopt, it is important to ensure that this solution delivers both the function and the performance you require.

In the previous chapters we have presented data from our test studies that demonstrate the CPU cost of the different Web-enabling methods, and we have illustrated how to apply this data to a typical legacy CICS COBOL application. In addition to the estimation processes, each chapter includes a general discussion of the important factors affecting the performance of each solution and provides some guidelines that will help you if you implement that particular solution.

In this chapter we will summarize the conclusions from our performance study and provide some

recommendations to improve the performance of your Web-enabled CICS application.

In Chapter 9 , "CICS Web capacity planning example" on page 153 we will go on to use our capacity planning methodologies to tell a fictional story of how the "Trader company" Web-enabled its legacy CICS application.

8.1 Interpreting the performance data

Although the studies presented in this book have been designed to give generally applicable results, they may not be a good representation of your application. Any capacity planning estimate you use, whatever the source, should always be verified on a test system before the application is put into production. If your test system does not perform as well as you expected, check whether you have followed the recommendations available. Try to understand which components are not working as well as you anticipated. You may be able to use the data presented to determine that one particular component is using excessive system resources. Don't just put your application into production, expecting it to fix itself!

8.2 Analysis of results

A comparison of our capacity planning estimates for the CPU costs of Web-enabling our Trader workload are shown in Figure 57 . The figures plotted are the total CPU ms used on an 9672-R55, for running 10 Trader business transactions.

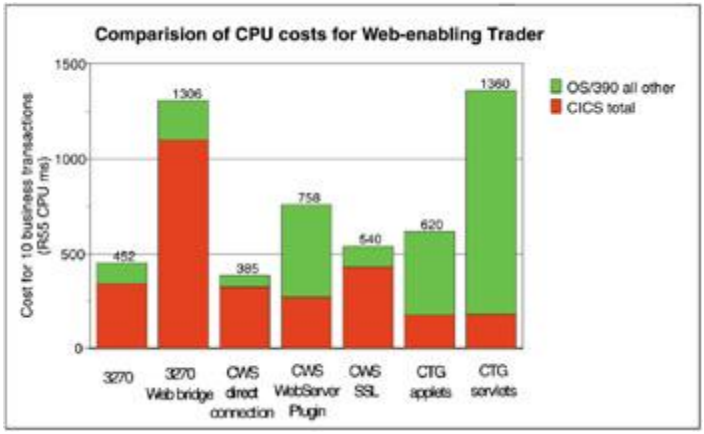


Figure 57: Capacity planning estimates to Web-enable the Trader application

When comparing the results of our capacity planning estimates the following points should be considered.

- In our test to confirm our capacity planning estimate for a *CWS directs connection* we found that our estimate was too low; refer to 5.3.3 , "Confirming our estimate" on page 81 for further details.
- In our *CWS SSL* capacity planning estimate we use minimal costs for SSL. There are many different combination possible when using SSL; refer to 6.3.1 , "Capacity planning methodology" on page 96 for more details.
- Our *CTG servlets* capacity planning estimate is based on a simple servlet with only minimal

presentation logic. Additional presentation logic such as the use of JSPs will need to be factored into your capacity planning estimates; refer to 7.4.2 , "Test methodology" on page 122 for more details.

- These figures are a snapshot taken at the time of this study. IBM is constantly striving to improve the performance of its Web-enablement and Java technology; refer to http://www.s390.ibm.com/java for more details.

All of the methods of CICS Web-enablement detailed in this book demonstrate the functionality and scalabilty of these OS/390-based solutions. This is shown by the general linear nature of the results shown. These studies have been performed with simple applications that have been designed to be generally applicable. As we have shown with our Trader application analysis, these results can be applied to give approximate costs for Web-enablement, but these estimations should used in conjunction with measurements of your own applications in a test environment.

Remember that the business logic, the processing in the CICS application of the business requests to update the business data, are largely unaltered by the changes in presentation logic. The same kinds of requests, to do the same kinds of work, are still going to be received by the business logic; it is the way that the results are viewed by the user that has changed. The main elements of capacity planning for such a change involve understanding how much extra it may cost, in which components of your systems is the cost to be applied and, if your current system cannot support such an increase, what upgrades you should consider.

It should be remembered that increased functionality will cost more to support; the old adage "there is no such thing as a free lunch" remains true, even when Web-enabling. For example, using the 3270 Web bridge is considerably more costly than using the CWS with new Web-aware presentation logic, but using the 3270 Web bridge does not require the application changes that the new Web-aware application would.

In the following sections, we discuss each of the CICS Web-enabling technologies in the light of our results.

8.2.1 CWS

The observations and recommendations given in this section apply to CWS support in general, either via the 3270 Web bridge or via a Web-aware application design. You should also note that a significant number of improvements to CWS were introduced in CICS TS V1.3. The main performance items of interest are the introduction of persistent HTTP connection support, the ability to store HTML templates in memory, and the removal of the 32KB restriction for a single CWS request. Functional improvements include the introduction of the CICS WEB and DOCUMENT APIs and the implementation of SSL support for a CWS direct connection.

Direct connection

Using a direct connection with the CWS uses less total CPU than using the WebServer Plugin; this is to be expected, since the instruction path-length is much shorter and there in less inter-process communication involved. However, it does use more CPU within the CICS address space than using the WebServer Plugin, which could be a disadvantage if your CICS address space is CPU constrained.

CICS WebServer Plugin

Using the WebServer Plugin does cost a bit more than using a direct connection. This extra cost is incurred within the Web server address space and includes the CPU usage of the CICS supplied CWS WebServer Plugin, which replaces the function of the CICS sockets listener. The extra cost should be borne against the extra functionality provided by the Web server. Using the Web server may be the preferred option if you wish to combine calls to different OS/390 servers, or want to isolate your CICS system from direct Web access, or off-load some of the CWS processing to the Web server.

The CWS WebServer Plugin uses an External CICS Interface (EXCI) connection to communicate with the CICS region. You can verify that this connection has sufficient sessions or pipes allocated by using CICS statistics reporting.

Network design

Good network design and capacity is vitally important to a successful Web-enablement. The response times in CICS are going to be a small contributor to the overall response time. Most of the user-perceived response time from a Web browser will depend on network response.

HTTP datastream

In our tests we found the costs of transmitting data over HTTP connections increased linearly with size, and at small sizes (a few KB), were only a small fraction of the costs involved. You should be aware that the HTTP headers are added to the data to be transmitted, and this can mean that an additional 200–300 bytes is added to the data actually transmitted. CICS monitoring information records the size of HTTP datastreams at a transaction level (for more information, look at the DFHWEBB performance data group described in the *CICS Performance Guide* , SC33-1699).

Persistent HTTP connections

Using persistent HTTP connections will reduce the overall cost of transmitting data over HTTP connections. However, be aware that when using a direct connection there will be a long-running Web attach transaction (CWXN) for every active connection. These will be terminated by the Web browser or when the SOCKETCLOSE time-out interval in the TCPIPSERVICE definition is reached. When using persistent HTTP connections, you should consider whether the number of long-running tasks can be supported compared to the amount of CPU time you save by using persistent connections. You can use the SIT parameter MXT and add the Web attach transaction (CWXN) to a TRANCLASS with a MAXACTIVE setting to stop Web-based requests from flooding your CICS region.

Restricting TCP/IP requests into CICS

As well as limiting Web requests into CICS by limiting the number of active tasks, you should consider the number of requests you are prepared to have buffered by TCP/IP. Specifying the BACKLOG parameter in the TCPIPSERVICE definition will limit the number of requests held by TCP/IP. Make sure that the TCPIP SOMAXCONN parameter (the maximum number of pending connection requests queued for any listening socket — the default is 10) is greater than or equal to the BACKLOG setting.

TCP/IP buffer sizes

Set TCP/IP buffer sizes large enough to contain the largest data transfer expected. In our tests, TCPSENDBFRSIZE and TCPRCVBUFRSIZE were set to 65536. These parameters are documented in *OS/390 V2R7.0 eNetwork CS IP Configuration* , SC31-8513, along with advice not to over-allocate

buffer space. However, in our tests we found no significant difference in CPU usage compared to using smaller buffers with smaller data exchanges, but we benefited from better network responses.

HTML template support

HTML template support has been greatly improved in CICS TS V1.3. You are no longer restricted to storing these templates in the DFHHTML PDS; now you can use DOCTEMPLATE definitions to locate them in many kinds of CICS-managed storage. The best performance is achieved by defining them as programs; details on how to do this are given in the redbook *CICS Transaction Server for OS/390 Version 1 Release3: Web Support and 3270 Bridge* , SG24-5480.

CWS temporary storage queue placement

The TSQPREFIX referred to in the CICS TCPIPSERVICE definition gives you the opportunity to choose the location of the CICS temporary storage queue (TSQ) that will be used to hold the data exchanged with the Web client. The best performance will be achieved if this TSQ is defined as MAIN storage. However, if large amounts of data are to be exchanged, it might be appropriate to make this storage AUXILLIARY. Note that using AUXILLARY storage queues will result in the data being stored on DASD. This will increase disk I/O and may therefore increase end user response times.

8.2.2 CWS and 3270 Web bridge

This section deals with the CWS factors affecting performance of the 3270 Web bridge. You should also refer to the general advice on CWS given in 8.2.1 , "CWS" on page 133 .

Pseudo-conversation length

The key factor when using the 3270 Web bridge is the length of the associated 3270 pseudo-conversational chain. This is because the 3270 bridge facilities and state data are created at the beginning of each pseudo-conversation and then destroyed afterwards. The longer the pseudo-conversation lasts, then the less management of bridge facilities and the less state data is needed. However, there is no benefit to be gained in deliberately lengthening a pseudo-conversational chain; you should only consider merging separate pseudo-conversations into one.

Size of HTTP datastream

The amount of data transmitted per 3270 screen image is not such an important factor when estimating CPU usage with the 3270 Web bridge. This is because the average amount of data representing a 3270 screen image shows little variation, and in any case, is relatively small (about 2KB). In our investigations in Chapter 5 , "CWS with Web-aware presentation logic" on page 65 , we showed that sending such relatively small amounts of data is only a small proportion of the overall CPU costs of using CWS. More important factors are the number of CICS tasks and whether or not HTTP persistent connections are used.

8.2.3 CWS with Web-aware presentation logic

This section deals with the factors affecting performance of CWS when using new Web-aware presentation logic. You should also refer to the general advice on CWS given in 8.2.1 , "CWS" on page 133 . Note that our capacity planning estimate for Trader using the CWS with Web-aware presentation logic was somewhat less than that measured for the actual Trader application; refer to 5.3.3 ,

"Confirming our estimate" on page 81 . We believe this to be because the Trader Web-aware logic contains additional logic such as building templates and storing state data. Such costs should be factored into any of your capacity planning estimates by careful measurement.

Size of HTTP datastream

The size of the HTTP data stream does affect the CPU usage of the CWS and is covered in more detail in Chapter 5 , "CWS with Web-aware presentation logic" on page 65 . However, as can be seen from the equations in Figure 34 on page 77 , the cost is a relatively small component if only a few KB of data are transmitted as in our Trader application. In this case it is more important how many CICS tasks run, since the dominant cost is the null or fixed cost. Since much larger amounts of data can now be transmitted in CICS TS V1.3, then the cost of data transfers can become significant. It was also discovered in our tests that the sending of data using CWS is significantly less expensive than the receiving of data.

Programming considerations

We recommend using the new DOCUMENT and WEB APIs provided in CICS TS V1.3 when creating new Web-aware presentation logic. This makes HTTP presentation programming much easier than before when using the COMMAREA manipulation technique. In our test results there was little difference in costs between using the WEB API and using COMMAREA manipulation (see Figure 31 on page 74).

The CICS Web Interface (CWI) in previous releases of CICS recommended running your Web-aware presentation logic in the converter phase (this was intended to ease access to HTTP data areas which are now readily accessible using the WEB API). This is no longer necessary, and by running your Web-aware presentation logic as a normal CICS program, this gives a small additional benefit of saving a CICS LINK call.

8.2.4 SSL with CWS

CTS V1.3 uses the system SSL toolkit, part of OS/390 V2.7. Make sure you have the current System SSL and Web server or CICS TS V1.3 service applied if you wish to use this function.

There are two processes that SSL supports: handshaking to establish a secure connection, and data transmission over this secure connection.

8.2.4.1 Handshaking

SSL handshaking is likely to the be most CPU intensive part of using SSL. In our capacity planning estimate SSL handshaking accounted for 82% of the SSL costs in our business transaction. Therefore in order to reduce the costs of SSL you should design your application to have the lowest handshaking costs possible, with regard to any security considerations you may have.

A full handshake is the most CPU-intensive phase of SSL and is performed at the start of each SSL session. An SSL session may be re-established when a client makes a new HTTP connection. This is achieved by passing the previous SSL session ID to the server. An SSL session ID remains valid for a period determined by the server; in the case of CTS V1.3 this time-out period is defined by the SIT parameter SSLDELAY. If an SSL session is re-established by this method then a shorter or "null" SSL handshake is performed, which is considerably less CPU intensive. The value of SSLDELAY in the SIT

should be set as high as possible, with regard to any security concerns you may have about the time an SSL session ID may remain unused but secured.

The use of the S/390 Cryptographic Coprocessor Feature was very successful in reducing the CPU costs associated with the full handshake, particularly when client certificates or the larger 1024 bit server key was used. If you are not able to use the Cryptographic Coprocessor Feature, the use of the smaller 512 bit server key will reduce the cost of the full SSL handshake. The use of persistent HTTP connections ensures that after a full or null SSL handshake, no other SSL handshaking is performed until the persistent HTTP connection is broken. In CICS, a persistent HTTP connection will be broken either when CICS times-out the connection according to the SOCKETCLOSE value in the TCPIPSERVICE definition, or when the Web browser terminates the connection.

Note that although we did not test the usage of the Cryptographic Coprocessor Feature with SSL handshaking and the CICS WebServer Plugin, the OS/390 Web server can utilize Cryptographic Coprocessor Feature in the same way as we demonstrated for the CWS direct connection.

TCBs

SSL support in CICS TS V1.3 uses a pool of TCBs dedicated to SSL work, the S8 TCBs. The number of S8 TCBs is specified using the CICS SIT parameter SSLTCBS. Each new TCB occupies an amount of storage below the 16MB line. Thus if your CICS DSA usage is critical (for instance, you have lots of old 24 bit programs) you may be restricted to the number of S8 TCBs your system can support. CICS monitoring and statistics data can be used to measure the amount of CPU time these TCBs use.

TCB stealing

All forms of the SSL handshake are expensive but good scaling was evident for all variations of the handshake in CTS 1.3. The amount of full handshakes should be minimized to reduce CPU usage by using persistent HTTP connections and sufficient S8 TCBs.

You should be aware that once the number of attached SSL clients exceeds the number of defined SSL TCBs in a CICS region, then subsequent HTTPS requests will 'steal' the least previously used SSL TCB. Since there is a one to one affinity between a HTTPS session and an individual SSL TCB, then TCB stealing will cause Web browsers that send a subsequent HTTPS request to CICS to incur the additional cost of an SSL null handshake and the creation of a new HTTP connection

8.2.4.2 Data transmission

Once the SSL handshake has been performed and the Web client and target CICS region maintain a persistent HTTP connection, data transmission is the only additional cost to SSL operation. In our capacity planning estimate the SSL data transmission accounted for only 18% of the SSL costs but you may experience a higher percentage than this, if you transfer larger datastreams or if you have lower SSL handshake costs.

The Cryptographic Coprocessor Feature supports data encryption using the DES or Triple DES ciphers, and can be used with either the CWS direct connection or the CICS WebServer Plugin. In our tests we quantified savings when transferring data using the triple DES cipher. We also found that using the RC4-MD5 bit cipher with either the 40 bit or 128 bit key cost the same in terms of OS/390 CPU usage, and also cost less than the use of the triple DES cipher using cryptographic hardware. The reason that the RC4-MD5 40 and 128 bit ciphers cost the same is because they both pass a 16 byte key length into

the encryption algorithm. The difference is that 40 bit encryption uses 'salted' (unencrypted random data) as part of key-block used to generate the 16 byte key. This reduces the strength of the encryption, but the path length remains the same.

8.2.5 CICS Transaction Gateway

In this section we shall examine the principal factors affecting CPU usage when using the CTG.

Java support

Java support in OS/390 is being continually improved. You will receive significant performance benefits from being at the most recent levels of OS/390 (with the associated Java Development Kit, TCP/IP and WebSphere Application Server versions).

8.2.5.1 Java applets

This section discusses the important factors when using a CTG applet architecture.

Network protocol

The network protocol used to connect your applet to the CTG Java gateway application will have a significant effect on system performance. This is illustrated in our test results (Figure 49 on page 116) where we found the CTG TCP/IP socket protocol performed better than the CTG HTTP protocol. You may, however, choose to use HTTP for its ease of routing, since HTTP flows are much easier to route through an HTTP application proxy server in a firewall.

Both the CTG HTTP and TCP/IP protocols allow for connection reuse, whereby the connection from the applet to the Java gateway application is kept open for the duration of several External Call Interface (ECI) calls. Figure 49 on page 116 demonstrates the considerable saving this has for data transfers using the TCP/IP protocol. The HTTP protocol will also experience a saving with connection re-use, but you should be aware that the CTG HTTP protocol handler does not support persistent HTTP connections. This means that even if the CTG HTTP connection is re-used across ECI calls there will still be a new underlying TCP/IP socket open and close for every HTTP request.

Applet data transmission

The size of the ECI COMMAREA has a significant effect on CPU usage in the Java gateway application as shown in Figure 50 on page 117 , and thus reduction of data transmitted is an important performance factor. There are techniques to reduce the amount of data passed, which fall into three broad categories:

1. First, you should design the application so it has the minimum number of data flows from the Web client through to the CICS server. Your options may be limited by the existing interface offered by your CICS application, and your ability to re-engineer these interfaces.

2. Second, you should design the application to transmit only the data essential to the Java application, that is, only the data that it directly needs for its presentation or business logic.

3. Third, you can compress or truncate the data flowed across the network.

Data truncation facilities are built into the CICS Transaction Gateway and CICS client-server flows and

can be easily invoked as follows:

- The CICS Transaction Gateway provides two methods for limiting the amount of data transferred when using ECI calls from an applet. The setCommareaOutboundLength method controls how much of the CICS COMMAREA will be flowed from the applet to the Java gateway application; and the setCommareaInboundLength controls how much of the COMMAREA returned by CICS is flowed from the Java gateway application to the applet. Note that these calls do not affect the actual length of the COMMAREA returned to the application, just the amount of the COMMAREA sent across the network. You should always design these calls into your applet code if you wish to minimize the data sent from the applet to the Java gateway application. Without these calls, the whole string representing the COMMAREA will be transmitted, including any trailing null characters. Note that these methods were not used in our testing.

- The CICS External Call Interface (EXCI), in combination with the CICS Inter System Communication (ISC) code, provides truncation for EXCI flows. Any trailing nulls are not physically passed from the client process to the CICS region. This truncation is automatic and not configurable. It is appropriate when transmitting data from CEC to CEC in a Parallel Sysplex, since this involves network communication by means of the sysplex coupling facility. Thus you should design your CICS COMMAREAs to be padded with trailing nulls and to store data efficiently in the beginning of the COMMAREA.

Data compression is applicable when the system is network I/O bound and yet still has CPU cycles spare. This was not investigated in our performance studies, but other internal IBM studies have shown that savings are only likely to occur if many clients are trying to transfer large amounts of data over a low bandwidth network. In most normal circumstances, using data compression will only add to the CPU usage in the OS/390 system.

- The CICS Transaction Gateway security exits can be used to compress data instead of, or as well as, encrypting data. The data is compressed as it leaves the applet and uncompressed as it enters the Java gateway application. Examples of how to use these exits for data compression are given in the ClientCompression.java and ServerCompression.java samples in the samples\Java\com\ibm\CICS Transaction Gateway\security directory, and a working example is given in *CICS Transaction Gateway with More CICS Clients Unmasked* , SG24-5277.

Figure 58 illustrates the possible points for data compression and truncation, when using an ECI based Java applet via the CICS Transaction Gateway to a CICS server.

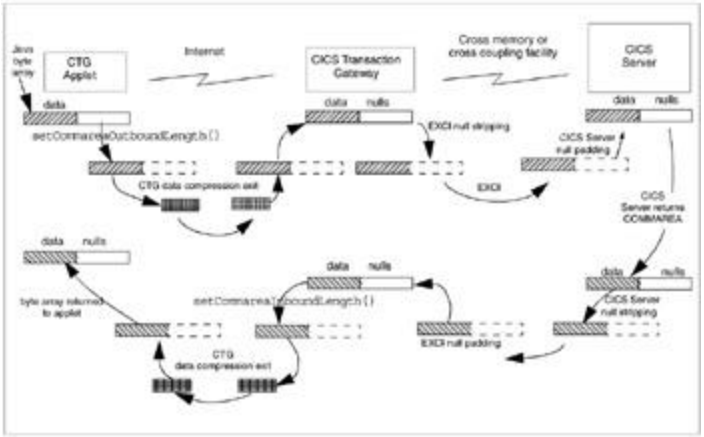


Figure 58: Data compression using CTG applets

If using the servlet architecture, it is only possible to reduce the data transmission at the point where the data flows from the CICS Transaction Gateway to the CICS Server, since the CICS Transaction Gateway methods are executed within the Web Server.

Whatever architecture you use, it is best to compress the data as early as possible in the life cycle of the data to reduce the flows through the different components. If encrypting data, you should ensure that this is performed after any compression routines for reasons of efficiency.

The items above are described in more detail in the "Performance and Scaling" chapter of *Revealed! Architecting Web Access to CICS* , SG24-5466.

Thread usage

The CTG Java gateway application is a multi-threaded Java application. These threads are held in two pools, connectionManager threads and worker threads. A connection thread is needed for every connected Web client, and a worker thread is needed to process the ECI request to CICS. The number of threads the Java gateway application uses is defined by the Maxconnec t and Maxworker parameters contained in the CTG.INI file. This is described further in "CTG thread usage" on page 108 .

It was found in our tests that the Java gateway application could not use more than 150 CPU% out of the 500% available on our R55 CEC. Increasing thread counts had no further effect on the systems utilization or throughput. We recommend that you consider using TCP/IP port sharing to distribute work across multiple Java gateway applications if you need to increase throughput in such circumstances. As is shown in Figure 51 on page 118 , using TCP/IP port sharing and multiple Java gateway application address spaces can give a highly scalable architecture when using the CTG. The point at which the CTG Java gateway application may become thread constrained will depend on several factors and can only be determined by experimentation. However, one of the principal factors is the longevity of the call to the CICS application, since this will have a bearing on the worker thread usage within the Java gateway application.

The number of concurrent EXCI calls that the Java gateway application can make to a CICS region is determined by the number of pipes (sessions in the CICS definition) defined on the EXCI connection to be used. A maximum of 100 can be defined, but it is unusual to find in practice that anywhere near this many are in concurrent use. You can use CICS statistics to determine use count of these sessions or pipes. If you have long-running programs in the CICS region, you are more likely to need more pipes.

8.2.5.2 Java Servlets

In this section we will discuss the important performance factors when using a CTG servlet architecture.

Java logic

The servlet is essentially Java code executing within the OS/390 JVM that builds and sends HTML to the Web Client. There are certain Java functions that are expensive to execute, string handling for example, so bear in mind that you must pay the cost of executing the Java presentation logic in the servlet. This is the primary reason why the CPU usage of the servlet test scenario was considerably higher than that of the Java applet test scenario.

Note that when developing servlet presentation logic you should consider that a complex servlet HTML GUI will require the re-transmission of the complete HTML page for every Web request. You may therefore want to reduce the size of servlet data transmissions by developing a less complex GUI.

An advantage of the servlet architecture over the applet architecture is the ability to condense multiple ECI calls to a CICS region into one Web user response using new business logic implemented in the servlet. In our tests we found that the ECI call within servlet was only 24% of the total servlet cost, thus using new business logic to condense multiple servlet calls into one Web request may well give large overall savings. We did not exploit this feature in our capacity planning model with Trader, since every user Web request drove one ECI call.

Servlet data transmission

With the servlet architecture there are two network connections:

- The connection from the Web browser to the Web server via HTTP
- The connection from the Web server to CICS via the EXCI

The IBM HTTP server provides for persistent HTTP connection support, and in our tests we found a saving of about 10% when enabling persistent HTTP connections. This is illustrated in Figure 55 on page 125 .

The connection between the servlet and target CICS region is via the EXCI. This uses the CICS MRO function. MRO functions are a very efficient cross-system communication mechanism, that can use cross-memory communication between partners in the same MVS image, or (cross coupling facility) XCF functions between partners on different members of a sysplex. Thus the points outlined in , " Applet data transmission " on page 140 should be considered. However, you should note that the CTG method setCommareaInboundLength has no effect within the servlet environment, since the communication from the CTG to the Java application is all within the Web server address space. Also note that the setCommareaOutboundLength method can be used to reduce the length of the COMMAREA sent from the servlet to the CICS region, which may be appropriate with large amounts of data transferred.

Thread usage

The Web server servlet engine is a sophisticated multi-threaded environment within which the CTG Java methods are invoked. You should make sure that the Web server has sufficient threads to support your workload. WebSphere Application Server provides a graphical monitoring function that enables you to determine thread usage.

However, the same considerations apply as they did to the CTG Java gateway application threading model discussed in the applet section " Thread usage " on page 143. If you experience an inability to increase throughput beyond a certain point, we would recommend using the OS/390 Web server in scalable mode, whereby multiple address spaces are created based on the rate of Web requests.

Additional costs are likely to be incurred if larger data streams are returned from the servlet to the Web browser, since these must be processed by the Web server and TCP/IP. Also, if more complex Java presentation logic is implemented in the servlet, additional costs are also likely to be incurred. This depends on processing of the COMMAREA within the servlet to produce output for inclusion in an

HTML page; this cost is not factored into our capacity planning study.

Apart from these CICS-specific items, you should also be aware of more general servlet performance considerations, such as those discussed in *OS/390 e-business Infrastructure: IBM WebSphere Application Server 1.1 - Customizing and Usage* , SG24-5604.

8.3 Using too much CPU

Multi-tasking in an S/390 environment is achieved by using multiple TCBs, or if using UNIX System Services, by using multiple threads. A multi-TCB or multi-threaded design enables an OS/390 address space to utilize more than one processor concurrently in a multi-processor CEC. The OS/390 CTG and the OS/390 Web server are both multi-threaded UNIX System Services applications.

However, for a CICS region, the majority of the processing occurs in one TCB, the QR (Quasi Re-entrant) TCB. Additional processing occurs in the RO TCB, when opening and closing CICS data sets and making calls to RACF, the FO TCB, when opening and closing user data sets, and optionally (when the SUBTSKS SIT parameter is set to 1), the CO TCB for processing concurrent operations like VSAM requests.

While this is still true for the business logic in a CICS Web application, the design of CICS Web support utilizes two additional TCBs to handle TCP/IP sockets and a configurable number to handle SSL related work. The *CICS Performance Guide* , SC33-1699, describes how to determine if a CICS regionis approaching maximum capacity using CICS statistics reports and RMF records. This method requires an analysis to determine how busy the different TCBs used by the CICS region are. If any single TCB approaches 70% busy, then this CICS region is reaching maximum capacity.

The CPU used by specific CICS TCBs is of particular interest if you are using a direct connection to CICS Web support or using the 3270 Web bridge, since the CPU consumption within CICS is likely to be considerably higher in these cases. However, it can happen in such a region that the overall CPU consumption exceeds that of one single processor without the CICS region actually being processor-constrained. For example, we show an extract from a statistics report in Figure 59 for one of our SSL test measurements.

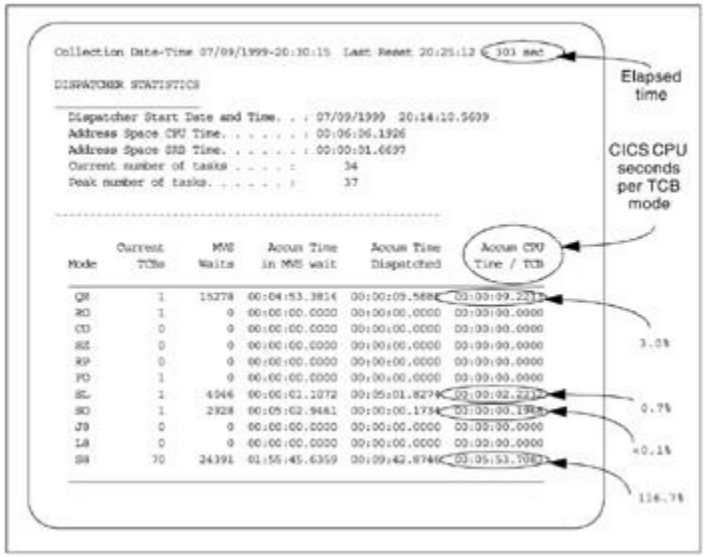


Figure 59: CICS dispatcher statistics extract

This figure gives details for a CICS region supporting Web connections with SSL support, and shows the following CPU% usage by TCB mode:

- 3.0% QR (quasi-reentrant)
- 0.7% SL (sockets listener)
- 0.1% SO (sockets requests)
- 116.7% S8 (SSL - using up to 70 TCBs)

You can see from the *Accum CPU Time/TCB* column that the CICS region used over 365 CPU seconds during a 303 second collection interval (which equates to 120% CPU), even though the traditionally critical QR TCB used only 9 CPU seconds (3% CPU). This data was from a specific SSL test and thus is by no means representative of normal CICS usage.

8.4 Balancing the CICS Web workload

Not only will you have to consider the additional cost that each business transaction may incur by implementing Web-based access, you will also need to pay attention to the response times perceived by the end user. If that user is connecting through the Internet, much of the network transport time will be beyond your control. In addition to the CPU utilization and response time you should also consider the impact Web-enablement will have on transaction rates. Are you going to make the application available to a larger group of users?

If access is by an intranet connection, then the potential group of users will be restricted to those within the intranet boundary; probably the same group of employees that would currently use the application using 3270 terminals. If the Web-enablement has widened the user group, then the additional costs of running the business application more frequently must be planned for.

Access by an extranet connection, two communicating intranets, allows access to the business application by a wider group of users. This is likely to happen where the extranet is allowing two companies to access the same application, such as a supplier company checking the state of another company's stock.

Access by an Internet connection would open up the CICS application to a potentially enormous group of users. The attempted access to the business application from an unrestricted Internet base of users could flood the target CICS system. It is possible to limit the number of requests that one CICS region, CTG Java gateway application, or Web server address space will allow to connect. However, slow response or rejected requests will not be what the user wants.

A better solution is to ensure that your system is scalable, and in an OS/390 Sysplex environment, this means considering forms of workload balancing. Figure 60 shows how different OS/390 components can fit together to provide a scalable server environment.

- TCP/IP port sharing enables multiple CICS regions on the same MVS image to accept incoming HTTP requests sent to a single, shared port number. CTG Java gateway applications and Web servers may also exploit this function.

- TCP/IP Dynamic DNS allows multiple CICS regions in the same sysplex to listen for requests sent to a generic hostname and port number. It can exploit OS/390 Workload Management to balance the workload across these systems within a sysplex.

- Multiple CICS regions controlled by CICSPlex SM can manage requests originating from HTTP requests. The ability to dynamically workload manage Distributed Program Link (DPL) calls within CICS TS V1.3 will greatly benefit such a configuration.

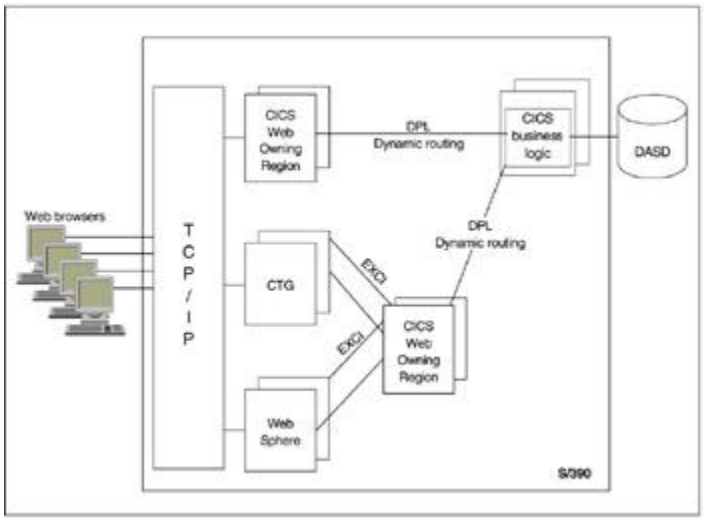


Figure 60: Components to provide workload balancing

The following manual is a good source of reference on OS/390 workload management: *MVS Planning: Workload Management* , GC28-1761.

For information on balancing work in a CICSPlex, refer to *CICSPlex: SM Concepts and Planning* , GC33-0786

For information on dynamic DNS, refer to *OS/390 eNetworks Communications Server: IP Planning and Migration Guide* , SC31-8512.

For information on TCP/IP Port sharing, refer to the *Communications Server: IP Configuration Manual* , SC31-8513.

8.5 Key points to consider

We have demonstrated in this book that for each of the Web-enabling alternatives presented, CICS Transaction Server V1.3 and other supporting software can provide a scalable solution. To summarize the main points of our studies, you need to address the following points as they relate to your application.

Separation of business and presentation logic

Can you separate the presentation and business logic in your existing CICS application and size the business logic costs?

If you cannot separate the presentation and business logic, you will need to consider a 3270 based option such as the CWS 3270 Web bridge, a non-OS/390 CTG using the EPI classes, or Host On-Demand. This redbook only gives information on the performance of the 3270 Web bridge. Note that if you can separate the business logic, this will give you more Web-enabling options and allow usage of solutions which are less CPU intensive and scale better.

CWS 3270 Web bridge

If you intend using the 3270 Web bridge, how long are the pseudo-conversations in the business transactions? Business transactions comprising short pseudo-conversations use more bridge facilities than longer pseudo-conversations, so they are proportionately more expensive.

CWS with Web-aware presentation logic

If you intend using CWS with Web-aware applications, do you know your HTTP send and receive sizes, since these have a significant affect on performance?

CWS with SSL

If you intend to use SSL to secure your CWS solution, your SSL handshake costs are likely to be the most CPU intensive part of the solution. You can reduce SSL handshake costs by:

- Using persistent HTTP connections
- Using the S/390 cryptographic hardware
- Enabling SSL session ID-reuse

OS/390 CTG

If you intend to use applets and the OS/390 CTG, you should use the CTG TCP/IP protocol if possible, and re-use the TCP/IP connection across ECI calls. You should minimize the amount of data transmitted in the ECI COMMAREA wherever possible.

If you intend to use servlets, you will need to use a servlet engine such as WebSphere Application Server. Ensure that you have enough threads defined for this server and that your Java presentation logic is efficient. Consider using new business logic in your servlet to combine the results of multiple ECI calls to reduce the number of network transmissions.

Workload management

Take into account the increased workload likely to be put upon your target OS/390 system. You may need to implement a form of workload management to handle the increased CPU usage or to eliminate a single point of failure.The OS/390 TCP/IP port sharing or Dynamic DNS features enable you to balance work across multiple CICS Web owning regions, CTG Java gateway applications, or instances of the OS/390 Web server.

Make sure that your network is capable of handling the projected extra work; any delays in the network are likely to significantly increase end user response times.

Check your Web-enabled application operation and performance on a test system. Use monitoring and statistics to verify your planning information.

Chapter 9: CICS Web capacity planning example

Overview

In this section we tell the fictional story of the Trader Company, and the capacity planning decisions it made when Web-enabling its CICS Trader application.

The Trader Company is a share trading corporation that runs its key business application on an OS/390 system using CICS. It is considering migrating this system to an e-business infrastructure and, as a first step, would like to enable access to its Trader application from a wider range of users than its traditional brokers who used only 3270 devices. We follow the steps the Company takes on this e-business path. It's a simple story, but the elements are applicable to more complex cases, too. Each step includes an estimate of the increase in systems usage as demand for the application increases, which are reported as:

- **Throughput —** the number of business transactions per second expected
- **CPU usage—** the amount of CPU time in ms estimated to be needed to support the transaction rate expected
- **Target system usage —** the percentage of the total CEC capacity the CPU usage represents, for the machine model specified

9.1 The 3270-based business

The Trader Company runs a CICS application, Trader, to buy and sell shares. In its original form, this application is accessed by users connected to a CICS region using a 3270 terminal.

From our measurements presented in Chapter 4 , "CWS with the 3270 Web bridge" on page 51 we know the cost of a single business transaction using 3270 access. These costs are shown in Table 19 .

Table 19: Single business transaction using 3270 access

CICS TRADERBL	CICS other	VTAM &TCP/IP	CTG	Web server	OS/390 other	Total
17.1	9.2	1.5	-	-	9.4	37.2

- **Throughput —** 10 business transactions per second
- **CPU usage —** 372 CPU ms
- **Target system —** 7% CEC of 9672-R55

9.2 Web access using CWS with the 3270 Web bridge

The Trader Company acquires another Company and needs to offer the Trader application to the clients of their new Company. They have two options, either to extend their 3270 network or to make the Trader application accessible from the Web browsers that every employee has on their workstations.

The Web-enablement strategy for the Trader application is still to be decided, but meanwhile the Company decides to implement a tactical solution, one that will solve the problem now but can be replaced by a longer term strategic solution at a later date. This tactical solution for Trader is to use CICS 3270 Web bridge to give access to all employees of the Trader Company through their Company intranet. No new applications are needed, and the Trader application requires no changes to be Web-enabled in this fashion. No particular skills in HTML or Web servers needed, but approximately double the number of requests are anticipated.

From our measurements presented in Chapter 4 , "CWS with the 3270 Web bridge" on page 51 we know the cost of a single business transaction using the CICW Web bridge. These costs are shown in Table 20 .

Table 20: Single business transaction using CWS with the 3270 Web bridge

CICS TRADERBL	CICS other	VTAM & TCP/IP	CTG	Web server	OS/390 other	Total
17.1	92.8	17.3	-	-	3.4	130.6

- **Throughput —** 20 business transactions per second — a doubling in the throughput.
- **CPU usage —** 2612 CPU ms — note that this exceeds the capacity of one CPU, and that most of that CPU usage is by CICS. There are a number of solutions possible to support this demand on the same processor. For example, create two CICS regions that can handle incoming requests through TCP/IP port sharing. Although no application changes are needed, some systems configuration work will be. Refer to 8.3 , "Using too much CPU" on page 146 for further details on solutions to this situation.
- **Target system —** 52% CEC of 9672-R55.

9.3 Web access using CWS with Web-aware presentation logic

The Web-enablement strategy for the Trader Company is beginning to take shape. There is further growth expected for the Trader application, and the Company is growing Web skills. Web page and HTML design is now understood by the application development group. The 3270 bridge solution works, but it has an expensive bridge layer and still looks like a 3270 screen.

To give the Trader application a better Web look and feel, the 3270 presentation logic in CICS is replaced by a new CICS Web-aware logic. This can not only use HTML but can also be used to create new paths to the business logic. For example, in the case of Trader, drop-down boxes are used to provide a Company selection, and the ten CICS tasks it took to execute the business transaction with a 3270 interface are reduced to five with the Web-aware presentation logic.

From our measurements presented in Chapter 5 , "CWS with Web-aware presentation logic" on page 65 we know the cost of a single business transaction using a direct connection using CWS. These costs are shown in Table 21 .

Table 21: Single business transaction using CWS and Web-aware logic

CICS TRADERBL	CICS other	VTAM & TCP/IP	CTG	Web server	OS/390 other	Total
13.0	19.7	3.4	-	-	-	38.5

- **Throughput —** 20 business transactions per second.
- **CPU usage —** 770 CPU ms, which is a substantial reduction compared to the previous CICS Web bridge solution. At these levels of system usage, multiple CICS regions are not necessary, but may just as well be kept for future growth or to improve application availability (by having more than one CICS region available to service any requests).
- **Target system —** 15% CEC of 9672-R55.

9.4 Web access using CWS and the CICS WebServer Plugin

The Trader Company is facing increasing demand for its Trader application, and it decides to invest further in its Web support. It has implemented the OS/390 Web server on its OS/390 Sysplex and has produced a standard format for its Company Web pages (such as including Company graphics, help and e-mail contacts). CICS provides a very effective way of accessing business logic from Web browser clients, but is not intended to provide full Web server facilities.

The Trader Company decides to update the presentation logic of the Trader application to meet Company standards, and to have the Web server provide the more complex graphics needed as it can efficiently cache such data. They also decide to start using the CICS WebServer Plugin, since this will reduce the load on their CICS region, even though the overall CPU cost increases.

From our measurements presented in Chapter 5 , "CWS with Web-aware presentation logic" on page 65 we know the cost of a single business transaction using a WebServer Plugin. These costs are shown in Table 22 .

Table 22: Single business transaction using CWS with WebServer Plugin

CICS TRADERBL	CICS other	VTAM & TCP/IP	CTG	Web server	OS/390 other	Total
13.0	14.4	3.8	-	40.2	4.4	75.8

- **Transaction rates —** 30 business transactions per second.
- **CPU usage —** 2274 CPU ms, of which only 822 ms is within CICS — just within the capacity of a single CICS region, should their load balancing system fail.
- **Target system —** 45% CEC of 9672-R55.

.5 Web access Using CICS Transaction Gateway and applets

The corporation now takes the strategic step of using Java to Web-enable its Trader application. By coding the presentation logic as an applet, the Trader Company can also include all sorts of other

features, such moving graphics and sound, and also continue to use the original CICS business logic.

They initially decide on using the CTG on OS/390 due to its high scalability, and decide to implement an applet architecture. The applet will be initially designed for usage by a limited group of intranet users. These users have known software levels, reasonably powerful workstations, and are within the corporate firewall, so should work well with an architecture using CTG applets and the CTG TCP/IP protocol.

From our measurements presented in Chapter 7 , "The OS/390 CTG" on page 103 we know the cost of a single business transaction using a CTG applet and a CTG TCP/IP connection. These costs are shown in Figure 23 .

Table 23: Single business transaction using CTG Java applets

CICS TRADERBL	CICS other	VTAM & TCP/IP	CTG	Web server	OS/390 other	Total
13.0	4.8	1.9	31.7	-	10.6	62.0

- **Transaction rates —** 40 business transactions per second
- **CPU usage —** 2480 CPU ms of which 712 ms is within CICS, and still within the capacity of a single CICS region.
- **Target system —** 50% CEC of 9672-R55

9.6 Web Access Using CICS Transaction Gateway and servlets

The Trader Company decide that the time has come to open their Trader application to wider set of users on the Internet. Initially this will be a pilot to a selected number of brokers via the connection of their intranet to the Trader Company's network — an extranet. They anticipate a further increase in workload due to this expansion.

The Trader Company decides to invest its application development in Java servlets. It plans to implement some new business logic for its Internet users within the servlet and to use the Java Server Pages (JSPs) instead of applets for the presentation logic. It can re-use its CTG Java applet code with the new servlet architecture. Usage of Java servlets is also seen as a strategic decision, since the Company is interested in Enterprise Java Bean (EJB) support, and this will position them well to be able to utilize this technology.

From our measurements presented in Chapter 7 , "The OS/390 CTG" on page 103 we know the cost of a single business transaction using a servlet to access a CICS application. These costs are shown in Figure 24

Table 24: Single business transaction using CTG Java servlets

CICS TRADERBL	CICS other	VTAM & TCP/IP	CTG	Web server	OS/390 other	Total
13.0	5.7	1.8	-	96.6	18.9	136.0

- **Transaction rates —** 50 business transactions per second
- **CPU usage —** 6800 CPU ms — which clearly exceeds the capacity of the current 9672 R55 S/390 system, since it has 5 CPUs (or 5,000 CPU ms per second). One solution is to upgrade the processors — for example to the next generation of 9672. A 9672-R56 would give approximately 220% the capacity of a 9672-R55 based on the LSPR ratio for CICS.
- **Target system —** 136% CEC of a 9672-R55 or 62% of a 9672-R56.

With a machine upgrade and software conversion to servlets, the Trader Company is well placed to exploit Enterprise Java, and to open up its business to users on the Internet.

9.7 The final configuration

The Trader Company now runs five times as many business transactions as it did when using employees working at 3270 screens. They are now developing Java programs to access CICS business logic and have customers directly connected through the Internet.

A final configuration could look something like Figure 61 . We have upgraded to a more powerful processor and this is shown as a single system, but the various components of this system could be spread across members of a sysplex to achieve system availability.

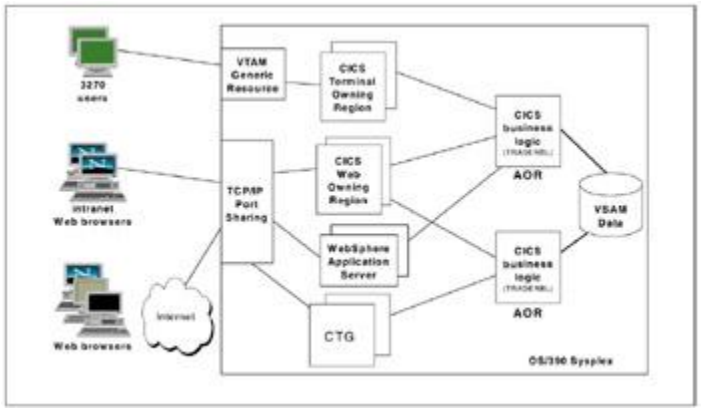


Figure 61: The final Trader configuration

TCP/IP port sharing and VTAM generic resource are used to balance work across multiple CICS regions, or across multiple WebSphere Application Servers or CTG Java gateway applications. Web clients and 3270 terminals are controlled by CICS Web Owning Regions (WORs) or Terminal Owning Regions (TORs). Multiple CICS Application Owning Regions (AORs) are used to spread the work of the CICS business logic. This requires that the data is able to be shared between them, thus VSAM Record Level Sharing (RLS) is used to allow multiple accesses to the same VSAM file.

Appendix A: Test environments

Overview

This section details the hardware and software configuration used in the laboratory performance tests.

A.1 Hardware environment

The same OS/390 hardware was used for all the measurement tests presented in this book. This configuration was a four member OS/390 Parallel Sysplex, but only two members of this sysplex were used for the measurement; one to provide a platform for the system under test, and a second to provide a platform for the network simulation driver (when using TPNS). Each sysplex member ran on a single 9672 Central Electronic Complex (CEC). This configuration comprised:

- A 9672-R55 processor (2GB storage) with 2 Cryptographic Coprocessors available where noted.
- A 9674-C05 coupling facility (2GB storage)
- Adequate RAMAC DASD to eliminate I/O constraints

For tests needing Web client simulations, either TPNS on OS/390 or the Compuware QALoad product on two nodes of an SP2 AIX processor was used.

The network connecting the OS/390 and AIX systems comprised:

- An ATM LAN emulation client (Token Ring) adaptor card on each of the AIX SP2 nodes
- An ATM (Asynchronous Transfer Mode) network
- An OSA-2 card on the S/390 processor set to operate in TCP/IP Passthru Mode to provide token-ring LAN emulation client (LEC) services via the ATM connection.

A.2 Software environments

The software levels used in all our tests were as follows; any variations or additional PTFs required are later noted in each section.

- OS/390 V2.7, including:
  - VTAM V4.7
  - DFSMS V1.5 (VSAM)
- CICS Transaction Server for OS/390 V1.3
- TPNS V3.5
- WebSphere Application Server V1.1, including:
  - IBM HTTP Server V5.1
- OS/390 Java Development Kit V1.1.8

- OS/390 CICS Transaction Gateway V3.1
- Compuware QALoad/QARun software at V4.3
- AIX V4.2.1.0

The following sections detail the pertinent configuration parameters in effect during the laboratory performance tests. These parameters are not necessarily recommended for all environments, but were in effect during our testing. You should validate these settings in your environment.

A.2.1 The 3270 Trader tests

The following CICS System Initialization Table (SIT) parameters shown in Table 25 were used during our tests.

Table 25: CICS SIT parameters

Parameter	Meaning	Value
AUXTR	Auxiliary trace flag	OFF
CMDPROT	EXEC storage checking	NO
EDSALIM	EDSA limit	260M
HPO	VTAM High Performance Option	YES
ICVR	Runaway task checking	0
INTTR	Internal tracing	ON
MN	CICS Monitoring	YES
MNCONV	Monitoring converse record option	OFF
MNEVE	Monitoring event class option	ON
MNPER	Monitoring performance class option	OFF
RLS	VSAM RLS support	NO
MROBATCH	Number of MRO requests to batch	1
SEC	Security	NO
STGPROT	Storage protection facility	NO
SPCTR	Special tracing	OFF
SUBTASKS	Number of concurrent mode TCBs	0
SYSTR	Master system trace flag	OFF
TRANSIO	Transaction isolation	NO
USERTR	User trace flag	ON

The LPA was used only for the following CICS modules which need to be located in the LPA: DFHIRP, DFHDSPEX, DFHCSVC.

A.2.2 CICS Web support with the 3270 Web bridge

The same CICS SIT parameters as used for the 3270 Trader tests in Table 25 on page 163 , were used for the 3270 Web bridge tests. The SIT parameters modified for the 3270 Web bridge tests are documented in Table 26 .

Table 26: CICS SIT parameters for CICS Web support

Parameter	Meaning	Value
TCPIP	TCP/IP support for HTTP and IIOP	YES
WEBDELAY	CWS time-out and garbage collection	1,1

The CICS TCPIPSERVICE definition used to configure the HTTP support for our test CICS region is shown in Table 27 .

Table 27: TCPIPSERVICE definition

Parameter	Meaning	Value
BACKLOG	TCP/IP queue length	128
SOCKETCLOSE	HTTP persistent connection time-out	000010
SSL	SSL security	NO
TSQPREFIX	TSQ template prefix for Web I/O	default

The eNetwork Communications Server configuration parameters used to configure TCP/IP support are listed in Table 28 .

Table 28: TCP/IP parameters

Parameter	Meaning	Value
MTU (on GATEWAY statement)	Maximum transmission unit size	4500
SOMAXCONN	Socket request queue length	1024
ARPAGE	Time-out of arp cache	20
TCPSENDBFRSIZE (on TCPCONFIG statement)	Size of TCP/IP send buffer	65536
TCPRCVBUFRSIZE (on TCPCONFIG statement)	Size of TCP/IP receive buffer	65536

The packet size for the AIX adapter card was allowed to default to 1,500 bytes, as using larger values caused network instability.

A.2.3 CICS Web support with Web-aware presentation logic

CWS direct connection

The same CICS SIT and TCPIPSERVICE parameters as documented in Appendix A.2.2 , "CICS Web support with the 3270 Web bridge" on page 164 , were used for the CWS tests with Web-aware presentation logic.

The only difference was that the TCPIPSERVICE SOCKETCLOSE value was set to 20 seconds as

opposed to 10. This enabled persistent HTTP connections to be used with the longer think time imposed by the larger number of clients. The value of SOCKETCLOSE was set to 0 when persistent HTTP connections were not used.

CICS WebServer Plugin

When using CWS and the CICS WebServer Plugin, the following parameters were used in the OS/390 Web server configuration file httpd.conf

```
DNS-Lookup     off
MaxActiveThreads  150
MaxPersistRequest 9999
ServerPriority -20
Service    /iycuzc14/*           /etc/dfhwbapi.so:DFHService
Service    /IYCUZC14/*           /etc/dfhwbapi.so:DFHService
PersistTimeout  1 minute
CacheLocalMaxBytes 6 M
```

A.2.4 CWS with SSL

The fix for the following CICS APAR was applied to the system:

- **PQ23421** - Enabling APAR for CTS 1.3 SSL

The fixes for the following System SSL APARs were applied to the system:

- **OW37136** - GA APAR for SSL base and strong crypto
- **PQ31399** - Provide full support for SSL session ID's
- **OW40099** - System SSL - externalization of gsk_user_set()
- **OW40974** - System SSL session ID comparison failure
- **OW38773** - System SSL utility program gskkyman generates csr files which do not contain state/province information

The microcode fix RPQ8P1987, feature code 834, was applied to the S3/90 system to enable the Cyrptographic Coprocessor Facility to assist in SSL handshaking.

The same CICS SIT and TCPIPSERVICE parameters as documented in Appendix A.2.3 , "CICS Web support with Web-aware presentation logic" on page 165 , were used for the CWS SSL tests. The only difference was that the TCIPIPSERVICE parameters SOCKETCLOSE was set to 10 and the following SIT parameters in Table 29 were used.

Table 29: CICS SIT parameters for CICS Web support with SSL

Parameter	Meaning	Value
DSALIM	Limit of dynamic storage areas	4M
SSLTCBS	Number of TCBs for SSL processing	70

In addition, the new SIT parameters SSLDELAY and ENCRYPTION and the TCPIPSERVICE parameter SOCKETCLOSE were modified during each of the tests to produce the desired SSL test scenario. A summary of the meaning of these new SSL SIT parameters is given in Table 30 .

Table 30: SSL configuration parameters

Parameter	Value	Meaning
ENCRYPTION	WEAK | NORMAL | STRONG	This parameter controls the cipher spec for the SSL record protocol negotiated during the SSL handshake. WEAK specifies the following list of ciphers: - RC4 encryption with a 40-bit key and an MD5 MAC - RC2 encryption with a 40-bit key and an MD5 MAC - No encryption with an MD5 MAC - No encryption with an SHA MAC. NORMAL specifies the following list of ciphers: - DES encryption with a 56-bit key and an SHA MAC - RC4 encryption with a 40-bit key and an MD5 MAC - RC2 encryption with a 40-bit key and an MD5 MAC - No encryption with an MD5 MAC - No encryption with an SHA MAC. STRONG Specifies the following list of ciphers: - Triple DES encryption with a 168-bit key and an SHA MAC - RC4 encryption with a 128-bit key and an MD5 MAC - RC4 encryption with a 128-bit key and an SHA MAC - DES encryption with a 56-bit key and an SHA MAC - RC4 encryption with a 40-bit key and an MD5 MAC - RC2 encryption with a 40-bit key and an MD5 MAC - No encryption with an MD5 MAC

		▪ No encryption with an SHA MAC.
SSLDELAY	{600|number}	This delay specifies the length of time in seconds for which CICS retains session IDs for SSL connections. Session IDs are tokens that represent a secure connection between a client and an SSL server. While the session ID is retained by CICS within the SSLDELAY period, CICS can continue to communicate with the client without the significant overhead of an SSL handshake. The value is a number of seconds in the range 0 through 86400.
SSLTCBS	{8|number}	This parameter specifies the number of CICS subtask TCBs that will be dedicated to processing secure sockets layer connections. The value is a number in the range 0 to 255. It controls the number of simultaneous SSL connections that CICS can establish. A value of 0 means that no SSL connections are to be established. This number is independent of and in addition to the TCBs specified in MAXOPENTCBS. The TCBs used by SSL can consume considerable storage below 16MB.

A.2.5 CICS Transaction Gateway

The same CICS SIT parameters as documented in Appendix A.2.1 , "The 3270 Trader tests" on page 163 , were used for the CICS Transaction Gateway tests.

When using the CTG applet architecture the following parameters were used in the ctg.ini configuration file for the CTG Java Gateway application.

```
maxconnect=1000
maxworker=75
protocol@tcp.handler=com.ibm.ctg.server.TCPHandler
protocol@tcp.parameters=port=2006; sotimeout=9000; connecttimeout=2000;
           idletimeout=600000; pingfrequency=600000
protocol@http.handler=com.ibm.ctg.server.HttpHandler
protocol@http.parameters=port=8080; sotimeout=9000; connecttimeout=2000;
           idletimeout=120000; pingfrequency=600000
```

The CTG values for initworker and initconnect are not given because our performance tests were run after the workload had stabilized; thus only the maximum thread values, not the initial values, are of interest.

When using the CTG servlet architecture, the following parameters were used in the OS/390 Web server configuration file httpd.conf :

```
    MaxActiveThreads  140
    MaxPersistRequest 9999
    ServerPriority -20
    PersistTimeout  1 minute
    CacheLocalMaxBytes 6 M
```

Appendix B: Performance data

Overview

This appendix contains all the unprocessed performance data from our laboratory workloads. This data was collected from RMF reports. Each test was run in isolation with no other work active within the OS/390 image. All Web client simulation software was executed on a separate system.

The CPU usage apportioned to each address space is reported together with the total CPU usage in the system. The RMF correction factor has already been applied to all the data; this factor is used to apportion to each address space that amount of CPU which is not quantifiable. The CPU usage in the tables is presented as *percentage usage of a single R55 CPU* . Thus the maximum possible total CPU usage is 500 CPU% (or 5 CPU seconds per second) on our 9672-R55 test system, which contains five CPUs. Note that the response times are not reported in our data, since all the recorded times were less than one second. This is due to the simple nature of our test programs and the high network capacity of our test network.

The definitions of the terms used in the tables are as follows:

CICS CPU	is the recorded CPU usage charged to the CICS address space, expressed as a percentage of one processor.
TCP/IP CPU	is the recorded CPU usage charged to the TCP/IP address space, expressed as a percentage of one processor.
VTAM CPU	is the recorded CPU usage charged to the VTAM address space, expressed as a percentage of one processor.
Web server CPU	is the recorded CPU usage charged to the OS/390 Web server address space, expressed as a percentage of one processor.
CTG CPU	is the recorded CPU usage charged to the CTG Java gateway application address space, expressed as a percentage of one processor.
Total CPU	is the total CPU usage within the OS/390 system, expressed as a percentage of one processor.
Throughput	is defined for each section.
Total CPU ms/request	is the total OS/390 CPU cost per request. It is calculated by multiplying the total CPU% by 10 to convert to CPU ms, then dividing by the throughput.

B.1 3270 Trader application

Table 31 details CPU usage when running a 3270 Trader workload using TPNS. The results were recorded using RMF monitoring. Throughput is defined as Trader business transactions per second; one business transaction consists of 10 CICS tasks. For a discussion of this data refer to 3.2 , "Measured CPU usage" on page 45 .

Table 31: 3270 Trader CPU usage

Throughput	CICS CPU%	VTAM CPU%	Total CPU%	Total CPU ms/request
9.0	30.8	0.9	41.6	46.2
10.6	37.1	1.1	48.2	45.5
12.1	41.1	1.3	52.0	43.0
15.1	50.2	1.5	61.2	40.5

B.2 CWS with the 3270 Web bridge

The data in Table 32 and Table 33 shows the results for the tests using the 3270 Web bridge. A CWS direct connection was utilized for these tests. Throughput is defined as Web requests per second; 200 simulated Web browser clients were in use for all tests. For a discussion of this data refer to Chapter 4 , "CWS with the 3270 Web bridge" on page 51 .

Table 32: 3270 Web bridge, continuous pseudo-conversation

Throughput	CICS CPU%	TCPIP & VTAM CPU%	Total CPU%	Total CPU ms/request
15.72	14.11	3.08	20.7	13.1
20.88	18.02	3.95	25.3	12.1
30.96	25.36	5.31	33.9	10.9
59.9	46.26	9.14	58.3	9.7
111.8	82.73	15.40	101.2	9.0

Table 33: 3270 Web bridge, non-continuous pseudo-conversation

Throughput	CICS CPU%	TCPIP & VTAM CPU%	Total CPU%	Total CPU ms/request
15.78	15.52	2.90	22.0	13.9
21.02	20.46	3.75	27.7	13.2
30.00	29.66	4.64	38.0	12.7
58.94	58.98	8.43	72.4	12.3
97.42	110.86	12.30	127.6	13.1

B.3 CWS with Web-aware presentation logic

In this section we present the results of our tests using CICS Web support and new HTTP based Web-aware presentation logic, first using a direct connection to CWS and then using the CICS WebServer Plugin. For discussion of this data refer to Chapter 5 , "CWS with Web-aware presentation logic" on page 65 .

B.3.1 CWS and a direct connection

Table 34 details the actual HTTP data stream sizes sent and received by CICS Web support in our test measurements using a direct connection. These data sizes include the HTTP header information. Send data tests were implemented using the HTTP GET method, and receive data tests were implemented using the HTTP POST method.

Table 34: CWS direct connection, data transmission sizes

Nominal data size	Send or receive	Application style	Persistent HTTP connections	Data received by CICS (bytes)	Data sent from CICS (bytes)
100 bytes	send	WEB API	persistent	284	194

5KB	send	WEB API	persistent	284	5094
15KB	send	WEB API	persistent	284	15094
32KB	send	WEB API	persistent	284	32094
33KB	send	WEB API	persistent	284	33094
50KB	send	WEB API	persistent	284	50094
100 bytes	receive	WEB API	persistent	421	116
5KB	receive	WEB API	persistent	5321	116
15KB	receive	WEB API	persistent	15321	116
32KB	receive	WEB API	persistent	32321	116
33KB	receive	WEB API	persistent	33321	116
50KB	receive	WEB API	persistent	50321	116
100 bytes	send	WEB API	non-persistent	279	170
5KB	send	WEB API	non-persistent	279	5070
15KB	send	WEB API	non-persistent	279	15070
32KB	send	WEB API	non-persistent	279	32070
33KB	send	WEB API	non-persistent	279	33070
50KB	send	WEB API	non-persistent	279	50070
100 bytes	receive	WEB API	non-persistent	421	116
5KB	receive	WEB API	non-persistent	5321	116
15KB	receive	WEB API	non-persistent	15321	116
32KB	receive	WEB API	non-persistent	32321	116
33KB	receive	WEB API	non-persistent	33321	116
50KB	receive	WEB API	non-persistent	50321	116
5KB	send	COMMAREA	persistent	284	5050
5KB	receive	COMMAREA	persistent	5323	153

The following data, presented in Table 35 on page 174 through Table 58 on page 180 , shows the results for a CWS direct connection with Web-aware presentation logic using the CICS WEB API. Throughput is defined as Web requests per second. 200 simulated Web browser clients were in use for all tests.

Table 35: CWS direct connection, persistent HTTP connection, 100 byte send

Throughput	CICS CPU%	VTAM CPU%	TCP/IP CPU%	Total CPU%	CPU ms/request
19.65	7.30	0.57	1.25	13.6	6.9
39.46	13.36	0.65	1.69	19.2	4.9
65.39	20.25	0.85	2.30	26.8	4.1
97.55	29.38	1.16	2.90	36.7	3.8
189.40	53.02	1.44	4.54	62.1	3.3

Table 36: CWS direct connection, persistent HTTP connection, 5KB send

Throughput	CICS CPU%	VTAM CPU%	TCP/IP CPU%	Total CPU%	CPU ms/request
19.63	8.11	0.57	1.42	134.0	7.1
39.57	14.40	0.76	2.17	20.9	5.3
65.45	22.29	1.08	2.88	29.6	4.5
97.49	31.55	1.27	3.90	40.0	4.1
189.30	57.98	1.76	5.71	68.9	3.6

Table 37: CWS direct connection, persistent HTTP connection, 15KB send

Throughput	CICS CPU%	VTAM CPU%	TCP/IP CPU%	Total CPU%	CPU ms/request
19.67	15.35	0.69	2.66	15.4	7.8
39.45	23.60	0.87	2.73	23.6	6.0
65.33	33.95	1.29	3.76	34.0	5.2
97.58	46.65	1.70	4.98	46.7	4.8
189.90	81.15	2.61	7.60	81.2	4.3

Table 38: CWS direct connection, persistent HTTP connection, 32KB send

Throughput	CICS CPU%	VTAM CPU%	TCP/IP CPU%	Total CPU%	CPU ms/request
19.60	10.95	0.92	2.38	18.0	9.2
39.49	20.05	1.44	3.84	28.7	7.3
65.15	31.25	2.06	5.27	41.9	6.4
97.40	45.48	2.55	6.54	57.9	5.9
178.80	77.24	9.50	15.26	105.3	5.9

Table 39: CWS direct connection, persistent HTTP connection, 33KB send

Throughput	CICS CPU%	VTAM CPU%	TCP/IP CPU%	Total CPU%	CPU ms/request
19.53	11.16	0.93	2.39	17.8	9.1
39.51	20.47	1.44	3.83	29.5	7.5
65.00	31.97	1.94	5.25	42.3	6.5
97.08	46.15	2.55	6.86	58.6	6.0
173.90	77.66	9.60	15.79	104.9	6.0

Table 40: CWS direct connection, persistent HTTP connection, 50KB send

Throughput	CICS CPU%	VTAM CPU%	TCP/IP CPU%	Total CPU%	CPU ms/request
19.61	13.02	1.15	2.68	20.3	10.4
39.38	23.90	1.76	4.45	33.5	8.5
65.09	37.69	2.68	7.05	50.6	7.8
97.22	52.07	4.03	12.64	71.9	7.4
116.10	62.85	10.40	14.37	90.9	7.8

Table 41: CWS direct connection, persistent HTTP connection, 100 byte receive

Throughput	CICS CPU%	VTAM CPU%	TCP/IP CPU%	Total CPU%	CPU ms/request
19.66	7.93	0.57	1.13	14.5	7.3
39.42	14.40	0.76	1.78	21.2	5.4
65.31	22.37	0.96	2.28	29.1	4.5
97.47	32.20	1.27	2.88	39.8	4.1
189.50	59.99	1.65	4.39	69.6	3.7

Table 42: CWS direct connection, persistent HTTP connection, 5KB receive

Throughput	CICS CPU%	VTAM CPU%	TCP/IP CPU%	Total CPU%	CPU ms/request
19.62	11.15	0.93	2.39	18.1	9.2
39.84	20.11	1.20	3.49	28.3	7.1
65.39	31.45	1.49	4.94	41.1	6.3
97.27	45.05	1.89	6.56	57.0	5.9
189.90	83.05	2.15	9.57	98.0	5.2

Table 43: CWS direct connection, persistent HTTP connection, 15KB receive

Throughput	CICS CPU%	VTAM CPU%	TCP/IP CPU%	Total CPU%	CPU ms/request
19.65	15.71	1.11	3.59	24.0	12.2
40.07	29.64	1.72	5.97	40.6	10.1
65.39	46.21	2.21	8.62	60.3	9.2
97.04	65.76	2.71	11.48	83.2	8.6
189.00	115.04	2.65	15.98	137.0	7.2

Table 44: CWS direct connection, persistent HTTP connection, 32KB receive

Throughput	CICS CPU%	VTAM CPU%	TCP/IP CPU%	Total CPU%	CPU ms/request
19.64	22.98	1.52	5.39	33.3	17.0
39.72	44.05	2.54	9.50	59.4	15.0
65.46	70.10	2.90	13.87	90.2	13.8
97.01	98.32	3.07	17.99	122.6	12.6
135.80	128.55	3.91	22.16	158.0	11.6

Table 45: CWS direct connection, persistent HTTP connection, 33KB receive

Throughput	CICS CPU%	VTAM CPU%	TCP/IP CPU%	Total CPU%	CPU ms/request
19.73	23.94	1.52	5.61	34.5	17.5
39.95	46.36	2.76	9.94	62.2	15.6
65.22	72.96	3.33	14.59	94.1	14.4
96.82	102.47	2.54	18.27	126.6	13.1
130/10	131.05	4.44	22.39	161.3	12.4

Table 46: CWS direct connection, persistent HTTP connection, 50KB receive

Throughput	CICS CPU%	VTAM CPU%	TCP/IP CPU%	Total CPU%	CPU ms/request
19.73	26.98	1.84	7.03	39.2	19.9
39.76	52.22	3.28	12.56	71.5	18.0
65.31	82.86	4.48	19.01	109.6	16.8
94.65	115.31	2.63	23.90	145.0	15.3

Table 47: CWS direct connection, non-persistent HTTP connection, 100 byte send

Throughput	CICS CPU%	VTAM CPU%	TCP/IP CPU%	Total CPU%	CPU ms/request
19.6	17.80	0.81	2.56	17.8	9.1
39.34	18.98	1.34	4.83	28.4	7.2
65.34	29.74	2.08	6.37	41.6	6.4
97.53	42.18	2.25	8.55	56.4	5.8
189.50	76.66	2.83	13.05	95.8	5.1

Table 48: CWS direct connection, non-persistent HTTP connection, 5KB send

Throughput	CICS CPU%	VTAM CPU%	TCP/IP CPU%	Total CPU%	CPU ms/request
19.52	10.81	0.93	3.20	18.6	9.5
39.54	19.66	1.45	5.43	30.2	7.6
65.19	30.71	2.30	8.05	44.4	6.8
97.62	43.66	2.24	10.19	59.6	6.1
189.60	80.50	3.14	15.80	102.9	5.4

Table 49: CWS direct connection, non-persistent HTTP connection, 15KB send

Throughput	CICS CPU%	VTAM CPU%	TCP/IP CPU%	Total CPU%	CPU ms/request
19.34	11.47	1.17	4.43	20.9	10.8
39.47	21.25	2.14	7.72	34.6	8.8
64.88	33.12	3.06	11.12	50.7	7.8
97.75	48.05	2.89	13.98	68.3	7.0
189.90	88.50	4.10	20.80	116.9	6.2

Table 50: CWS direct connection, non-persistent HTTP connection, 32KB send

Throughput	CICS CPU%	VTAM CPU%	TCP/IP CPU%	Total CPU%	CPU ms/request
19.30	12.82	1.51	6.03	24.0	12.4
39.44	24.05	2.79	10.34	40.6	10.3
64.86	37.96	3.92	14.78	59.9	9.2
97.19	54.31	3.06	17.59	79.0	8.1
164.80	85.91	8.32	22.94	120.8	7.3

Table 51: CWS direct connection, non-persistent HTTP connection, 33KB send

Throughput	CICS CPU%	VTAM CPU%	TCP/IP CPU%	Total CPU%	CPU ms/request

19.32	13.01	1.50	6.01	24.2	12.5
39.57	24.64	2.79	10.58	41.5	10.5
64.98	38.55	3.69	14.86	60.5	9.3
97.41	55.57	3.28	18.16	80.4	8.3
162.20	86.80	7.55	21.59	119.3	7.4

Table 52: CWS direct connection, non-persistent HTTP connection, 50KB send

Throughput	CICS CPU%	VTAM CPU%	TCP/IP CPU%	Total CPU%	CPU ms/request
19.32	14.39	1.95	7.32	27.2	14.1
39.42	27.08	3.07	12.40	46.2	11.7
65.32	42.73	3.08	16.63	65.9	10.1
97.50	62.25	4.86	19.99	90.5	9.3

Table 53: CWS direct connection, non-persistent HTTP connection, 100 byte receive

Throughput	CICS CPU%	VTAM CPU%	TCP/IP CPU%	Total CPU%	CPU ms/request
19.64	11.06	0.80	2.53	18.3	9.3
39.32	20.14	1.33	4.34	30.2	7.6
65.15	31.33	2.07	6.45	43.2	6.6
97.47	44.80	2.46	8.62	59.3	6.1
189.50	82.36	3.90	13.85	103.4	5.5

Table 54: CWS direct connection, non-persistent HTTP connection, 5KB receive

Throughput	CICS CPU%	VTAM CPU%	TCP/IP CPU%	Total CPU%	CPU ms/request
19.47	13.73	1.02	3.56	22.0	11.3
39.49	25.40	1.87	6.09	36.8	9.3
65.14	40.03	2.92	9.11	55.6	8.5
97.34	56.99	3.017	11.64	75.0	7.7
190.00	105.29	3.45	18.32	130.4	6.9

Table 55: CWS direct connection, non-persistent HTTP connection, 15KB receive

Throughput	CICS CPU%	VTAM CPU%	TCP/IP CPU%	Total CPU%	CPU ms/request
19.32	18.52	1.33	5.08	28.5	14.7
39.46	35.08	2.49	8.60	49.5	12.5
64.91	55.42	3.73	12.40	74.9	11.5
97.55	80.24	3.46	16.20	103.4	10.6
189.90	147.30	3.18	24.52	178.4	9.4

Table 56: CWS direct connection, non-persistent HTTP connection, 32KB receive

Throughput	CICS CPU%	VTAM CPU%	TCP/IP CPU%	Total CPU%	CPU ms/request
19.49	26.37	1.73	6.59	38.1	19.5

39.53	50.39	3.07	11.09	67.9	17.2
54.40	79.79	2.68	15.76	101.5	15.5
97.51	117.03	2.45	21.17	144.0	14.8
126.10	150.22	2.31	25.33	181.7	14.4

Table 57: CWS direct connection, non-persistent HTTP connection, 33KB receive

Throughput	CICS CPU%	VTAM CPU%	TCP/IP CPU%	Total CPU%	CPU ms/request
19.56	27.37	1.73	6.70	39.2	20.0
39.46	52.32	2.96	11.19	69.7	17.7
65.44	83.05	2.46	15.86	104.9	16.0
97.14	123.23	2.98	22.17	151.9	15.6
119.00	152.91	5.59	26.77	119.0	15.9

Table 58: CWS direct connection, non-persistent HTTP connection, 50KB receive

Throughput	CICS CPU%	VTAM CPU%	TCP/IP CPU%	Total CPU%	CPU ms/request
19.78	29.90	2.05	7.99	43.3	21.9
39.54	57.35	3.38	13.85	77.9	19.7
65.17	90.54	2.13	19.52	115.5	17.7
95.55	136.39	2.45	27.00	169.4	17.7

The data in Table 59 and Table 60 is from tests that used the COMMAREA manipulation technique instead of the WEB API in the Web-aware presentation logic. Both tests used a persistent HTTP connection, and 200 simulated Web browser clients.

Table 59: CWS direct connection, COMMAREA manipulation, 5KB send

Throughput	CICS CPU%	VTAM CPU%	TCP/IP CPU%	Total CPU%	CPU ms/request
19.66	9.60	0.94	1.89	17.0	8.7
39.36	16.37	1.24	2.75	24.9	6.3
65.18	25.01	1.67	3.98	34.5	5.3
97.70	35.17	2.19	5.11	46.3	4.7
189.30	62.73	3.43	7.88	77.7	4.1

Table 60: CWS direct connection, COMMAREA manipulation, 5KB receive

Throughput	CICS CPU%	VTAM CPU%	TCP/IP CPU%	Total CPU%	CPU ms/request
19.88	11.48	1.25	3.01	20.0	10.0
39.94	19.97	1.66	4.61	30.0	7.5
65.49	30.25	1.56	6.00	41.7	6.4
97.44	43.22	2.55	8.11	57.4	5.9
190.90	80.10	3.89	12.00	99.7	5.2

B.3.2 CWS and the CICS WebServer Plugin

Table 61 details the actual HTTP data stream sizes sent and received by CICS Web support in our test measurements with the CICS WebServer Plugin. Send data tests were implemented using the HTTP GET method, and receive data tests implemented using the HTTP POST method.

Table 61: CWS and WebServer Plugin, data transmission sizes

Nominal data size	Send/receive	Application style	Persistent HTTP connections	Data received by CICS (bytes)	Data sent CICS (bytes)
100 bytes	send	WEB API	persistent	293	194
5KB	send	WEB API	persistent	293	5094
15KB	send	WEB API	persistent	293	15094
32KB	send	WEB API	persistent	293	32094
100 bytes	receive	WEB API	persistent	430	116
5KB	receive	WEB API	persistent	5330	116
15KB	receive	WEB API	persistent	15330	116
32KB	receive	WEB API	persistent	32330	116
100 bytes	send	WEB API	non-persistent	288	194
5KB	send	WEB API	non-persistent	288	5094
15KB	send	WEB API	non-persistent	288	15094
32KB	send	WEB API	non-persistent	288	32094

The following data, presented in Table 62 on page 182 through Table 73 on page 185 , shows the results for the CWS tests using the CICS WebServer Plugin, with Web-aware presentation logic using the CICS WEB API. Throughput is defined as Web requests per second. For all tests 70 simulated Web browser clients were in use.

Table 62: WebServer Plugin, persistent HTTP connection, 100 bytes send

Throughput	CICS CPU%	Web Server CPU%	VTAM CPU%	TCP/IP CPU%	Total CPU%	CPU ms/request
13.63	3.33	10.80	0.13	0.80	19.6	14.4
16.92	4.04	12.63	0.13	0.91	22.0	13.0
19.49	4.57	14.34	0.25	1.02	24.5	12.6
32.51	7.15	22.52	0.36	1.43	35.5	10.9
60.47	12.38	37.02	0.68	2.16	56.1	9.3

Table 63: WebServer Plugin, persistent HTTP connection, 5KB send

Throughput	CICS CPU%	Web Server CPU%	VTAM CPU%	TCP/IP CPU%	Total CPU%	CPU ms/request
13.20	3.58	10.47	0.27	0.93	19.8	15.0

16.44	4.27	12.31	0.26	1.04	22.3	13.5
21.19	5.39	15.29	0.38	1.35	26.5	12.5
30.95	7.44	20.99	0.48	1.56	34.6	11.2
56.49	12.85	35.82	0.91	2.50	60.0	9.9

Table 64: WebServer Plugin, persistent HTTP connection, 15KB send

Throughput	CICS CPU%	Web Server CPU%	VTAM CPU%	TCP/IP CPU%	Total CPU%	CPU ms/request
13.60	4.34	13.41	0.38	1.15	23.5	17.3
16.81	5.08	15.74	0.50	1.24	26.7	15.9
22.16	6.49	20.44	0.72	1.56	33.3	15.0
32.27	9.02	27.87	0.93	1.97	44.0	13.6
60.07	16.09	47.70	1.77	3.11	72.6	12.1

Table 65: WebServer Plugin, persistent HTTP connection, 32KB send

Throughput	CICS CPU%	Web Server CPU%	VTAM CPU%	TCP/IP CPU%	Total CPU%	CPU ms/request
13.58	5.45	16.34	0.62	1.49	28.1	2.1
16.78	6.51	19.42	0.72	1.69	32.5	1.9
22.05	8.44	24.98	1.17	2.23	40.7	1.8
32.09	11.7	34.65	1.48	2.84	54.7	1.7
59.52	21.07	60.14	2.74	4.61	92.4	1.6

Table 66: WebServer Plugin, persistent HTTP connection, 100 byte receive

Throughput	CICS CPU%	Web Server CPU	VTAM CPU%	TCP/IP CPU%	Total CPU%	CPU ms/request
17.21	4.68	12.90	0.51	1.01	23.7	13.7
22.98	5.97	16.58	0.61	1.22	28.7	12.5
34.18	8.43	23.31	0.70	1.52	38.3	11.2
66.63	15.43	40.47	1.12	2.35	63.5	9.5
241.90	58.43	153.52	1.50	5.99	224.1	9.3

Table 67: WebServer Plugin, persistent HTTP connection, 5KB receive

Throughput	CICS CPU%	Web Server CPU%	VTAM CPU%	TCP/IP CPU%	Total CPU%	CPU ms/request
17.28	4.89	12.28	0.75	1.88	24.3	14.1
23.14	6.17	16.10	0.73	2.18	29.7	12.8
34.18	8.83	22.77	0.93	2.79	39.5	11.6
66.52	16.61	41.31	1.33	4.43	68.0	10.2
224.20	59.15	179.38	1.69	15.69	260.6	11.6

Table 68: WebServer Plugin, persistent HTTP connection, 15KB receive

Throughput	CICS CPU%	Web Server CPU%	VTAM CPU%	TCP/IP CPU%	Total CPU%	CPU ms/request
17.28	5.52	17.64	0.84	3.12	31.2	18.1
23.12	7.07	22.95	0.93	3.94	39.3	17.0
34.14	10.23	32.83	1.12	5.85	54.2	15.9
66.67	19.35	58.61	1.62	12.87	96.5	14.5
175.20	56.06	193.24	3.46	94.08	351.6	20.1

Table 69: WebServer Plugin, persistent HTTP connection, 32KB receive

Throughput	CICS CPU%	Web Server CPU%	VTAM CPU%	TCP/IP CPU%	Total CPU%	CPU ms/request
17.15	7.13	25.87	1.03	5.86	44.2	25.7
22.98	9.33	34.41	1.35	8.43	58.7	25.5
33.49	13.27	48.79	1.64	13.93	81.8	24.2
66.60	26.45	94.40	2.35	44.47	172.1	25.8
100.30	42.91	153.77	2.75	129.04	333.0	33.2

Table 70: WebServer Plugin, non-persistent HTTP connection, 100 byte send

Throughput	CICS CPU%	Web Server CPU%	VTAM CPU%	TCP/IP CPU%	Total CPU%	CPU ms/request
13.57	3.27	12.29	0.78	1.83	22.8	16.8
16.88	3.96	14.42	0.89	2.04	25.7	15.2
22.13	5.04	18.44	0.98	2.71	31.6	14.3
32.44	7.07	25.35	1.41	3.54	41.5	12.8
60.06	12.39	43.47	2.25	5.86	67.9	11.3

Table 71: WebServer Plugin, non-persistent HTTP connection, 5KB send

Throughput	CICS CPU%	Web Server CPU%	VTAM CPU%	TCP/IP CPU%	Total CPU%	CPU ms/request
13.51	3.60	13.11	0.51	1.80	23.4	17.3
16.72	4.37	15.36	0.50	2.00	26.6	15.9
22.12	5.55	20.03	0.72	2.65	33.3	15.1
32.18	7.79	27.45	1.16	3.61	43.9	13.6
59.86	13.86	47.17	1.90	5.70	72.6	1.21

Table 72: WebServer Plugin, non-persistent HTTP connection, 15KB send

Throughput	CICS CPU%	Web Server CPU%	VTAM CPU%	TCP/IP CPU%	Total CPU%	CPU ms/request
13.57	4.27	14.32	0.63	2.26	25.8	19.0
16.81	5.14	17.27	0.73	2.69	30.0	17.8

22.07	6.54	21.66	0.95	3.33	36.7	16.6
32.23	9.18	29.94	1.38	4.59	49.1	15.2
59.37	16.23	51.78	2.54	7.40	81.7	13.8

Table 73: WebServer Plugin, non-persistent HTTP connection, 32KB send

Throughput	CICS CPU%	Web Server CPU%	VTAM CPU%	TCP/IP CPU%	Total CPU%	CPU ms/request
13.51	5.36	16.92	1.10	2.92	30.8	22.8
16.80	6.55	19.77	1.19	3.33	35.3	21.0
21.99	8.37	25.82	1.51	4.42	44.6	20.3
32.07	11.73	35.87	2.03	6.09	59.9	18.7
58.57	20.79	61.50	2.74	9.52	98.7	16.9

B.4 CWS with SSL

In this section we present the results of our tests using SSL with CICS Web support and new HTTP based Web-aware presentation logic, using both a direct connection to CWS and the CICS WebServer Plugin. For further discussion of this data refer to Chapter 6 , "SSL with CWS" on page 85 .

The measurements were generated using HTTP GET requests and a simple CICS WEB API program that sent the requested amount of data. The SSL handshake measurements used non-persistent HTTP connections and the CICS application returned 1 byte of data. The SSL data transmission measurements used persistent HTTP connection and thus incurred no SSL handshake costs. 70 Web browser clients were in use for all tests.

Table 74 and Table 75 detail the actual HTTP data stream sizes sent and received in the CWS SSL test measurements.

Table 74: Data transmission sizes, CWS direct connection

Nominal data size	Data received by CICS (bytes)	Data sent from CICS (bytes)
1 bytes	284	95
8KB	284	8095
16KB	284	16095

Table 75: Data transmission sizes, WebServer Plugin

Nominal data size	Data received by CICS (bytes)	Data sent from CICS (bytes)
1 byte	293	95
8KB	293	8095
16KB	293	16095

B.4.1 SSL handshakes with a CWS direction connection

The following data, presented in Table 76 on page 187 through Table 99 on page 194 , shows the results for the SSL handshake tests with a CWS direct connection. All the handshake tests used non-persistent HTTP connections and sent 1 byte of data from the CICS application. The results in Table 79 and Table 80 on page 188 marked *with crypto* used the S/390 Cryptographic Coprocessor to assist in the CPU costs of SSL handshaking. The *Non-SSL* figures are the cost of establishing a non-persistent HTTP connection.

Table 76: Non-SSL, non-persistent HTTP connection, CWS direct connection

Throughput	CICS CPU%	VTAM CPU%	TCP/IP CPU%	Total CPU%	CPU ms/request
13.57	6.79	0.67	1.73	16.9	12.5
16.93	8.41	0.79	2.10	18.7	11.0
22.23	10.54	0.89	2.54	21.9	9.9
32.47	14.79	1.22	3.54	27.3	8.4
60.73	25.77	2.09	5.80	40.4	6.7

Table 77: SSL full handshake, 1024-bit key, CWS direct connection

Throughput	CICS CPU%	VTAM CPU%	TCP/IP CPU%	Total CPU%	CPU ms/request
12.97	127.02	0.83	2.18	136.4	105.2
15.76	153.15	0.93	2.48	163.0	92.6
20.29	196.41	1.13	3.08	207.0	102
28.52	275.69	1.53	4.29	287.9	100.9
43.58	419.19	2.34	6.21	434.3	99.7

Table 78: SSL full handshake, 512-bit key, CWS direct connection

Throughput	CICS CPU%	VTAM CPU%	TCP/IP CPU%	Total CPU%	CPU ms/request
13.20	38.96	0.89	2.21	48.7	36.9
16.27	47.6	0.98	2.51	57.8	35.5
21.38	61.33	1.29	3.33	72.5	33.9
30.87	87.35	1.80	4.44	100.2	32.5
50.55	147.15	2.72	6.80	163.0	32.2

Table 79: SSL full handshake with crypto, 1024-bit key, CWS direct connection

Throughput	CICS CPU%	VTAM CPU%	TCP/IP CPU%	Total CPU%	CPU ms/request
12.94	19.42	4.68	2.40	34.1	26.4
15.90	24.59	4.71	2.71	39.7	25.0
20.90	30.16	4.97	3.58	46.5	22.2
29.81	42.01	5.42	4.63	59.9	20.1
50.66	72.86	6.46	6.46	93.9	18.5

Table 80: SSL full handshake with crypto, 512-bit key, CWS direct connection

Throughput	CICS CPU%	VTAM CPU%	TCP/IP CPU%	Total CPU%	CPU ms/request

13.01	19.49	4.66	2.27	34.0	26.1
16.12	23.68	4.59	2.59	38.4	23.8
21.2	30.59	4.85	3.46	46.4	21.9
30.45	42.60	5.40	4.72	60.3	19.8
51.87	75.21	6.45	6.56	96.2	18.5

Table 81: SSL null handshake, 1024-bit key, CWS direct connection

Throughput	CICS CPU%	VTAM CPU%	TCP/IP CPU%	Total CPU%	CPU ms/request
12.90	12.17	0.91	2.35	20.8	16.1
15.96	14.50	1.02	2.67	23.4	14.7
20.57	18.18	1.35	3.44	28.0	13.6
29.07	24.66	1.78	4.50	35.8	12.3
49.16	39.48	2.27	6.37	52.5	10.7

Table 82: SSL null handshake, 512-bit key, CWS direct connection

Throughput	CICS CPU%	VTAM CPU%	TCP/IP CPU%	Total CPU%	CPU ms/request
12.91	12.09	0.91	2.47	20.8	16.1
15.82	14.51	1.02	2.67	23.3	14.7
20.48	18.08	1.35	3.44	27.8	13.6
29.07	24.70	1.78	4.51	35.8	12.3
49.32	39.69	2.27	6.37	53.0	10.7

The following measurements marked *client certs* used SSL client certificates in addition to server certificates.

Table 83: SSL full handshake, 1024-bit key, client certs, CWS direct connection

Throughput	CICS CPU%	VTAM CPU%	TCP/IP CPU%	Total CPU%	CPU ms/request
12.47	243.98	1.23	3.48	255.6	205.0
15.04	306.71	1.53	4.29	318.8	212.0
18.93	389.35	1.83	5.19	403.4	213.1
23.65	481.63	2.34	6.61	497.7	210.4
23.99	483.09	2.34	6.93	500.0	208.4

Table 84: SSL full handshake with crypto, 1024-bit key, client certs, CWS direct connection

Throughput	CICS CPU%	VTAM CPU%	TCP/IP CPU%	Total CPU%	CPU ms/request
13.08	19.77	4.37	2.40	34.8	26.6
16.26	23.81	4.34	2.83	39.6	24.4
21.21	30.58	4.26	3.59	47.2	22.2
30.51	42.95	4.49	4.73	61.6	20.1

48.27	68.14	4.55	6.27	89.0	18.4

B.4.2 SSL data transmission with a CWS direction connection

The following data, presented in Table 88 on page 191 through Table 99 on page 194 , shows the results for the SSL data transmission tests with a CWS direct connection. All the data transmission tests used persistent HTTP connections. The results shown in Table 97 on page 194 through Table 99 on page 194 marked with *crypto* used the S/390 Cryptographic Coprocessor; the non-SSL figures in Table 85 through Table 87 are given for comparison.

Table 85: Non-SSL 1 byte transmission, CWS direct connection

Throughput	CICS CPU%	VTAM CPU%	TCP/IP CPU%	Total CPU%	CPU ms/request
13.57	4.62	0.28	0.70	14.6	10.8
16.89	5.66	0.28	0.83	15.6	9.2
22.24	7.22	0.40	1.07	17.3	7.8
32.65	10.06	0.52	1.29	20.0	6.1
61.00	17.37	0.73	1.82	27.45	4.5

Table 86: Non-SSL 8KB transmission, CWS direct connection

Throughput	CICS CPU%	VTAM CPU%	TCP/IP CPU%	Total CPU%	CPU ms/request
13.58	5.34	0.26	1.10	15.75	11.6
16.95	6.58	0.39	1.34	17.05	10.1
22.24	8.39	0.50	1.70	19.0	8.5
32.54	11.66	0.60	2.13	22.7	7.0
60.86	20.21	1.03	3.45	32.45	5.3

Table 87: Non-SSL 16KB transmission, CWS direct connection

Throughput	CICS CPU%	VTAM CPU%	TCP/IP CPU%	Total CPU%	CPU ms/request
13.62	6.07	0.38	1.48	16.6	12.2
16.86	7.39	0.37	1.58	18.0	10.7
22.25	9.63	0.60	2.18	20.6	9.3
32.48	13.23	0.82	2.84	24.9	7.7
60.50	23.28	1.22	4.45	36.5	6.0

Table 88: SSL 1 byte transmission, RC4-MD5(40 bit), CWS direct connection

Throughput	CICS CPU%	VTAM CPU%	TCP/IP CPU%	Total CPU%	CPU ms/request
13.60	6.28	0.28	0.84	16.1	11.8
16.85	7.32	0.27	0.81	16.8	10.0
22.27	9.19	0.26	1.05	18.9	8.5
32.62	12.92	0.38	1.27	22.6	6.9

60.93	22.35	0.71	1.90	32.5	5.3

Table 89: SSL 8KB transmission, RC4-MD5(40 bit), CWS direct connection

Throughput	CICS CPU%	VTAM CPU%	TCP/IP CPU%	Total CPU%	CPU ms/request
13.59	9.10	0.26	1.19	18.9	13.9
16.90	10.69	0.39	1.29	20.4	12.0
22.21	13.80	0.50	1.62	23.8	10.7
32.51	18.99	0.60	2.16	29.5	9.1
60.57	33.62	1.14	3.42	45.3	7.5

Table 90: SSL 16KB transmission, RC4 -MD5(40 bit), CWS direct connection

Throughput	CICS CPU%	VTAM CPU%	TCP/IP CPU%	Total CPU%	CPU ms/request
13.59	11.75	0.38	1.40	21.5	15.8
16.84	14.08	0.50	1.62	23.8	14.1
22.17	18.15	0.60	2.04	28.3	12.8
32.20	24.92	0.82	2.79	35.8	11.1
60.13	44.00	1.22	4.33	56.7	9.4

Table 91: SSL1 byte transmission, RC4-MD5(128 bit), CWS direct connection

Throughput	CICS CPU%	VTAM CPU%	TCP/IP CPU%	Total CPU%	CPU ms/request
13.67	6.19	0.41	0.83	16.5	12.0
16.86	7.28	0.40	0.81	17.4	10.3
22.23	9.33	0.39	1.05	19.5	8.8
32.56	13.06	0.51	1.27	23.2	7.1
60.96	22.58	0.71	1.90	33.2	5.4

Table 92: SSL 8KB transmission, RC4-MD5(128 bit), CWS direct connection

Throughput	CICS CPU%	VTAM CPU%	TCP/IP CPU%	Total CPU%	CPU ms/request
13.56	9.10	0.26	1.19	19.0	14.0
16.85	10.70	0.39	1.29	20.5	12.2
22.24	13.64	0.50	1.63	23.7	10.7
32.52	19.14	0.60	2.17	29.5	9.1
60.62	33.61	1.03	3.42	45.4	7.5

Table 93: SSL 16KB transmission, RC4 -MD5(128 bit), CWS direct connection

Throughput	CICS CPU%	VTAM CPU%	TCP/IP CPU%	Total CPU%	CPU ms/request
13.58	11.69	0.38	1.40	21.9	16.1
16.80	13.93	0.37	1.62	23.8	14.1
22.21	17.85	0.60	2.05	28.4	12.8
32.28	24.94	0.82	2.68	35.9	11.1

60.17	43.92	1.22	4,33	57.5	9.5

Table 94: SSL 1 byte transmission, triple DES, CWS direct connection

Throughput	CICS CPU%	VTAM CPU%	TCP/IP CPU%	Total CPU%	CPU ms/request
13.59	7.60	0.27	0.81	17.1	12.6
16.91	8.88	0.26	0.79	18.2	10.8
22.19	11.26	0.26	1.02	20.6	9.3
32.48	15.72	0.37	1.24	25.0	7.7
61.03	27.51	0.70	1.86	37.2	6.1

Table 95: SSL 8KB transmission, triple DES, CWS direct connection

Throughput	CICS CPU%	VTAM CPU%	TCP/IP CPU%	Total CPU%	CPU ms/request
13.49	25.02	0.35	1.04	33.2	24.6
16.72	30.17	0.34	1.13	38.3	22.9
22.01	38.65	0.44	1.44	47.2	21.4
32.13	54.88	0.65	1.94	64.1	19.9
59.41	100.44	1.06	3.27	111.0	18.7

Table 96: SSL 16KB transmission, triple DES, CWS direct connection

Throughput	CICS CPU%	VTAM CPU%	TCP/IP CPU%	Total CPU%	CPU ms/request
13.47	41.57	0.33	1.21	50.4	37.4
16.62	50.38	0.43	1.41	59.3	35.6
21.87	65.03	0.53	1.71	74.2	33.9
31.82	93.89	0.74	2.53	104.0	32.7
56.90	164.92	1.03	3.62	176.4	31.0

Table 97: SSL 1 byte transmission, triple DES with crypto, CWS direct connection

Throughput	CICS CPU%	VTAM CPU%	TCP/IP CPU%	Total CPU%	CPU ms/request
13.63	7.85	0.41	0.81	17.6	12.9
16.92	9.22	0.4	0.94	18.7	11.1
22.17	11.45	0.51	1.03	21.1	9.5
32.51	16.14	0.62	1.24	25.7	7.9
60.67	28.01	0.82	1.87	38.1	6.3

Table 98: SSL 8KB transmission, triple DES with crypto, CWS direct connection

Throughput	CICS CPU%	VTAM CPU%	TCP/IP CPU%	Total CPU%	CPU ms/request
13.54	15.78	0.49	1.11	25.2	18.6
16.82	18.85	0.48	1.32	28.1	16.7
22.07	23.94	0.59	1.64	33.5	15.2
32.24	33.62	0.79	2.15	43.7	13.6

59.26	59.35	1.20	3.48	71.0	12.0

Table 99: SSL 16KB transmission, triple DES with crypto, CWS direct connection

Throughput	CICS CPU%	VTAM CPU%	TCP/IP CPU%	Total CPU%	CPU ms/request
13.48	22.96	0.35	1.29	31.75	23.6
16.69	27.59	0.46	1.49	36.4	21.8
21.96	35.36	0.56	1.90	44.7	20.3
31.96	50.02	0.87	2.51	60.1	18.8
58.04	90.16	1.17	3.95	102.0	17.6

B.4.3 SSL handshakes with the CICS WebServer Plugin

The following data, presented in Table 100 on page 195 through Table 103 on page 196 , shows the results for the SSL handshake tests with CWS and the CICS WebServer Plugin. All the handshake tests used non-persistent HTTP connections and sent 1 byte of data from the CICS application.

Table 100: SSL full handshake, 1024 bit key, WebServer Plugin

Throughput	CICS CPU%	Web server CPU%	VTAM CPU%	TCP/IP CPU%	Total CPU%	CPU ms/request
12.62	2.51	154.18	0.98	2.73	171.0	135.5
15.28	3.09	195.39	1.21	3.31	214.3	140.2
19.00	4.14	259.73	1.57	4.25	281.2	148.0
21.5	5.01	320.09	1.82	5.12	344.1	160.0
21.44	5.00	319.52	1.93	5.12	343.9	160.4

Table 101: SSL full handshake, 512-bit key, WebServer Plugin

Throughput	CICS CPU%	Web server CPU%	VTAM CPU%	TCP/IP CPU%	Total CPU%	CPU ms/request
12.62	2.62	59.81	1.14	2.84	76.8	60.8
15.46	3.28	81.16	1.29	3.40	101.7	65.8
19.90	4.48	117.33	1.70	4.36	140.0	70.4
26.5	6.45	183.33	2.11	5.95	210.6	79.5
27.78	7.16	208.96	2.51	6.78	239.9	86.3

Table 102: SSL null handshake, 1024-bit key, WebServer Plugin

Throughput	CICS CPU%	Web server CPU%	VTAM CPU%	TCP/IP CPU%	Total CPU%	CPU ms/request
12.57	2.63	18.29	0.96	2.51	35.5	28.2
15.35	3.16	21.92	1.05	3.05	40.6	26.4
19.63	4.06	28.32	1.39	3.71	48.8	24.9
29.96	5.50	39.42	1.72	4.81	62.8	21.0

42.87	8.87	65.99	2.62	7.28	96.3	22.5

Table 103: SSL null handshake, 512-bit key, WebServer Plugin

Throughput	CICS CPU%	Web server CPU%	VTAM CPU%	TCP/IP CPU%	Total CPU%	CPU ms/request
12.68	2.62	18.36	0.95	2.62	35.7	28.2
15.42	3.18	22.35	1.18	3.06	41.1	26.7
19.67	4.06	28.30	1.39	3.71	49.0	24.9
26.88	5.40	39.55	1.72	4.71	63.0	23.4
42.78	8.82	66.47	2.52	7.22	97.0	22.7

B.4.4 SSL data transmission with the CICS WebServer Plugin

The following data, presented in Table 104 through Table 106, shows the results for the SSL data transmission tests with CWS and the CICS WebServer Plugin. All the data transmission tests used persistent HTTP connections.

Table 104: SSL 1 byte transmission, RC4-MD5(40 bit), WebServer Plugin

Throughput	CICS CPU%	Web server CPU%	VTAM CPU%	TCP/IP CPU%	Total CPU%	CPU ms/request
13.62	3.28	10.98	4.16	0.76	27.0	19.8
16.86	3.97	13.41	4.10	0.87	30.1	17.9
22.21	5.19	17.37	4.10	1.09	35.4	15.9
32.55	7.14	23.88	4.10	7.14	43.9	13.5
60.33	12.62	40.45	4.39	2.14	66.7	11.1

Table 105: SSL 8KB transmission, RC4-MD5(40 bit), CWS direct connection

Throughput	CICS CPU%	Web server CPU%	VTAM CPU%	TCP/IP CPU%	Total CPU%	CPU ms/request
13.61	3.81	14.97	4.05	0.86	31.3	23.0
16.85	4.56	18.26	4.08	1.08	35.6	21.1
22.15	5.88	23.15	4.23	1.29	42.0	19.0
32.23	8.23	32.00	4.34	1.60	53.3	16.5
59.34	14.31	55.68	4.88	2.55	84.4	14.2

Table 106: SSL 16KB transmission, RC4 -MD5(40 bit), CWS direct connection

Throughput	CICS CPU%	Web server	VTAM CPU%	TCP/IP CPU%	Total CPU%	CPU ms/request
13.52	4.06	18.40	4.06	1.08	35.25	26.1
16.81	5.05	22.45	4.11	1.18	40.2	23.9
22.08	6.55	28.83	4.25	1.49	48.35	21.9

32.27	9.19	40.34	3.59	1.90	61.85	19.2
59.58	16.24	71.06	2.62	3.05	99.4	16.7

B.5 CICS Transaction Gateway

In this section we present the results of our tests using the OS/390 CICS Transaction Gateway (CTG), first using a Java applets and then using Java servlets. For further discussion of this data refer to Chapter 7 , "The OS/390 CTG" on page 103 .

B.5.1 CTG Java applets.

The following data, presented in Table 107 on page 198 through Table 114 on page 201 , shows the results for CTG Java applets.Throughput is defined as ECI requests per second. For the tests with the TCP/IP protocol 500 clients were used and for the tests with the HTTP protocol 100 clients were used.

The measurements in Table 107 were performed using a TCP/IP connection from the applet to the CTG Java gateway application that was not re-used.

Table 107: Applets, TCP/IP, no connection re-use, COMMAREA 100 bytes

Throughput	CICS CPU%	CTG CPU%	VTAM CPU%	TCP/IP CPU%	Total CPU%	CPU ms/request
52.89	7.9	89.0	2.8	7.9	123.6	23.4
61.92	9.4	104.5	3.3	9.3	144.3	23.30
69.51	10.7	122.0	3.6	10.3	166.8	24.0
76.55	11.8	136.6	4.0	11.3	185.0	24.2
90.93	14.3	167.3	4.9	14.0	225.7	24.8

The following measurements in Table 108 on page 199 through Table 113 on page 200 re-used the TCP/IP connection across ECI calls. The figures are for a range of COMMAREA sizes from 100 bytes to 16,000 bytes.

Table 108: Applets, TCP/IP connection, COMMAREA 100 bytes

Throughput	CICS CPU%	CTG CPU%	VTAM CPU%	TCP/IP CPU%	Total CPU%	CPU ms/request
32.53	4.5	26.6	0.5	1.3	41.9	12.9
49.28	6.8	39.4	0.8	1.8	58.3	11.8
65.79	9.1	54.8	1.0	2.4	77.6	11.8
98.55	13.7	82.3	1.4	3.5	112.9	11.5
133.0	19.9	135.0	1.8	4.9	185.7	14.0

Table 109: Applets, TCP/IP connection, COMMAREA 1000 bytes

Throughput	CICS CPU%	CTG CPU%	VTAM CPU%	TCP/IP CPU%	Total CPU%	CPU ms/request

30.37	4.3	28.6	0.5	1.2	44.2	14.5
46.02	6.5	44.1	0.7	1.8	63.4	13.8
61.47	8.8	58.6	0.9	2.4	81.9	13.3
91.75	13.3	89.9	1.3	3.5	122.4	13.3
117.7	18.5	143.7	1.7	4.7	194.0	16.5

Table 110: Applets, TCP/IP connection, COMMAREA 2000 bytes

Throughput	CICS CPU%	CTG CPU%	VTAM CPU%	TCP/IP CPU%	Total CPU%	CPU ms/request
28.81	4.2	35.9	0.5	1.2	51.9	18.0
43.45	6.4	53.0	0.7	1.8	77.4	17.8
57.86	8.5	70.1	0.9	2.3	94.3	16.3
78.91	12.5	113.0	1.2	3.2	150.9	19.1
102.60	16.5	151.2	1.4	4.2	200.7	19.6

Table 111: Applets, TCP/IP connection, COMMAREA 4000 bytes

Throughput	CICS CPU%	CTG CPU%	VTAM CPU%	TCP/IP CPU%	Total CPU%	CPU ms/request
29.41	4.6	38.4	0.5	1.3	55.2	18.8
44.10	6.9	56.2	0.8	1.9	77.4	17.6
58.59	9.3	76.3	1.0	2.8	124.0	21.2
79.86	13.5	120.1	1.2	3.5	160.3	20.1
99.42	17.1	152.4	1.4	4.5	202.6	20.4

Table 112: Applets, TCP/IP connection, COMMAREA 8000 bytes

Throughput	CICS CPU%	CTG CPU%	VTAM CPU%	TCP/IP CPU%	Total CPU%	CPU ms/request
32.90	5.6	49.9	0.6	1.6	68.7	20.9
48.55	8.4	76.4	0.9	2.3	101.6	20.9
62.36	11.2	103.3	1.1	3.0	137.4	22.0
84.03	15.6	142.9	1.4	4.3	189.7	22.6
91.46	17.1	154.0	1.5	4.8	204.7	22.4

Table 113: Applets, TCP/IP connection, COMMAREA 16000 bytes

Throughput	CICS CPU%	CTG CPU%	VTAM CPU%	TCP/IP CPU%	Total CPU%	CPU ms/request
25.90	5.1	53.5	0.8	2.4	73.5	28.4
38.31	7.6	79.8	1.2	3.4	106.4	27.8
49.74	10.3	102.3	1.5	4.7	138.0	27.7
66.52	14.3	132.0	2.1	7.1	180.2	27.1

81.31	17.8	154.6	2.6	8.7	212.1	26.1

In the following measurements in Table 114 the work from the 500 clients was balanced across four CTG Java gateway application address spaces using TCP/IP port sharing. The TCP/IP connection was re-used across ECI calls.The COMMAREA size was 100 bytes. The CTG CPU usage is the sum of all four CTG address spaces.

Table 114: Applets, TCP/IP connection, multiple CTG address spaces

Throughput	CICS CPU%	All CTG CPU%	VTAM CPU%	TCP/IP CPU%	Total CPU%	CPU ms/request
98.8	13.4	54.8	1.3	3.3	84.1	8.5
123.3	16.6	69.3	1.6	4.0	103.6	8.4
164.1	22.0	93.2	2.1	5.3	135.8	8.3
242.2	32.4	146.5	2.8	7.5	206.5	8.5
454.1	60.7	296.3	4.3	13.8	241.5	5.3

The following measurements in Table 115 on page 202 through Table 120 on page 203 were performed using a HTTP connection from the applet to the CTG Java gateway application that was not re-used, a range of COMMAREA sizes from 100 bytes to 16,000 bytes was used.

Table 115: Applets, HTTP connection, COMMAREA 100 bytes

Throughput	CICS CPU%	CTG CPU%	VTAM CPU%	TCP/IP CPU%	Total CPU%	CPU ms/request
13.59	2.0	72.1	0.6	1.5	90.4	66.5
19.09	2.9	103.6	0.7	1.8	126.0	66.0
23.78	3.6	134.8	1.0	2.3	163.7	68.8
27.15	4.2	146.4	1.1	2.8	175.2	64.5
31.79	4.9	181.3	1.4	3.4	216.9	68.2

Table 116: Applets, HTTP connection, COMMAREA 1000 bytes

Throughput	CICS CPU%	CTG CPU%	VTAM CPU%	TCP/IP CPU%	Total CPU%	CPU ms/request
13.53	2.0	71.3	0.6	1.4	89.0	65.9
19.02	2.9	104.4	0.7	1.8	127.3	66.9
23.82	3.7	134.8	1.0	2.3	162.9	68.4
27.03	4.2	149.1	1.2	2.8	178.3	66.0
31.73	4.9	177.7	1.4	3.3	213.0	67.1

Table 117: Applets, HTTP connection, COMMAREA 2000 bytes

Throughput	CICS CPU%	CTG CPU%	VTAM CPU%	TCP/IP CPU%	Total CPU%	CPU ms/request

13.64	2.0	71.80	0.8	1.9	89.9	65.9
19.04	2.8	109.7	0.9	2.1	135.2	71.0
23.85	3.7	140.9	1.2	2.9	171.8	72.0
27.14	4.1	152.5	1.4	3.4	183.9	67.7
29.98	4.7	170.7	1.6	4.0	205.9	68.7

Table 118: Applets, HTTP connection, COMMAREA 4000 bytes

Throughput	CICS CPU%	CTG CPU%	VTAM CPU%	TCP/IP CPU%	Total CPU%	CPU ms/request
13.64	2.1	66.9	0.8	2.5	84.9	62.2
18.96	2.9	106.8	0.9	2.3	132.1	69.7
23.85	3.5	132.5	1.0	2.2	160.4	67.3
27.22	4.1	148.6	1.1	2.8	178.2	65.5
31.68	4.8	178.3	1.4	3.3	213.8	67.5

Table 119: Applets, HTTP connection, COMMAREA 8000 bytes

Throughput	CICS CPU%	CTG CPU%	VTAM CPU%	TCP/IP CPU%	Total CPU%	CPU ms/request
13.56	2.3	68.8	1.0	3.0	88.2	65.0
19.02	2.9	102.5	0.8	1.7	125.9	66.2
23.78	3.7	132.0	1.0	2.3	160.5	67.5
27.29	4.2	148.4	1.2	2.8	178.0	65.2
31.48	4.9	180.9	1.4	3.3	217.1	69.0

Table 120: Applets, HTTP connection, COMMAREA 16000 bytes

Throughput	CICS CPU%	CTG CPU%	VTAM CPU%	TCP/IP CPU%	Total CPU%	CPU ms/request
13.30	2.6	70.8	1.3	4.0	90.5	68.0
18.95	3.9	119.9	1.8	5.2	150.8	79.6
23.84	3.6	130.1	1.0	2.2	155.8	65.33
27.25	4.1	146	1.1	2.7	173.2	63.5
31.85	4.8	176.8	1.4	3.2	209.2	65.7

B.5.2 CTG Java servlets

The following data, presented in Table 121 on page 204 through Table 124 on page 205 , shows the results for CTG Java servlets.Throughput is defined as ECI requests per second (or Web requests per second if no ECI call). For all the tests 100 clients were used. For further details on the test scenario refer to Chapter 7 , "The OS/390 CTG" on page 103 .

The following CTG servlet measurements in Table 121 on page 204 through Table 124 on page 205

were conducted to measure the effect of persistent HTTP connection and the cost of the ECI call within the servlet. The COMMAREA size was 39 bytes for all servlets that used the ECI.

Table 121: Servlets, persistent HTTP connection, ECI

Throughput	CICS CPU%	Web server CPU%	VTAM CPU%	TCP/IP CPU%	Total CPU%	CPU ms/request
19.69	2.8	46.2	0.4	0.9	60.4	30.7
24.5	3.5	57.6	0.5	0.4	73.2	29.9
32.34	4.7	77.8	0.6	1.2	97.4	30.1
47.37	7.3	124.6	0.8	1.6	158.7	33.5
60.82	9.7	166.4	1.0	2.2	211.6	34.8

Table 122: Servlets, non-persistent HTTP connection, ECI

Throughput	CICS CPU%	Web server CPU%	VTAM CPU%	TCP/IP CPU%	Total CPU%	CPU ms/request
18.89	2.7	47.7	0.9	1.8	63.7	33.7
21.10	3.3	58.5	1.0	2.2	76.5	36.3
27.15	4.2	77.5	1.2	2.7	101.7	37.5
38.29	7.3	113	1.6	4.0	150.6	39.3
47.89	9.7	141	2.0	4.8	186.9	39.0

Table 123: Servlets, persistent HTTP connection, no ECI

Throughput	CICS CPU%	Web server CPU%	VTAM CPU%	TCP/IP CPU%	Total CPU%	CPU ms/request
19.75	0.0	36.4	0.4	0.8	47.3	24.0
24.49	0.0	45.2	0.5	0.9	56.9	23.2
32.42	0.0	59.	0.5	1.1	72.0	22.2
47.37	0.0	96.7	0.7	1.5	118.7	23.1
62.16	0.0	1313	0.9	1.9	161.9	26.0

Table 124: Servlets, non-persistent HTTP connection, no ECI

Throughput	CICS CPU%	Web server CPU%	VTAM CPU%	TCP/IP CPU%	Total CPU%	CPU ms/request
18.89	0.0	36.6	0.8	1.7	50.5	26.7
21.71	0.0	42.8	0.9	2.0	57.9	26.7
27.3	0.0	55.8	1.1	.2.5	72.7	26.6
38.48	0.0	75.0	1.5	3.5	94.9	24.7
48.80	0.0	106.8	1.8	4.4	143.5	29.4

Appendix C: Using the additional material

Overview

This redbook also contains additional material that can be downloaded from the Internet as described below.

C.1 Locating the additional material on the Internet

The Web material associated with this redbook is also available in softcopy on the Internet from the IBM Redbooks Web server. Point your Web browser to:

ftp://www.redbooks.ibm.com/redbooks/SG24-5748

Alternatively, you can go to the IBM Redbooks Web site at:

http://www.redbooks.ibm.com/

Select the **Additional materials** and open the directory that corresponds with the redbook form number.

C.2 Using the Web material

The additional Web material that accompanies this redbook includes the following:

File name	*Description*
TraderCicsWebSamples.zip	Zipped code samples for the CICS Trader application, and Web-enablement using CWS and the CTG.

C.2.1 System requirements for downloading the Web material

The following system configuration is recommended for downloading the additional Web material.

Hard disk space :	1 MB minimum
Operating System :	Windows NT or 95
Processor :	Intel 286 or higher
Memory :	16 MB

C.2.2 How to use the Web material

Create a subdirectory (folder) on your workstation, download the contents of the Web material into this folder, then unzip the file.

Appendix D: Special notices

Overview

This publication is intended to help technical professionals to understand and plan for the performance

impact of Web-enabling legacy CICS applications. The information in this publication is not intended as the specification of any programming interfaces that are provided by CICS Transaction Server v1.3 or OS/390 WebSphere Application Server. See the PUBLICATIONS section of the IBM Programming Announcement for CICS Transaction Server, and OS/390 WebSphere Application Server for more information about what publications are considered to be product documentation.

References in this publication to IBM products, programs or services do not imply that IBM intends to make these available in all countries in which IBM operates. Any reference to an IBM product, program, or service is not intended to state or imply that only IBM's product, program, or service may be used. Any functionally equivalent program that does not infringe any of IBM's intellectual property rights may be used instead of the IBM product, program or service.

Information in this book was developed in conjunction with use of the equipment specified, and is limited in application to those specific hardware and software products and levels.

IBM may have patents or pending patent applications covering subject matter in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to the IBM Director of Licensing, IBM Corporation, North Castle Drive, Armonk, NY 10504-1785.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact IBM Corporation, Dept. 600A, Mail Drop 1329, Somers, NY 10589 USA.

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

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.

Any pointers in this publication to external Web sites are provided for convenience only and do not in any manner serve as an endorsement of these Web sites.

Any performance data contained in this document was determined in a controlled environment, and therefore, the results that may be obtained in other operating environments may vary significantly. Users of this document should verify the applicable data for their specific environment.

This document contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples contain the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

Reference to PTF numbers that have not been released through the normal distribution process does not imply general availability. The purpose of including these reference numbers is to alert IBM customers to specific information relative to the implementation of the PTF when it becomes available to each

customer according to the normal IBM PTF distribution process.

The following terms are trademarks of the International Business Machines Corporation in the United States and/or other countries:

AIX	AS/400
AT	CICS
CICS/ESA	CICS/MVS
CICS/VSE	CICSPlex
CT	DB2
DFSMS	eNetwork
IBM ¯	IMS
Language Environment	MQ
Netfinity	OS/390
Parallel Sysplex	RACF
RAMAC	RMF
RS/6000	S/390
SecureWay	SP
SP2	System/390
VisualAge	VTAM
WebSphere	XT
400	

The following terms are trademarks of other companies:

C-bus is a trademark of Corollary, Inc. in the United States and/or other countries.

Java and all Java-based trademarks and logos are trademarks or registered trademarks of Sun Microsystems, Inc. in the United States and/or other countries.

Microsoft, Windows, Windows NT, and the Windows logo are trademarks of Microsoft Corporation in the United States and/or other countries.

PC Direct is a trademark of Ziff Communications Company in the United States and/or other countries and is used by IBM Corporation under license.

ActionMedia, LANDesk, MMX, Pentium and ProShare are trademarks of Intel Corporation in the United States and/or other countries.

UNIX is a registered trademark in the United States and other countries licensed exclusively through The Open Group.

SET and the SET logo are trademarks owned by SET Secure Electronic Transaction LLC.

Other company, product, and service names may be trademarks or service marks of others.

Appendix E: Related publications

Overview

The publications listed in this section are considered particularly suitable for a more detailed discussion of the topics covered in this redbook.

E.1 International Technical Support Organization publications

For information on ordering these ITSO publications see " How to get ITSO redbooks " on page 217 .

- *Revealed! Architecting Web Access to CICS* , SG24-5466
- *OS/390 Version 2 Release 4 Performance Figures for CICS Web-Enabled Applications* , SG24-5612
- *CICS Transaction Server for OS/390 Version 1 Release3: Web Support and 3270 Bridge* , SG24-5480
- *Revealed! CICS Transaction Gateway with More CICS Clients Unmasked* , SG24-5277
- *TCP/IP Implementation Guide* , SG24-5227
- *IBM SecureWay Host On-Demand: Enterprise Communications Era Network Computing* , SG24-2149
- *Java Application Development for CICS: Base Services and CORBA Client Support* , SG24-5275
- *TCP/IP Tutorial and Technical Overview* , GG24-3376
- *CICS/ESA and TCP/IP for MVS Sockets Interface* , GG24-4026
- *Enterprise-Wide Security Architecture and Solutions* , SG24-4579
- *VisualAge for Java Enterprise Version 2: Data Access Beans - Servlets - CICS Connector* , SG24-5265
- *OS/390 MVS Parallel Sysplex Capacity Planning* , SG24-4680
- *OS/390 e-business Infrastructure: IBM WebSphere Application Server 1.1 - Customizing and Usage* , SG24-5604

E.2 Redbooks on CD-ROMs

Redbooks are also available on the following CD-ROMs. Click the CD-ROMs button at http://www.redbooks.ibm.com/ for information about all the CD-ROMs offered, updates and formats.

CD-ROM Title	Collection Kit Number
System/390 Redbooks Collection	SK2T-2177
Networking and Systems Management Redbooks Collection	SK2T-6022
Transaction Processing and Data Management Redbooks Collection	SK2T-8038
Lotus Redbooks Collection	SK2T-8039
Tivoli Redbooks Collection	SK2T-8044
AS/400 Redbooks Collection	SK2T-2849
Netfinity Hardware and Software Redbooks Collection	SK2T-8046
RS/6000 Redbooks Collection (BkMgr)	SK2T-8040
RS/6000 Redbooks Collection (PDF Format)	SK2T-8043
Application Development Redbooks Collection	SK2T-8037
IBM Enterprise Storage and Systems Management Solutions	SK3T-3694

E.3 Other publications

These publications are also relevant as further information sources:

- *CICS Performance Guide* , SC33-1699
- *CICS Internet Guide* , SC34-5445
- *CICS Internet and External Interfaces Guide* , SC33-1944
- *CICS Web Interface Guide* , SC33-1892
- *IBM HTTP Server for OS/390 Release 7, Planning, Installing, and Using, Version 5.1* , SC31-8690
- *WebSphere Application Server for OS/390, Application Server Planning, Installing,and Using, Version 1.1* , GC34-4757
- *IBM TCP/IP Performance Tuning Guide* , SC31-7188
- *OS/390 eNetwork Communications Server, IP Application Programming Interface Guide* , SC31-8516
- *CICS Transaction Gateway Administration Guide* , SC34-5448
- *IBM TCP/IP Performance Tuning Guide* , SC31-7188
- *OS/390 eNetworks Communications Server: IP Planning and Migration Guide* , SC31-8512
- *Communications Server: IP Configuration Manual* , SC31-8513
- *Applied Cryptography, ISBN 0-471-11709-9* , SR28-5808

How to get ITSO redbooks

Overview

This section explains how both customers and IBM employees can find out about ITSO redbooks, redpieces, and CD-ROMs. A form for ordering books and CD-ROMs by fax or e-mail is also provided.

- **Redbooks Web Site** http://www.redbooks.ibm.com/

  Search for, view, download, or order hardcopy/CD-ROM redbooks from the redbooks Web site. Also read redpieces and download additional materials (code samples or diskette/CD-ROM images) from this redbooks site.

  Redpieces are redbooks in progress; not all redbooks become redpieces and sometimes just a few chapters will be published this way. The intent is to get the information out much quicker than the formal publishing process allows.

- **E-mail Orders**

  Send orders by e-mail including information from the redbooks fax order form to:

	e-mail address
In United States	< usib6fpl@ibmmail.com >
Outside North America	Contact information is in the "How to Order" section at this site: http://www.elink.ibmlink.ibm.com/pbl/pbl

- **Telephone Orders**

United States (toll free)	1-800-879-2755
Canada (toll free)	1-800-IBM-4YOU
Outside North America	Country coordinator phone number is in the "How to Order" section at this site: http://www.elink.ibmlink.ibm.com/pbl/pbl

- **Fax Orders**

United States (toll free)	1-800-445-9269
Canada	1-403-267-4455
Outside North America	Fax phone number is in the "How to Order" section at this site: http://www.elink.ibmlink.ibm.com/pbl/pbl

This information was current at the time of publication, but is continually subject to change. The latest information may be found at the redbooks Web site.

IBM Intranet for Employees

IBM employees may register for information on workshops, residencies, and redbooks by accessing the IBM Intranet Web site at http://w3.itso.ibm.com/ and clicking the ITSO Mailing List button. Look in the Materials repository for workshops, presentations, papers, and Web pages developed and written by the ITSO technical professionals; click the Additional Materials button. Employees may access MyNews at http://w3.ibm.com/ for redbook, residency, and workshop announcements.

Glossary

An excellent glossary of Internet and Internet related terms is available at:

http://www.matisse.net/files/glossary.html

Other terms not covered in the above-mentioned Web document or clarified in this document are listed below.

abend.
: Abnormal end of task.

API.
: Application programming interface. A set of calling conventions defining how a service is invoked through a software package.

APPC.
: Advanced program-to-program communication. An implementation of the SNA LU 6.2 protocol that allows interconnected systems to communicate and share the processing of programs.

asynchronous.
: Without regular time relationship; unexpected or unpredictable with respect to the execution of program instructions. See *synchronous* .

browser.
: An application that displays World Wide Web documents, usually referred to as a Web browser.

CEC
: (also known as **CPC**). Central Electronic Complex (or Central Processing Complex) is the physical machine that contains main storage(memory), central processing units and connections to devices.

CPU.
: Central Processing Unit (also known as an engine or processor) is the part of the CEC that executes the program instructions. There may be one or many CPUs in a CEC. Each CPU in the CEC may access the main storage (memory) in that CEC. If there are multiple CPUs in a CEC, then multiprocessing (or simultaneous execution of two threads of control) is possible.

CERN.
: The Conseil Europeen pour la Recherche Nucleaire (European Particle Physics Laboratory), which developed hypertext technologies.

distributed program link (DPL).
: Enables an application program executing in one CICS system to link (pass control) to a program in a different CICS system. The linked-to program executes and returns a result to the linking program. This process is equivalent to remote procedure calls (RPCs). You can write applications that issue RPCs that can be received by members of the CICS family.

distributed transaction processing (DTP).
: Enables a transaction running in one CICS system to communicate synchronously with transactions running in other systems. The transactions are designed and coded specifically to communicate with each other. This method is typically used by banks, for example in "just-in-time" stock replacement.

Customer Information Control System (CICS).
: A distributed on-line transaction processing system designed to support a network of many terminals. The CICS family of products is available for a variety of platforms ranging from a single workstation to the largest mainframe.

client.
: As in client/server computing, the application that makes requests to the server and, often, handles the interaction necessary with the user.

client/server computing.
: A form of distributed processing, in which the task required to be processed is accomplished by a client portion that requests services and a server portion that fulfills those requests. The client and server remain transparent to each other in terms of location and platform. See *client* and *server* .

commit.
: An action that an application takes to make permanent the changes it has made to recoverable resources during a logical unit of work.

Common Gateway Interface (CGI).
: The defined standard for the communications between HTTP servers and external executable programs.

conversational.
: A communication model where two distributed applications exchange information by way of a conversation; typically one application starts (or allocates) the conversation, sends some data, and allows the other application to send some data. Both applications continue in turn until one decides to finish (or deallocate). The conversational model is a synchronous form of communication.

Coupling facility.

: Is a special logical partition that provides high-speed caching, list processing, and locking functions between systems in a Parallel Sysplex.

database.
: (1) A collection of interrelated data stored together with controlled redundancy according to a scheme to serve one or more applications. (2) All data files stored in the system. (3) A set of data stored together and managed by a database management system.

Distributed Computing Environment (DCE).
: Adopted by the computer industry as a de facto standard for distributed computing. DCE allows computers from a variety of vendors to communicate transparently and share resources such as computing power, files, printers, and other objects in the network.

delimiter.
: A character or sequence of characters used as a separator in text or data files.

Distributed processing.
: An application or systems model in which function and data can be distributed across multiple computing resources connected on a LAN or WAN. See *client/server computing* .

External Call Interface (ECI).
: An application programming interface (API) that enables a non-CICS client application to call a CICS program as a subroutine. The client application communicates with the server CICS program using a data area called a COMMAREA.

External Presentation Interface (EPI).

An application programming interface (API) that allows a non-CICS application program to appear to the CICS system as one or more standard 3270 terminals. The non-CICS application can start CICS transactions and send and receive standard 3270 data streams to those transactions.

environment.

The collective hardware and software configuration of a system.

File Transfer Protocol (FTP).

A protocol that defines how to transfer files from one computer to another.

forms.

Parts of HTML documents that allow users to enter data.

function shipping.

A CICS Inter Systems Communication protocol that enables an application program running in one CICS system to access resources owned by another CICS system. In the resource-owning system, a mirror transaction is initiated to perform the necessary operation; for example, to access CICS files or temporary storage, and to reply to the requester.

gateway.

Software that transfers data between normally incompatible applications or between networks.

Graphic Interchange Format (GIF).

256-color graphic format.

Graphical user interface (GUI).

A style of user interface that replaces the character-based screen with an all-points-addressable, high-resolution graphics screen. Windows display multiple applications at the same time and allow user input by means of a keyboard or a pointing device such as mouse, pen, or trackball.

host.

(1) In a computer network, a computer providing services such as computation, database access, and network control functions. (2) In a multiple computer installation, the primary or controlling computer.

hypertext.

Text that activates connection to other documents when selected.

Hypertext Markup Language (HTML).

Standard language used to create hypertext documents.

Hypertext Transmission Protocol (HTTP).

Standard WWW client/server communications protocol.

Internet.

A collection of networks.

LU type 6.2 (LU 6.2).

A type of logical unit used for CICS intersystem communication (ISC). LU 6.2 architecture supports CICS host-to-system-level products and CICS host-to-device-level products. APPC is the protocol boundary of the LU 6.2 architecture.

Logical unit of work (LUW).

An update that durably transforms a resource from one consistent state to another consistent state. A sequence of processing actions (for example, database changes) that must be completed before any of the individual actions can be regarded as committed. When changes are committed (by successful completion of the LUW and recording of the synch point on the system log), they do not need to be backed out after a subsequent error within the task or region. The end of an LUW is marked in a transaction by a synch point that is issued by either the user program or the CICS server, at the end of task. If there are no user synch points, the entire task is an LUW.

LPAR.

Logical Partition is a subset of the CEC hardware. The CEC resources, CPUs and main memory, can be shared between LPARs. Each LPAR is capable of running an instance, or image, of an

operating system.

On-line Transaction Processing (OLTP).
: A style of computing that supports interactive applications in which requests submitted by terminal users are processed as soon as they are received. Results are returned to the requester in a relatively short period of time. An on-line transaction processing system supervises the sharing of resources to allow efficient processing of multiple transactions at the same time.

Parallel Sysplex.
: This is a sysplex that uses one or more coupling facilities.

proxy.
: A software gateway between connecting networks that allows communication between the two networks, by acting as both a client and a server. A popular usage of a proxy is a HTTP proxy server, which allow Web browsers in a private intranet to connect to Web servers on the Internet, but restricts all other network communications between the two networks.

pseudo-conversational.
: A type of CICS application design that appears to the user as a continuous conversation but consists internally of multiple tasks.

server.
: Any computing resource dedicated to responding to client requests. Servers can be linked to clients through LANs or WANs to perform services, such as printing, database access, fax, and image processing, on behalf of multiple clients at the same time.

Socket Secure (SOCKS).
: An proxy gateway that allows compliant client code (client code made socket secure) to establish a TCP/IP session with a remote host via means of the SOCKS gateway.

Standard Generalized Markup Language (SGML).
: The standard that defines several markup languages, HTML included.

synchronous.
: (1) Pertaining to two or more processes that depend on the occurrence of a specific event such as a common timing signal. (2) Occurring with a regular or predictable time relationship.

syncpoint (Synchronization point).
: A logical point in execution of an application program or transaction where the changes made to the recoverable resources are consistent, complete and can be committed. The output, which has been held up to that point, is sent to its destination, the input is removed from the message queues, and the database updates are made available to other applications.

sysplex.
: A sysplex is a set of MVS systems (also called images) that communicate using multi-system hardware components and software. Systems in a sysplex will share disk storage.

transaction.
: A unit of processing (consisting of one or more application programs) initiated by a single request. A transaction can require the initiation of one or more tasks for its execution.

transaction processing.
: A style of computing that supports interactive applications in which requests submitted by users are processed as soon as they are received. Results are returned to the requester in a relatively short period of time. A transaction processing system supervises the sharing of resources for processing multiple transactions at the same time.

transaction routing.
: Enables a terminal connected to one CICS system to run a transaction in another CICS system. It is common for CICS/ESA, CICS/VSE, and CICS/MVS users to have a terminal-owning region (TOR) that "owns" end-user network resources.

List of abbreviations

AIX	Advanced Interactive eXecutive
AOR	application owning region
API	application programming interface
APPC	Advanced Program-to-Program Communication
ASCII	American National Standard Code for Information Interchange
BLI	business logic interface
BMS	basic mapping support
CGI	Common Gateway Interface
CICS	Customer Information Control System
CIG	CICS Internet Gateway
COMMAREA	communication area
CORBA	Common Object Request Broker
CSD	CICS system definition
CTG	CICS Transaction Gateway
CWI	CICS Web Interface
CWS	CICS Web support
DNS	Domain Name Server
DPL	distributed program link
DTP	distributed transaction processing
ECI	external call interface
EJB	Enterprise JavaBeans
EPI	external presentation interface
ESA	Enterprise Systems Architecture
ESI	external security interface
EXCI	external CICS interface
HOD	Host on-Demand
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
IBM	International Business Machines Corporation
IIIOP	Internet Inter ORB Protocol
IP	Internet Protocol
ISC	intersystem communication
ITSO	International Technical Support Organization
JCICS	CICS Java support
JDK	Java Development Kit
JCT	Journal Control Table
JIT	just-in-time
JNI	Java Native Interface
JRE	Java Runtime Environment
JVM	Java Virtual Machine
LAN	local area network

LUW	logical unit of work
OLTP	on-line transaction processing
RACF	Resource Access Control Facility
RDO	Resource definition on-line
RMI	Java remote method invocation
RPC	remote procedure call
SNA	Systems Network Architecture
SIT	system initialization table
SNT	sign-on table
SOCKS	socket secure
SSL	Secure Socket Layer
TCP/IP	Transmission Control Protocol/Internet Protocol
TOR	terminal owning region
TRUE	task-related user exit
URI	Uniform Resource Identifier
URL	Uniform Resource Locator or Universal Resource Locator
WLM	work load manager
WOR	Web owning region
WWW	World Wide Web

Index

Numerics

A

B

C

D

E

H

I

J

L

N

P

R

S

T

V

W

List of Figures

Chapter 1: CICS and Web-enabling

Chapter 2: Performance and capacity planning factors

Chapter 3: The 3270 green screen Trader application

Chapter 4: CWS with the 3270 Web bridge

Chapter 5: CWS with Web-aware presentation logic

Chapter 6: SSL with CWS

Chapter 7: The OS/390 CTG

Chapter 8: Conclusions and recommendations

Chapter 9: CICS Web capacity planning example

List of Tables

Chapter 2: Performance and capacity planning factors

Chapter 3: The 3270 green screen Trader application

Chapter 4: CWS with the 3270 Web bridge

Chapter 5: CWS with Web-aware presentation logic

Chapter 6: SSL with CWS

Chapter 7: The OS/390 CTG

Chapter 9: CICS Web capacity planning example

Appendix A: Test environments

Appendix B: Performance data

List of Sidebars

Chapter 1: CICS and Web-enabling

Chapter 5: CWS with Web-aware presentation logic

Chapter 6: SSL with CWS

Chapter 7: The OS/390 CTG

How to get ITSO redbooks