IBM CICS Transaction Server V3.1
Web services performance

May 2006

IBM Hursley Performance Team
John Burgess
Trevor Clarke

Overview

In this paper we compare the performance of the same workload in 3270-based and Web service architectures. We used one of our traditional CICS®/DB2® workloads and explain an easy way to measure its performance. Then we'll look at how we converted this to a Web services-based architecture using the CICS Web Services Assistant (a CICS Transaction Server V3.1 feature) and show the results of re-measuring performance with the new architecture. We will also show the effects of using HTTPS to secure the TCP/IP sessions.

Highlights for CICS Web services

The measurements that we took highlight the following points:

- CPU usage in the CICS Web services solution scales in a linear fashion, and exploits multiple CPUs concurrently by using the CICS OTE to achieve high transaction rates.
- CPU usage per request is closely related to the complexity of the language data structure passed between client and server. Simplifying these gives better performance.
- The additional cost of a Web service solution depends on your original architecture and the amount of application code. Our workload migration from 3270 showed a 14% increase in CPU per transaction.
- We have made many Web service performance improvements since CICS V2.2, and continue to do so.
- Using SSL for encrypting and decrypting data between client and server is relatively inexpensive if hardware assist is available.
- Enabling business components as CICS Web services is straightforward.

This paper has enough detail to estimate the costs for enabling your own Web services.

3270 CICS/DB2 workload (RTW)

The RTW is a standard workload used by the Hursley Performance team to assess changes in performance characteristics within new releases of CICS code when running DB2 applications. In these applications, the presentation logic is separated from the business logic by an EXEC CICS LINK.

The workload has the following characteristics.

- All COBOL programs
- 7 unique transactions
- 20 Database Tables
- Average of 200 DB2 calls per transaction
- 54% Select, 1% insert, 1% update, 1% delete, 8% open cursor, 27% fetch cursor, 8% close cursor

The front-end presentation logic is very simple. It receives data from the terminal, passes it to the back-end business logic, and sends a response to the terminal when the logic returns control. In this comparison, the front-end function will be replaced by the Web services interface, consisting of a TCPIPSERVICE and a Web services PIPELINE, with the same data passed to the back-end program. In the Web services

architecture, this data is generated by the client in XML format. The PIPELINE converts it to the correct COMMAREA structure for the business logic, and back into outbound XML for the client. Because of this conversion, the structure/complexity of the COMMAREA becomes a major performance factor.

Network simulation

In the 3270 version of the RTW, the network is simulated by TPNS LU2 terminals. TPNS runs on a separate z/OS system to avoid it affecting the CPU usage of the system under test. The transaction rate can be increased by dynamically reducing the 'user think time'. For this evaluation we have used 5 different transaction rates to ensure scalability of each solution.

Environment

- LPAR with 3 dedicated CPUs on a 2084 332
- 2084 includes built in CPACF (CP assist for Cryptographic Function) and one PCIXCC (Peripheral Component Interconnect Extended Cryptographic Coprocessor)
- ESS800 DASD
- z/OS 1.7
- DB2 V7.1
- CICS TS V3.1 with PTFs UK11615 and UK11616

Performance data collection

We used RMF to measure two important metrics for this comparison:

- Transaction rate (ETR), recorded by using TSO's WLM definition panels to put the CICS APPLID in a unique reporting group in the CICS subtype.
- CPU usage, recorded by putting the CICS region (JOBNAME) in a unique WLM reporting group in the JES subtype.

After making the WLM definitions, we generated RMF 'Mon I' interval reports for the 5 different ETRs. Since RMF reports CPU-used for a region as a percentage of the RMF interval, with multiple CPUs available it is possible that the CPU time used is greater than the interval time and hence reported as > 100%.

You can see the ETR and CICS CPU percentages in Figure 1.

3270 based RTW Benchmark results

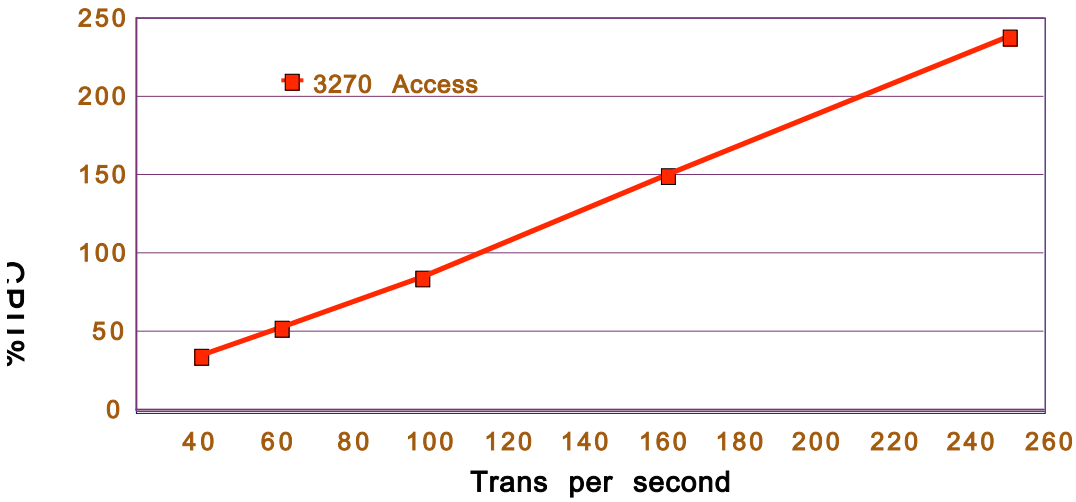

Figure 1

We calculated CPU time per transaction of approximately 8.8 millisecond by dividing the CPU% used by the transaction rate for each interval and then taking the average of the five points.

After establishing this base for the 3270 architecture, we converted the workload to a Web services architecture and measured again.

Conversion to Web services

CICS side

We enabled the same backend business logic components as Web services. There are currently two ways to achieve this. One is by using WebSphere Developer for zSeries (WD/z) to generate CICS programs that will convert XML to COMMAREA and back again. This technique results in a unique CICS conversion program for each Web service. The other way is to use the CICS Web services Assistant (CWA) to generate a WSBIND file for each Web service which is used by a single generic converter program for all Web services in CICS.

We wanted to evaluate a CICS-only solution, so we chose CWA to enable our application programs as Web services.

LS2WS

The CICS Web services Assistant supplies JCL (L2WS) which takes a Language Structure source file and a parameter file as input. The Language structure is the COMMAREA that's the target for the converted XML. The field names in the COMMAREA are used to create the XML tag names. The parameter file describes certain characteristics of the Web service being created as shown in this example:

```
PDSLIB=//JOHNB.WEB.SERVICE  (library containing commarea)
LANG=COBOL
```

```
PGMNAME=DB900002              (back-end CICS application)
REQMEM=COMMAREA               (inbound commarea)
RESPMEM=COMMAREA              (outbound commarea)
LOGFILE=/u/clarket/order0
URI=wsdl/soap11/order0        (URI for web service)
PGMINT=COMMAREA
WSBIND=/u/clarket/cts31/pipelines/soap11/pickup/order0.ws
bind
WSDL=/u/clarket/wsdl/order0.wsdl
```

The sample JCL to run LS2WS is supplied in DFHINST. When run, it creates WSDL that describes how the client can invoke the Web service. Another output is the WSBIND file that describes to the converter program how it should generate the COMMAREA from the inbound XML and how to generate the outbound XML from the COMMAREA. It is placed in an HFS directory which is associated with a PIPELINE definition. When the PIPELINE is installed in CICS, a Web service and a URIMAP definition are installed for all WSBIND files that are found in this directory.

TCPIPSERVICE

Since this project uses SOAP/HTTP as the Web service transport, the first resource needed is the CICS TCPIPSERVICE. We simply defined one TCPIPSERVICE for a Port 5025 and left the default transaction name as CWXN. The CWXN transaction uses the inbound URI to locate the corresponding URIMAP which defines the PIPELINE to use, the Web service name and transaction name (if not the default).

PIPELINE

The PIPELINE definition, new to CICS TS V3.1, points to the directory containing the WSBIND files needed to autoinstall Web services and associated URIMAPs when you install the PIPELINE. However, we needed to change the transaction ID from CPIH to unique names so that we could associate the different transactions with different DB2 Entries. We manually defined and installed all the URIMAPs and Web service definitions so that we could give each Web service a unique Transaction ID. You still need to define a transaction resource but the program name on the definition is ignored and it runs the PIPELINE program DFHPIDSH under that Transaction ID instead.

Overall view of CICS Web services

Figure 2 shows the basic components of the Web services infrastructure.

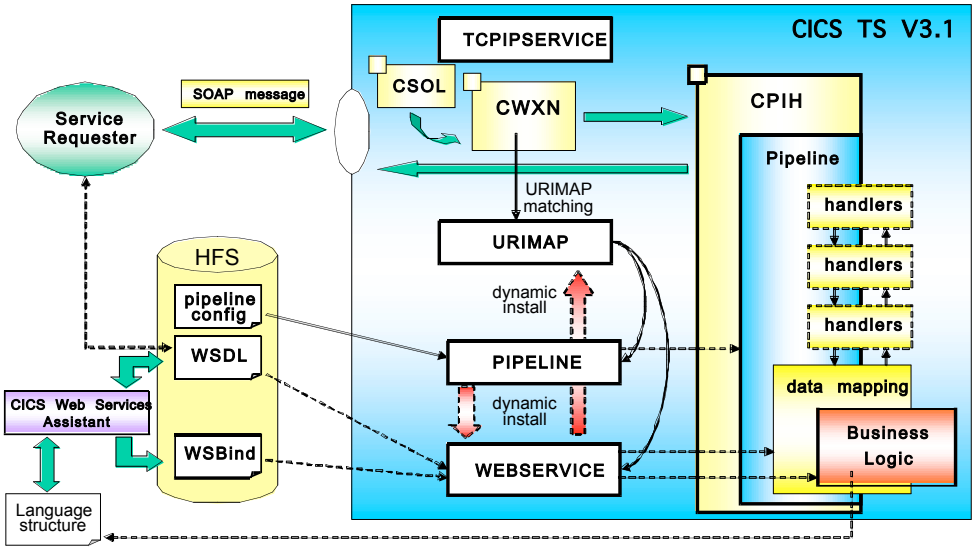

Figure 2

Client side

Once an application program has been made available as a Web service, it is the responsibility of the client side to exploit it. How you accomplish this depends on the platform and tooling you intend to use, but the common factor is the WSDL file that was created by the LS2WS JCL. The WSDL describes how to invoke the Web service and structure of the XML fields, and it can be used at run time by an application or it can be used by platform-dependent pre-processing tooling to build the code needed to invoke the Web service. For our testing purposes, we used our own internal tools to simulate network clients invoking these services – the details are not relevant to the results of the test.

Web service benchmark results

We used the same techniques as before to collect performance data for this variation of the workload. The tool simulating the network clients was run at 5 different "think times" to match the previously achieved transaction rates and performance data collected at these throughput rates. The data was mapped onto the chart (see Figure 3) and compared to the previous 3270 architecture. The *Trans per second* for Web services is based on what we call a business transaction -- that is, we count a CWXN and the corresponding application transaction as one. Thus the *Trans per sec* is the same as the inbound Web service requests per second, and is the same as the RMF transaction rate divided by two.

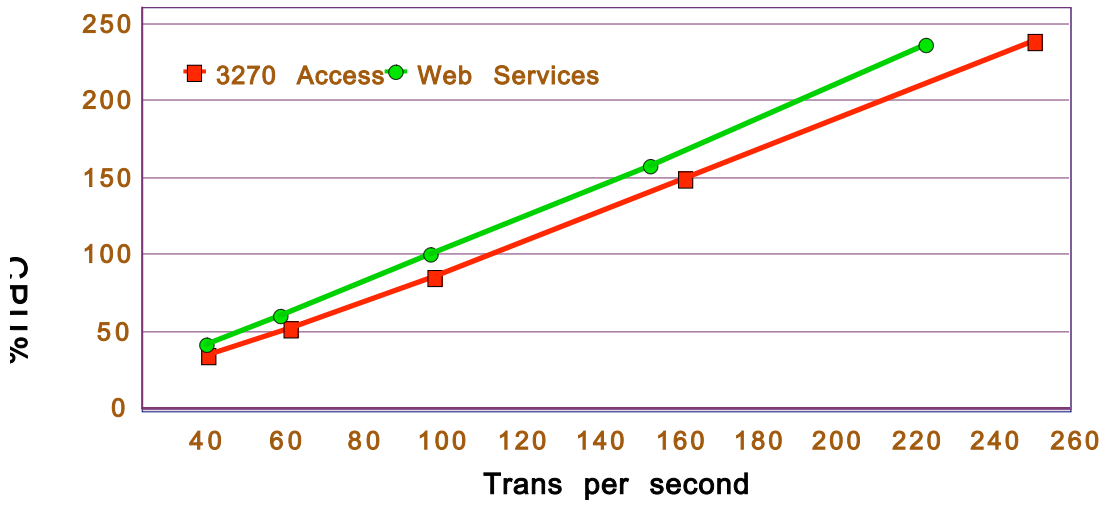

Figure 3

Using the same methodology as before, the average cost of these applications used as Web services is 10.1 milliseconds of CPU per transaction. This is approximately 14% more CPU per transaction than for the 3270 architecture. The percentage change you will see in your own system will depend on the architecture that you are migrating from and the original size of your applications.

Using HTTPS

To assess the overhead of encrypting the data, we ran the same workload with SSL connections and encrypted and decrypted the data between the clients and CICS Transaction Server using the Triple Des 168 encryption algorithm. As in the previous environment we used persistent connections.

We achieved this change of protocol on the server side by changing the TCPIPSERVICE definition to SECURITY SSL(YES). Figure 4 shows the results.

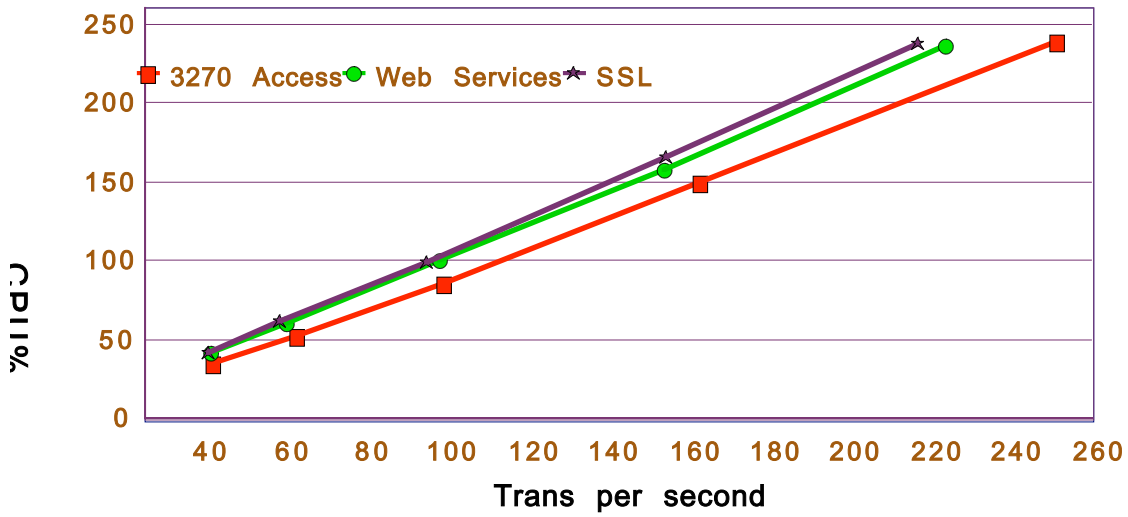

Figure 4

There was an increase in CPU per transaction of about 3% when using HTTPS persistent connections compared to HTTP. The cost of SSL encryption and decryption will increase as the length of the data being transferred increases. In this case the data passing between client and server was about 1K bytes per request. On this system, all the encryption and decryption is done in the hardware by the CPACF associated with each CPU, making SSL relatively inexpensive.

Effects of COMMAREA complexity

The greatest factor in Web services CPU costs is the XML parsing due to the complexity of the COMMAREA structure. Data length and number of elements both play a part in this. As the number of elements in the SOAP message increases, so does the cost of parsing them. The length of the resulting data on the wire is affected, not only by the number of elements, but also the length of the XML tags. You can use the table to estimate additional costs for your own applications. If you use CWA, the maximum tag name length is the same as the maximum COBOL data name length (30 bytes), so we have shown the two extremes.

	1 byte tag name	30 byte tag name
Every 100 elements Inbound	0.27ms CPU	0.56ms CPU
Every 100 elements Outbound	0.18ms CPU	0.39ms CPU

For example, our RTW workload had a COMMAREA with 10 inbound and 10 outbound elements. You might estimate the extra costs for an application with 50 inbound and 100 outbound elements with average tag sizes of 15 bytes like this:
(0.27 + (0.56-0.27)*15/30)*50/100 = 0.20millisecs CPU for inbound elements
(0.18 + (0.39-0.18)*15/30)*100/100) = 0.28millisecs CPU for outbound elements

That is, approximately an additional 0.48millisecs of CPU per transaction.

Conclusion

There have been many performance improvements since the SOAP for CICS Feature was introduced in CICS TS V2.2 and we are continually investigating ways of improving performance even more.

These current performance measurements show that the Web services infrastructure processing costs, excluding any application business logic, is just over 1millisec of CPU per request for simple COMMAREA structures using HTTP on our 2084. Use the LSPR to scale this to your own CPU model for capacity planning purposes.
The standard PIPELINE runs in OPENAPI and can also exploit multiple CPUs within the CICS address space concurrently, making this solution scaleable within one CICS region.