z/OS Connect Enterprise Edition V3.0

Performance Summary using the CICS Service Provider

Version 1.0

May 2018

Alan Hollingshead
alan_hollingshead@uk.ibm.com

z/OS Connect EE
IBM UK Laboratories
Hursley Park
Winchester
Hampshire
SO21 2JN

1.1 Notices

This report is intended for Architects, Systems Programmers, Analysts and Programmers wanting to understand the performance characteristics of z/OS Connect EE V3.0. The information is not intended as the specification of any programming interfaces that are provided by z/OS Connect EE.

It is assumed that the reader is familiar with the concepts and operation of z/OS Connect EE V3.0.

References in this report to IBM products or programs do not imply that IBM intends to make these available in all countries in which IBM operates.

Information contained in this report has not been submitted to any formal IBM test and is distributed “asis”. The use of this information and the implementation of any of the techniques is the responsibility of the customer. Much depends on the ability of the customer to evaluate this data and project the results to their operational environment.

Disclaimer: All performance data this is contained in this report was obtained in the specified operating environment and configurations, and must be considered as an example. Performance characteristics of other operating environments might differ.

IBM does not represent, warrant, or guarantee that the same or similar results will be achieved in a user's environment as the results that are shown in this report.

1.2 Trademarks and service marks

1.3 Terminology

CICS	-	CICS Transaction Server
Cost per transaction (ms)	-	CPU usage per transaction, in milliseconds
CP	-	Central Processor
CPU %	-	Percentage of CPU time used by transactions running on general purpose processors
EE	-	Enterprise Edition
GCP	-	General purpose Central Processor
JIT optimization	-	Just-In-Time optimization. The JIT compiles the JVM bytecode to machine code at runtime and uses optimization techniques such as moving calculations and caching them, moving methods to be inline, removing unnecessary locks, thus allowing the code to become smaller and faster.
PID	-	Product Identification Number

RMF	-	Resource Measurement Facility
SMF	-	System Management Facility
SSL	-	Secure Sockets Layer
TPS	-	Number of Transactions Per Second
TT	-	Think Time in seconds. The time between individual requests.
Workload Driver	-	An application written for the purpose of these performance tests to simulate multiple simultaneous client requests
zIIP	-	IBM z Systems Integrated Information Processor

2. Overview

This report contains performance measurements for z/OS Connect EE V3.0, program number (PID) 5655-CEE, using the CICS service provider to CICS Transaction Server. The CICS service provider is shipped as part of z/OS Connect EE V3.0.

What is z/OS Connect EE?

IBM z/OS Connect EE V3.0 delivers RESTful APIs as a discoverable, first-class resource with Swagger 2.0 descriptions. It includes an API package artifact that encapsulates the RESTful API, together with necessary detail to invoke underlying services in the z/OS subsystems (such as CICS Transaction Server, IMS, DB2 or MQ).

API Mapping

API mapping adds an abstraction layer between the API consumer and the underlying z/OS assets, allowing in-line manipulation of requests such as the mapping of HTTP headers, pass-through, redaction or defaulting of JSON fields, and rearranging the order of JSON fields and data.

Service Providers

IBM z/OS Connect EE V3.0 supports various service providers to access major z/OS subsystems such as CICS Transaction Server and IMS. This report uses the CICS service provider.

3. Test Environment

This report focuses on an end-to-end solution that uses the CICS service provider to access a CICS region running CICS Transaction Server 5.4 over an IPIC connection. It also uses the API mapping feature of z/OS Connect EE V3.0 to map fields from the request to the channel payloads.

Figure 1 shows the flow of requests and responses for the scenario:

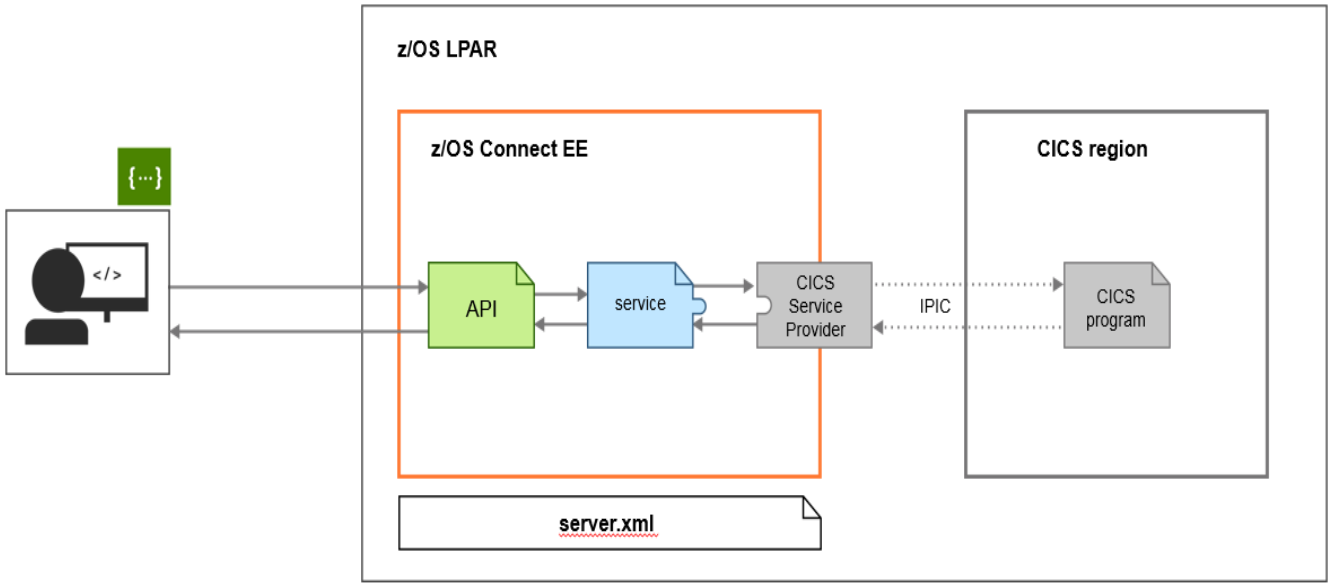


Figure 1 End-to-end solution using the CICS service provider

3.1 Flow of work

The flow of work is as follows:

1. The workload driver sends concurrent client HTTP JSON requests to z/OS Connect EE using persistent HTTP connections.
2. The JSON request calls the API feature within z/OS Connect EE.
3. The API feature performs header and field mapping.
4. The API calls the service.
5. The service invokes the CICS service provider
6. The CICS service provider forwards the transformed data request to CICS Transaction Server over an IPIC connection.
7. The CICS COBOL application sends a channel payload in the response.

3.2 Hardware

- IBM System z13 2964-NE1 model 7A5
- 10GB of Central Storage (RAM)
- LPAR with 6 dedicated GCPs (no zIIPs)

- OSA-Express5S 10GB Ethernet

3.3 Software

- z/OS V2.1
- z/OS Connect Enterprise Edition (EE) V3.0.8
  - z/OS Connect EE build 20180416-1517
  - Liberty 18.0.0.1 including Angel V7
- IBM 64-bit SDK for z/OS Java Technology Edition, Version 8.0
  - Java 1.8 SR5 FP7
- CICS Transaction Server V5.4

3.4 Workload Driver

The workload driver used to simulate multiple client requests is a Java application that runs in its own JVM on the same LPAR as z/OS Connect EE. Requests issued by the workload driver specify 127.0.0.1 (i.e. “localhost”) to minimise potential network latency.

Thinktime

Requests are issued by the workload driver at a regular pace, using a “thinktime”. For each client this is the time the workload driver waits after a response has been received before sending the next request. All performance runs in this report use a thinktime of 500ms to simulate a typical customer environment.

HTTP PUT Requests

All scenarios issue HTTP PUT requests.

3.5 Asymmetric, Flat Structure Payloads

Each payload was asymmetric, meaning that the size of data for the request was different to the size of data for the response. For example, a 50 byte payload request may have generated a 32K response (as shown in Figure 2 on page 9). The size of the response was dependent upon the data within the request.

The payloads had a flat structure, as described in *4.1 Service perfChannelService”*. This report does not provide an analysis of the effect of complex data structure transformation.

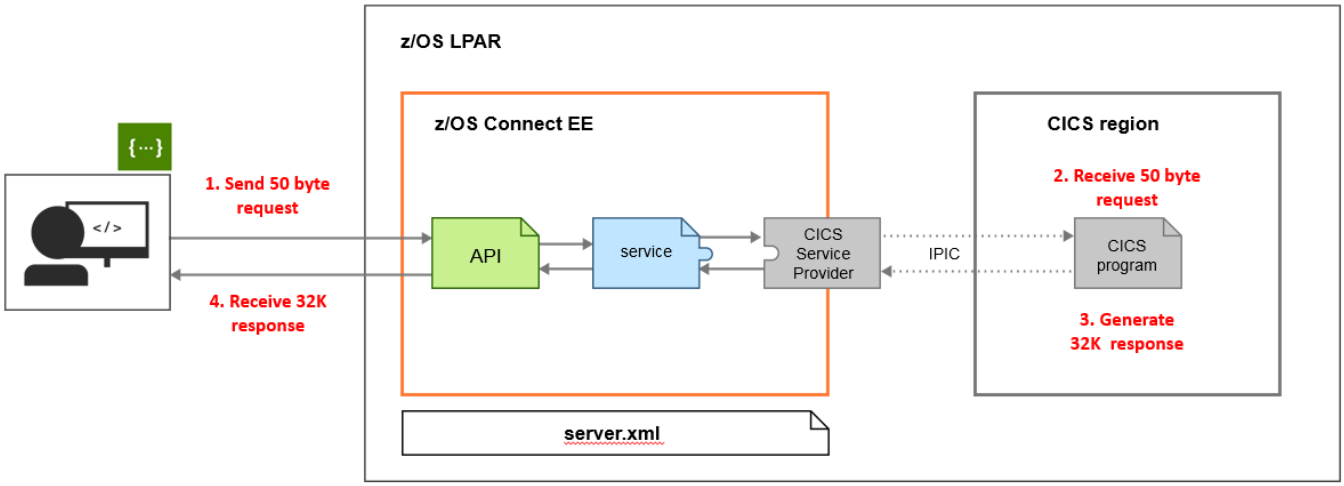


Figure 2 Example of an asymmetric payload showing a 50 byte request generating a 32K response

3.6 Performance measurement procedure

The same procedure was carried out for each measurement and consisted of the following steps:

1. z/OS Connect EE was started with “clean” option.
2. CICS cold started with a TCPIPService installed and CICS program available.
3. The workload to be measured rapidly reached a steady throughput for requests (transactions) per second. However, to ensure JIT optimisation, several hundred thousand requests were run before any measurements were taken.
4. IBM z/OS Resource Measurement Facility (RMF) data was collected at one minute intervals.

3.7 Performance metrics

The following metrics were gathered during the performance runs:

- Request rate (transactions per second)
- Average response time
- CP (general processor) usage
- Potential zIIP usage

CP and potential zIIP usage values were extracted from RMF workload activity data. RMF reports CPU utilisation that is based on uniprocessor capacity, which means that a value of 100% represents the capacity of one processor.

Figure 3 shows an example of a Workload Activity extract from an RMF report for a z/OS Connect EE server. For the one minute interval, the APPL% section shows 104.86% of general processor time was used (the “CP” value) and that 104.82% (the “IIPCP” value) is potentially offloadable to zIIP processors.

The request rate (transactions per second) and average response time values are gathered by the workload driver.

```
                                W O R K L O A D   A C T I V I T Y

     z/OS V2R1              SYSPLEX MV22              DATE 05/01/2018          INTERVAL 00.59.999   MODE = GOAL
                            RPT VERSION V2R1 RMF      TIME 13.15.29

REPORT BY: POLICY=NORMAL                          REPORT CLASS=ZOSCONN
                                                  DESCRIPTION =zosConnect reports

-TRANSACTIONS-  TRANS-TIME HHH.MM.SS.TTT  --DASD I/O--  ---SERVICE---   SERVICE TIME  ---APPL %---  --PROMOTED--  ----STORAGE----
AVG       1.00  ACTUAL                 0  SSCHRT   0.0  IOC     62813   CPU   62.905  CP    104.86  BLK    0.000  AVG    482030.5
MPL       1.00  EXECUTION              0  RESP     0.0  CPU      4681K  SRB    0.013  AAPCP   0.00  ENQ    0.000  TOTAL  482030.5
ENDED        0  QUEUED                 0  CONN     0.0  MSO         0   RCT    0.000  IIPCP 104.82  CRM    0.000  SHARED     20.00
END/S     0.00  R/S AFFIN              0  DISC     0.0  SRB       940   IIT    0.000                LCK    0.003
#SWAPS       0  INELIGIBLE             0  Q+PEND   0.0  TOT      4745K  HST    0.000  AAP      N/A  SUP    0.000  -PAGE-IN RATES-
EXCTD        0  CONVERSION             0  IOSQ     0.0  /SEC    79086   AAP      N/A  IIP     0.00                SINGLE       0.0
AVG ENC   0.00  STD DEV                0                                IIP    0.000                               BLOCK        0.0
REM ENC   0.00                                          ABSRPTN    79K                                          SHARED       0.0
MS ENC    0.00                                          TRX SERV   79K                                          HSP          0.0

TRANSACTION APPL% :   TOTAL :   CP 104.86   AAP/IIP ON CP 104.82   AAP/IIP   0.00
                      MOBILE :  CP   0.00   AAP/IIP ON CP   0.00   AAP/IIP   0.00
```

Figure 3 Example of RMF Workload Activity extract

4. API and Service Definitions

The performance runs used a single API and a single service:

- API: *perfChannel*.
- Service: *perfChannelService.*

Both the API and the service were created using the API toolkit v3.0.4.

The CICS program called by the service was *PFPADCON* and used channels and containers.

The service was created first and then imported in to the API project.

4.1 Service perfChannelService

Using the API toolkit, the z/OS Connect EE Service Editor dialog was used to create the service, *perfChannelService.*

Figure 4 shows the service configuration “Overview” tab. The CICS channel program, *PFPADCON,* was specified here.

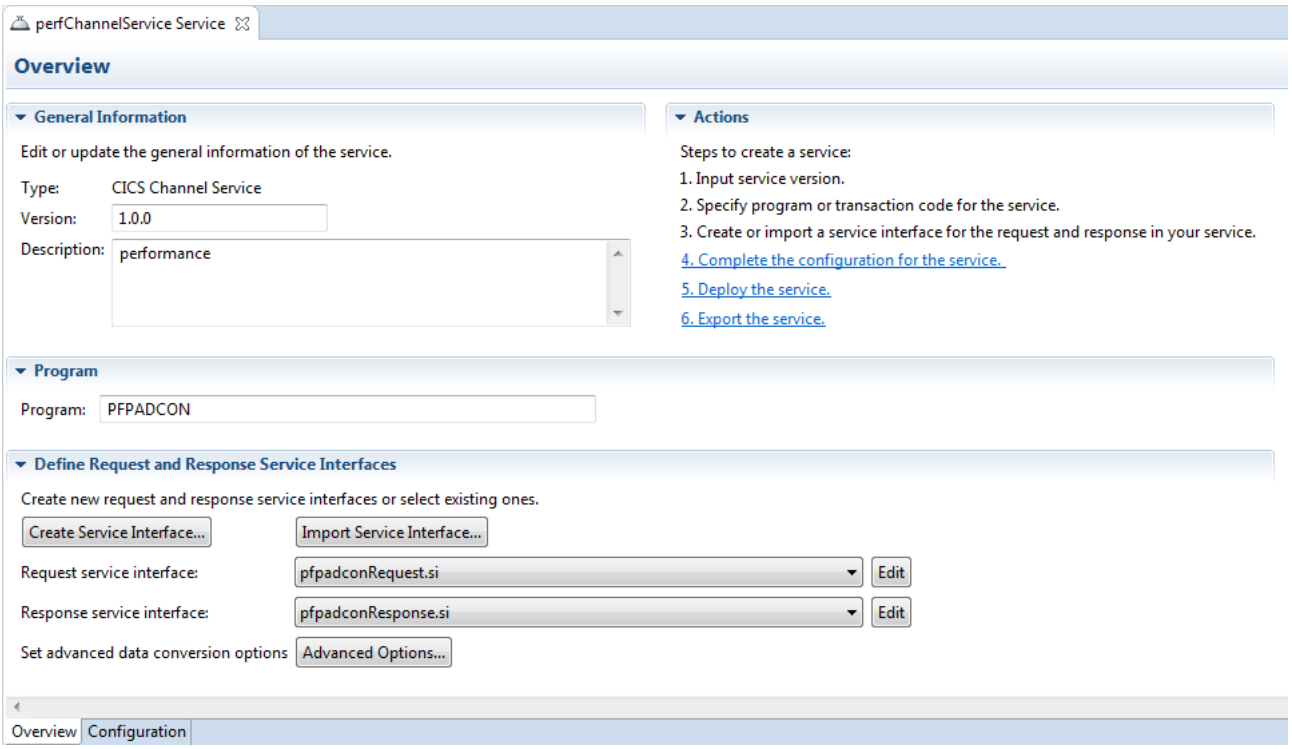


Figure 4 Service configuration "Overview" tab

The service request contained a BIT container, LEN-TO-RETURN (see Figure 5), which was populated by the client request with a value informing the CICS program to return a container of a particular size, for example, 32K.

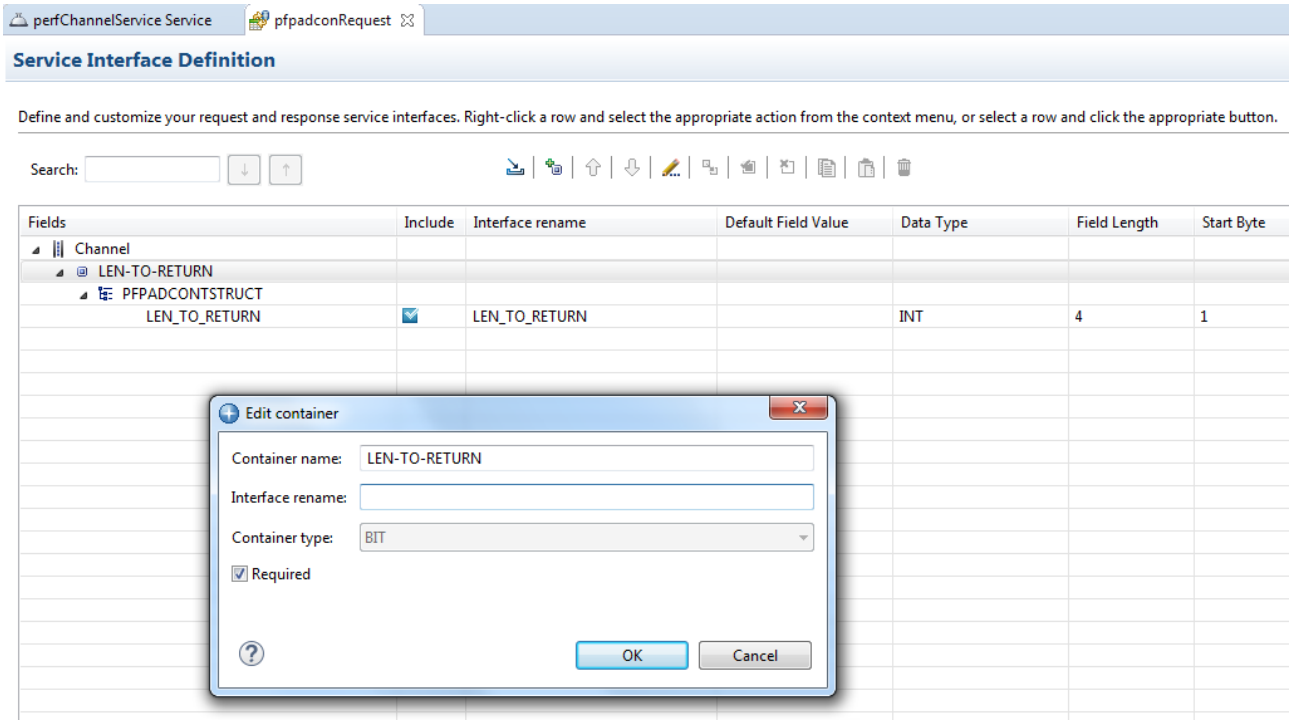


Figure 5 Service request definition

The service response contained two BIT containers, RESPONSE-DATA and RESPONSE-LENGTH (see Figure 6), and optionally a CHAR container, ERROR, for cases where an error may have occurred in CICS.

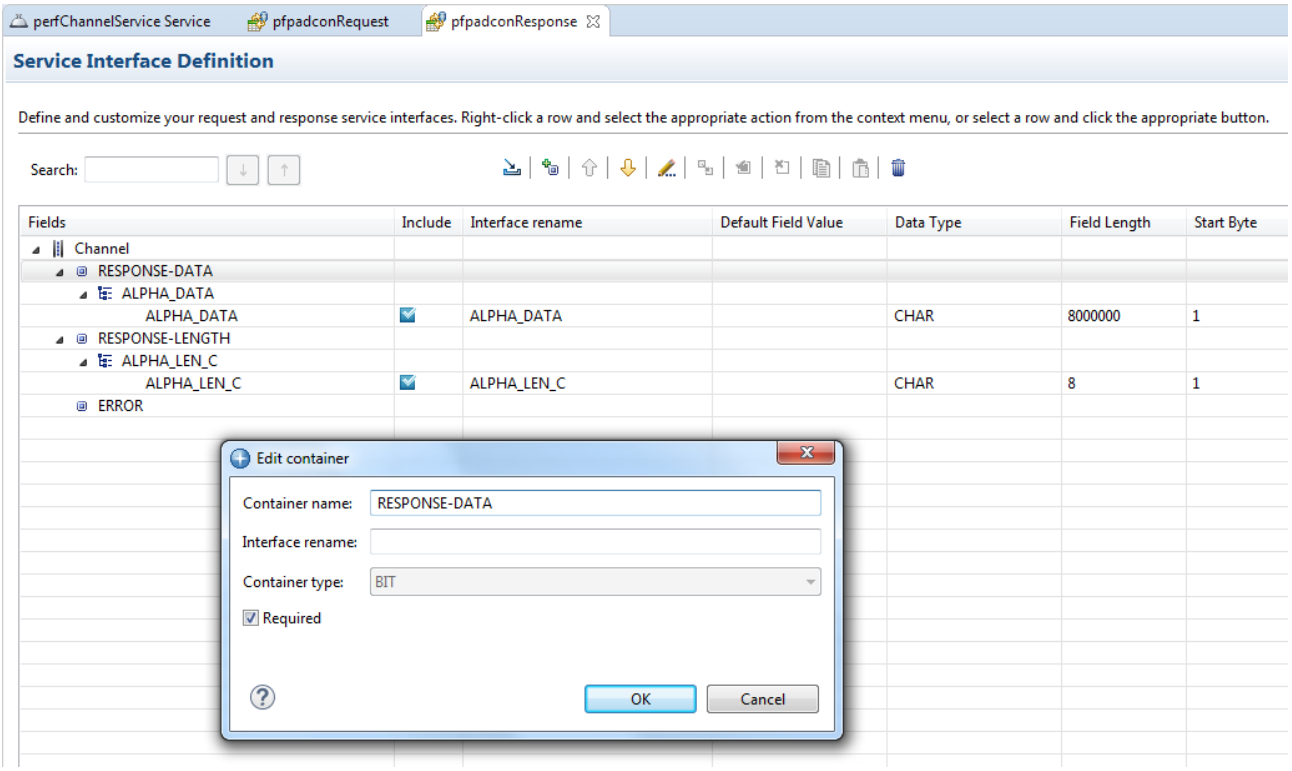


Figure 6 Service response definition

On the service “Configuration” tab (see Figure 7), a *Connection reference* of **cicsConn** was defined to match the `zosconnect_cicsIpicConnection` connection reference specified in the server.xml configuration file (see Figure 12 on page 19).

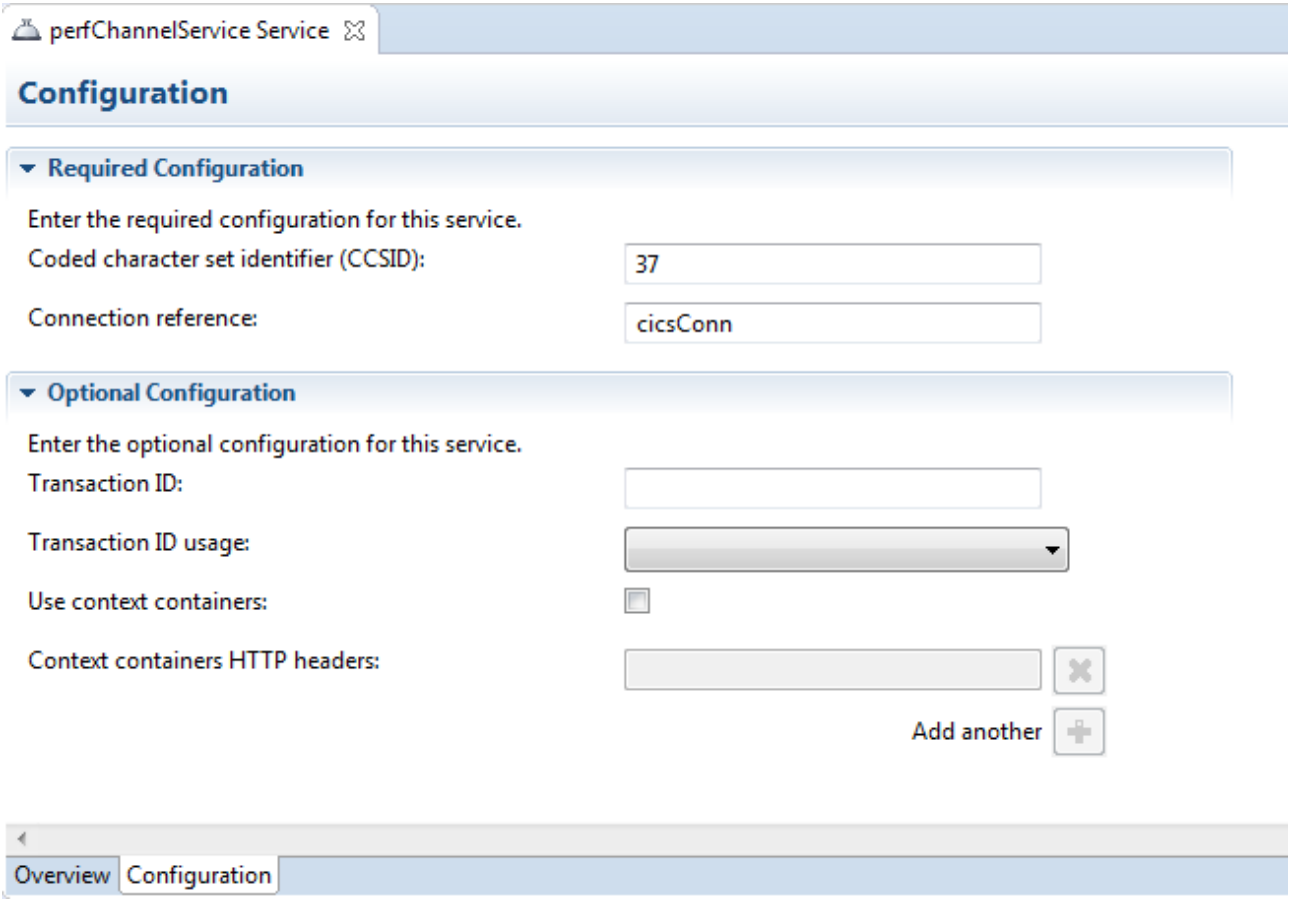


Figure 7 Service "Configuration" tab

The service definition for *perfChannelService* was ready to be used.

4.2 API perfChannel

Using the API toolkit, the z/OS Connect EE API Editor dialog was used to create the API *perfChannel*.

The API was defined with a ***Base path*** of ***/perfchannel*** and a ***Path*** parameter called ***/response/{response_size}***, *see* Figure 8. The service *perfChannelService* was imported into the API package. All requests used HTTP PUT requests.

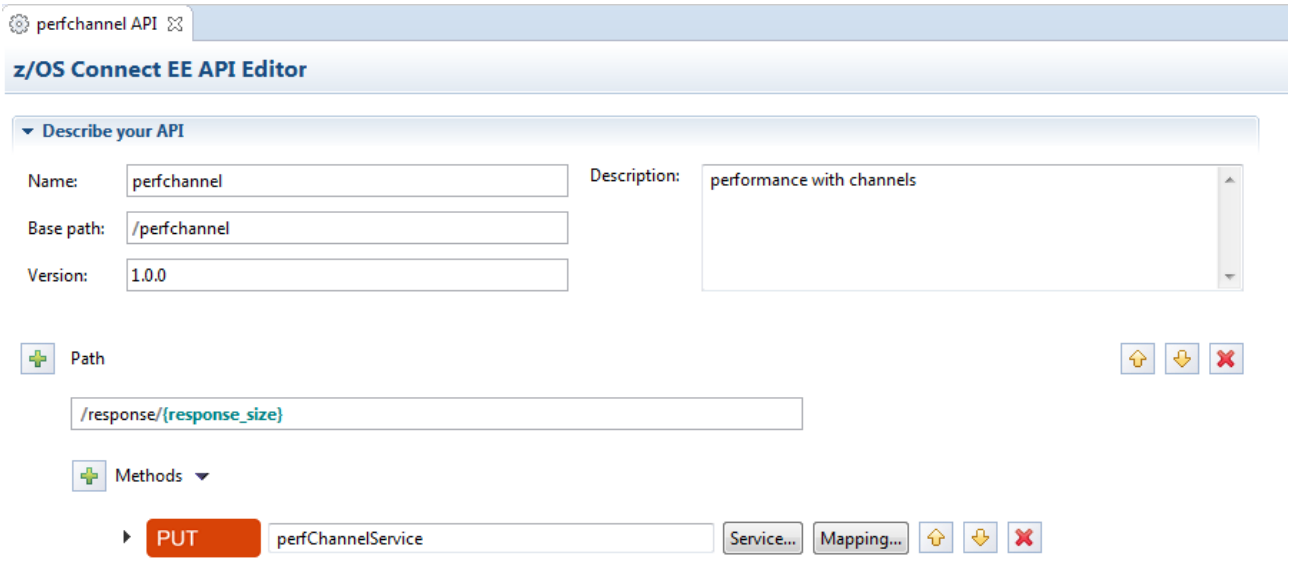


Figure 8 API definition

The request mapping, shown in Figure 9, mapped the ***Path Parameters*** value ***response_size*** to the *LEN_TO_RETURN* field.

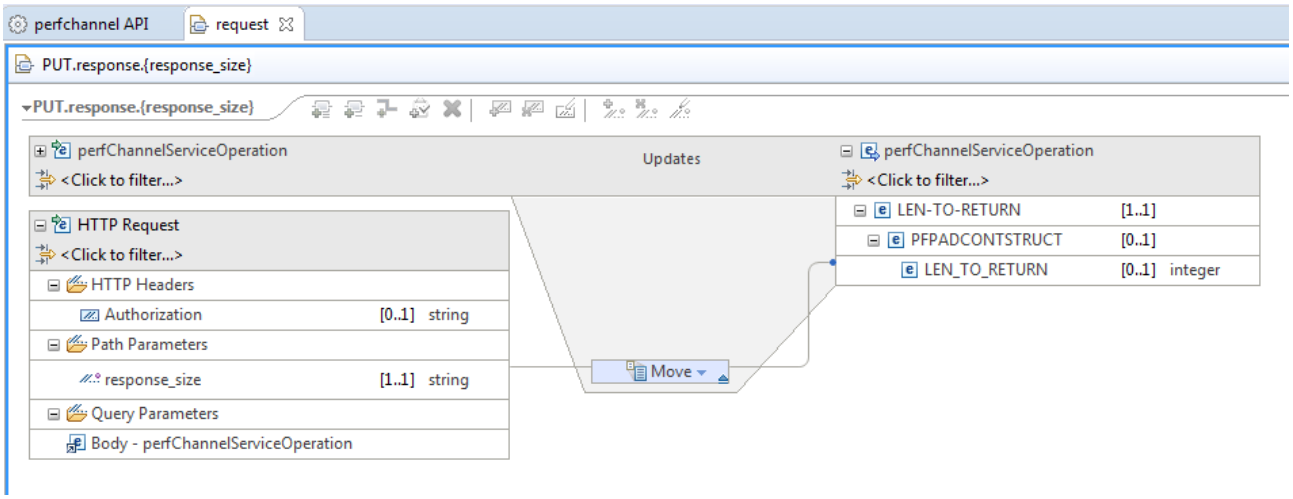


Figure 9 API request mapping

The response mapping is shown in Figure 10.

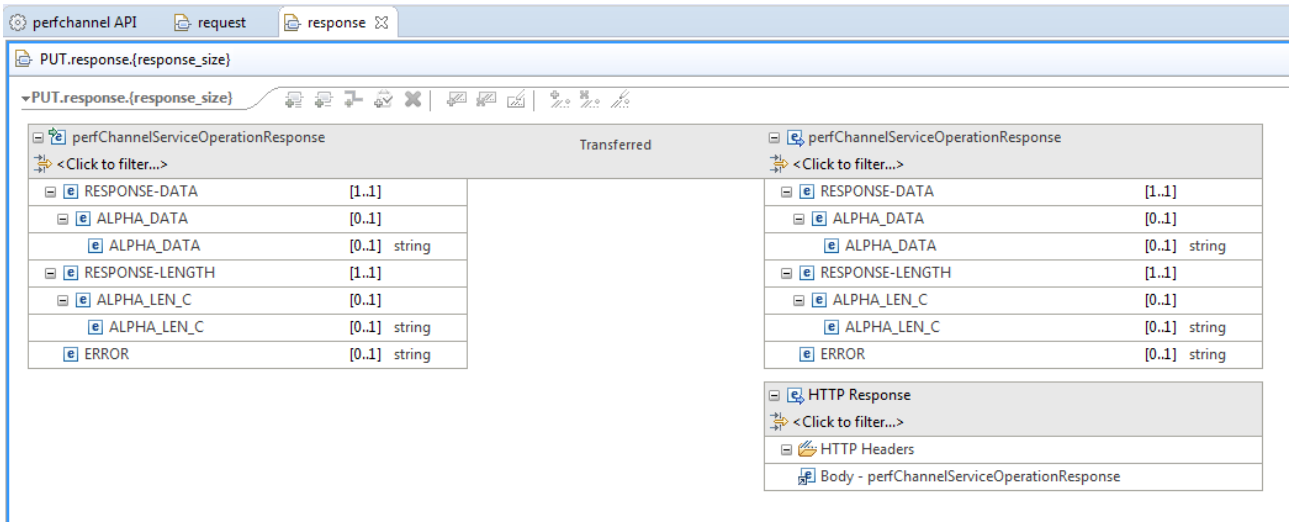


Figure 10 API response mapping

The API definition for *perfChannel* was now configured. Both the service, *perChannelService*, and the API, *perfChannel*, were then deployed to the z/OS Connect EE server.

5. Calling the API

All API calls were made from the workload driver.

5.1 Format of Requests

The details of each request are described below.

URL

Each HTTP request specified a URL with the following format:

- http://<host>:<port>/<basepath>/<relative path>

  For example, for a 1k (1024 bytes) response, the HTTP PUT request was:
  http://myhost:2222/perfchannel/response/1024

- The <basepath> in the example was “perfchannel”

- The <relative path> in the example was “response/1024”

- The “1024” at the end of the relative path was the **Path** parameter **response_size** used by the API Mapping feature within the API Editor (see Figure 8 on page 14).

Request Field

Each JSON request was approximately 50 bytes long and consisted of a single field, LEN-TO-RETURN. This field was mapped in the API toolkit from the **Path Parameters** field ***response_size*** (as shown in Figure 9 on page 14). The value in LEN-TO-RETURN was used to inform the CICS program the length of the container, RESPONSE-DATA, to return.

For example, if LEN-TO-RETURN in the request was set to 1024, then the container RESPONSE-DATA in the response was to be populated with 1024 characters by CICS.

5.2 Format of Responses

The details of the responses are described below.

Response Fields

The size of the response to be sent by the CICS COBOL application was determined by the Path Parameter ***response_size*** in the request. Figure 11 shows the response generated from the following API request:

http://myhost:2222/perfchannel/response/1024

```
{
    "perfChannelServiceOperationResponse":
    {
        "RESPONSE-DATA":
        {
            "ALPHA_DATA":
            {
                "ALPHA_DATA":
"XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXX"
            }
        },
        "RESPONSE-LENGTH":
        {
            "ALPHA_LEN_C":
            {
                "ALPHA_LEN_C": "00001024"
            }
        }
    }
}
```


Figure 11 Format of 1K response

In this report the response sizes varied from 1K to 256K. The request size remained at approximately 50 bytes regardless of the response size.

6. z/OS Connect EE Environment

This chapter describes the z/OS Connect EE environment the performance tests were run in.

6.1 JCL Configuration for z/OS Connect EE

The started procedure for z/OS Connect EE was configured with:

- REGION size 0M
- MEMLIMIT 4G

6.2 JVM Configuration

The JVM used by z/OS Connect EE was configured with:

- IBM 64-bit SDK for z/OS Java Technology Edition, Version 8.0
  - Java 1.8 SR5 FP7
- Maximum Java heap size (Xmx) 1G
  - Other Java heap (Garbage Collector) values set to minimise the impact of the heap variation during performance runs:
    - -Xms1G to match the maximum Java heap size
    - -Xmnx512M to set the maximum new (nursery) size
    - -Xmns512M to set the minimum new (nursery) size
    - -Xmox512M to set the maximum old (tenured) size
    - -Xmos512M to set the minimum old (tenured) size
- All other JVM system properties use default values (including garbage collection mode: gencon)

6.3 The server.xml Configuration File

The server.xml file used for the z/OS Connect EE server contained the following definitions:

```
<?xml version="1.0" encoding="UTF-8"?>
<server description="z/OS Connect EE 3.0. Polling not enabled.">

    <config updateTrigger="mbean"></config>

    <!-- Enable features -->
    <featureManager>
        <feature>zosconnect:cicsService-1.0</feature>
        <feature>zosconnect:zosConnectCommands-1.0</feature>
    </featureManager>

    <!-- ZosConnectManager definition -->
    <zosconnect_zosConnectManager
          requireSecure="false"
          requireAuth="false"
          operationMode="ASYNC" >
    </zosconnect_zosConnectManager>

    <zosconnect_cicsIpicConnection id="cicsConn"
          host="12.34.56.789" port="1154" sendSessions="500" />

    <zosconnect_services updateTrigger="mbean">
    </zosconnect_services>

    <!—Server port -->
    <httpEndpoint id="defaultHttpEndpoint"
          enabled="true"
          host="127.0.0.1"
          httpPort="2222" />
</server>
```

Figure 12 The server.xml configuration

Feature Definitions

The feature definitions specified in the server.xml configuration file shown in Figure 12 contained two features:

- The first feature, `zosconnect:cicsService-1.0`, was for the CICS service provider. This automatically installs the z/OS Connect EE V3.0 feature during the initialization of the z/OS Connect EE server.
- The second feature, `zosconnect:zosConnectCommands-1.0`, was to allow SDSF MODIFY commands to enable any changes made to the server.xml configuration file. Note that polling was not enabled in the server.xml configuration file as this can have an impact on performance and is not recommended in a production environment.

zosConnectManager Definition

The `zosconnect_zosConnectManager` definition specified in the server.xml configuration file shown in Figure 12 contained:

- `operationMode="ASYNC"`– this is required when z/OS Connect EE is running in standalone mode (as opposed to running inside CICS) to provide optimum performance.
- Security elements `requireSecure` and `requireAuth` were disabled as the performance runs did not use any security characteristics.

cicsIpicConnection Definition

The `zosconnect_cicsIpicConnection` definition specified in the server.xml configuration file shown in Figure 12 on page 19 contained:

- `host` and `port` for the IPIC listener on the CICS region
- `sendSessions` set to 500 IPIC sessions on the IPIC connection

The CICS region had a TCPIPService defined with `protocol IPIC` and default `URM DFHISAIP`. An IPCONN was autoinstalled when the IPIC connection was established.

zosconnect_services Definition

The `zosconnect_services` definition specified in the server.xml configuration file shown in Figure 12 on page 19 was required to make services available, including the perfChannelService. The `updateTrigger="mbean"` attribute was specified to allow SDSF MODIFY commands to be honoured for services, if required.

API

The API used for the performance runs, perfChannel, was modelled using the API Editor (see API and Service on page 11) and then deployed through the API Editor.

- The deployed API is automatically loaded upon first invocation, and does not need to be explicitly defined in the server.xml file.
- There were no interceptors used in these performance tests.

.

7. Results

The following sections of this report show the results of the performance runs.

Disclaimer: All performance data this is contained in this report was obtained in the specified operating environment and configurations, and must be considered as an example. Performance characteristics of other operating environments might differ.

IBM does not represent, warrant, or guarantee that the same or similar results will be achieved in a user's environment as the results that are shown in this report.

7.1 Transactions Per Second

The performance runs simulated:

- 100 clients
- 200 clients
- 300 clients
- 400 clients
- 500 clients

The performance runs used the following payload sizes for each response in turn:

- 1K response
- 4K response
- 8K response
- 16K response
- 32K response
- 64K response
- 128K response

Although z/OS Connect EE supports payload sizes much larger than 128K, the CPU resources available for the LPAR used for these performance runs became constrained when larger payloads with large numbers of clients were used. See *“12 Exceeding system capacity”.* This is due to available CPU capacity and is not a limitation of the product.

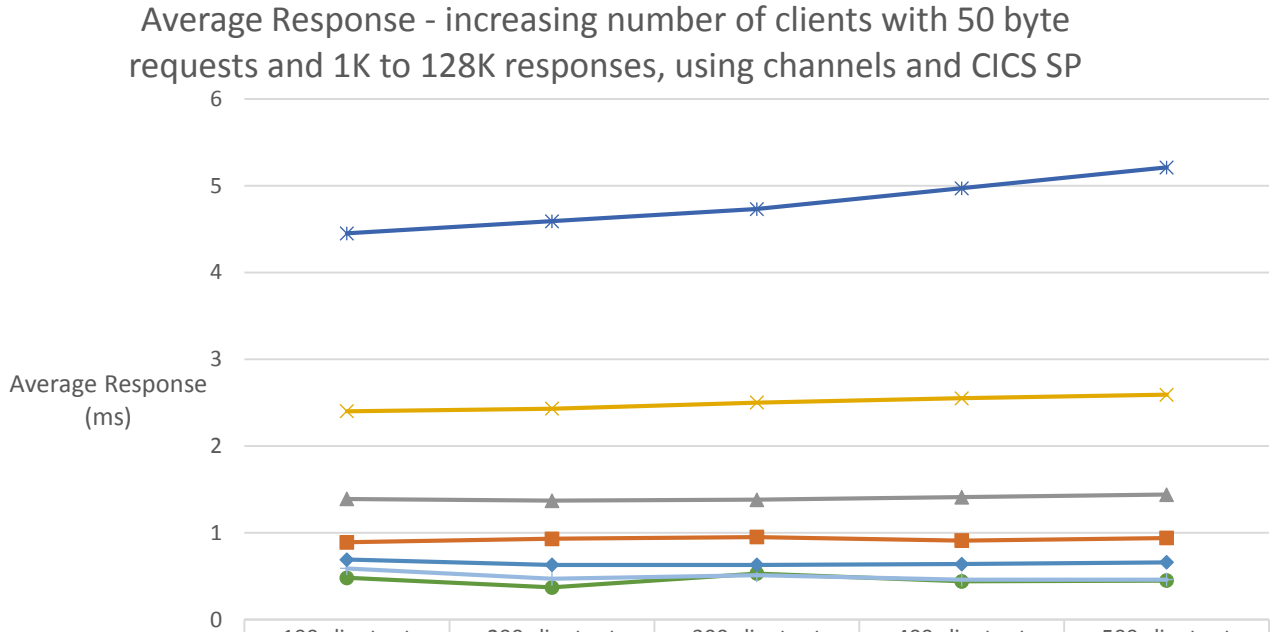
The transactions per second (TPS) for each run was maintained at the following rate regardless of payload size:

	100 clients	200 clients	300 clients	400 clients	500 clients
TPS:	200	400	600	800	1000

8. Average Response time

Ideally the average response time should be the same whether there is one client or 500 clients. Realistically, as the number of clients increase, the average response time can be expected to also increase due to resource contention in the system. Typically a maximum throughput in the system will be reached somewhere in any configuration at which point requests will begin to queue, thus increasing the average response time.

Figure 13 shows the average response for 1K to 128K responses for different numbers of clients and transaction rates.



	100 clients at 200 TPS	200 clients at 400 TPS	300 clients at 600 TPS	400 clients at 800 TPS	500 clients at 1000 TPS
1K response	0.48	0.37	0.53	0.44	0.45
4K response	0.59	0.47	0.51	0.46	0.46
8K response	0.69	0.63	0.63	0.64	0.66
16K response	0.89	0.93	0.95	0.91	0.94
32K response	1.39	1.37	1.38	1.41	1.44
64K response	2.4	2.43	2.5	2.55	2.59
128K response	4.45	4.59	4.73	4.97	5.21

Figure 13 Average Response for 1K to 128K responses with increasing numbers of clients and transaction rates.

Observations

- ✓ z/OS Connect EE demonstrated acceptable scalability for average response times.
- ✓ Increasing the number of clients to run in parallel did not create unacceptable response times.

9. CPU Cost Per Transaction

The cost per transaction in CPU terms is measured in milliseconds (ms).

The calculation is made by dividing the CPU service time by the number of seconds in the SMF interval. This is then divided by the number of transactions per second (TPS) for the same SMF interval, and finally multiplied by one thousand to get the result in milliseconds.

Figure 14 shows the CPU cost per transaction for 1K to 128K responses for different numbers of clients and transaction rates.

	100 clients at 200 TPS	200 clients at 400 TPS	300 clients at 600 TPS	400 clients at 800 TPS	500 clients at 1000 TPS
1K response	0.2	0.2	0.21	0.21	0.21
4K response	0.29	0.28	0.28	0.27	0.27
8K response	0.41	0.41	0.4	0.4	0.4
16K response	0.65	0.64	0.64	0.64	0.64
32K response	1.14	1.13	1.13	1.13	1.14
64K response	2.08	2.08	2.08	2.09	2.1
128K response	3.92	3.94	3.96	3.97	3.98

Figure 14 CPU cost per transaction for 1K to 128K responses with increasing numbers of clients and transaction rates.

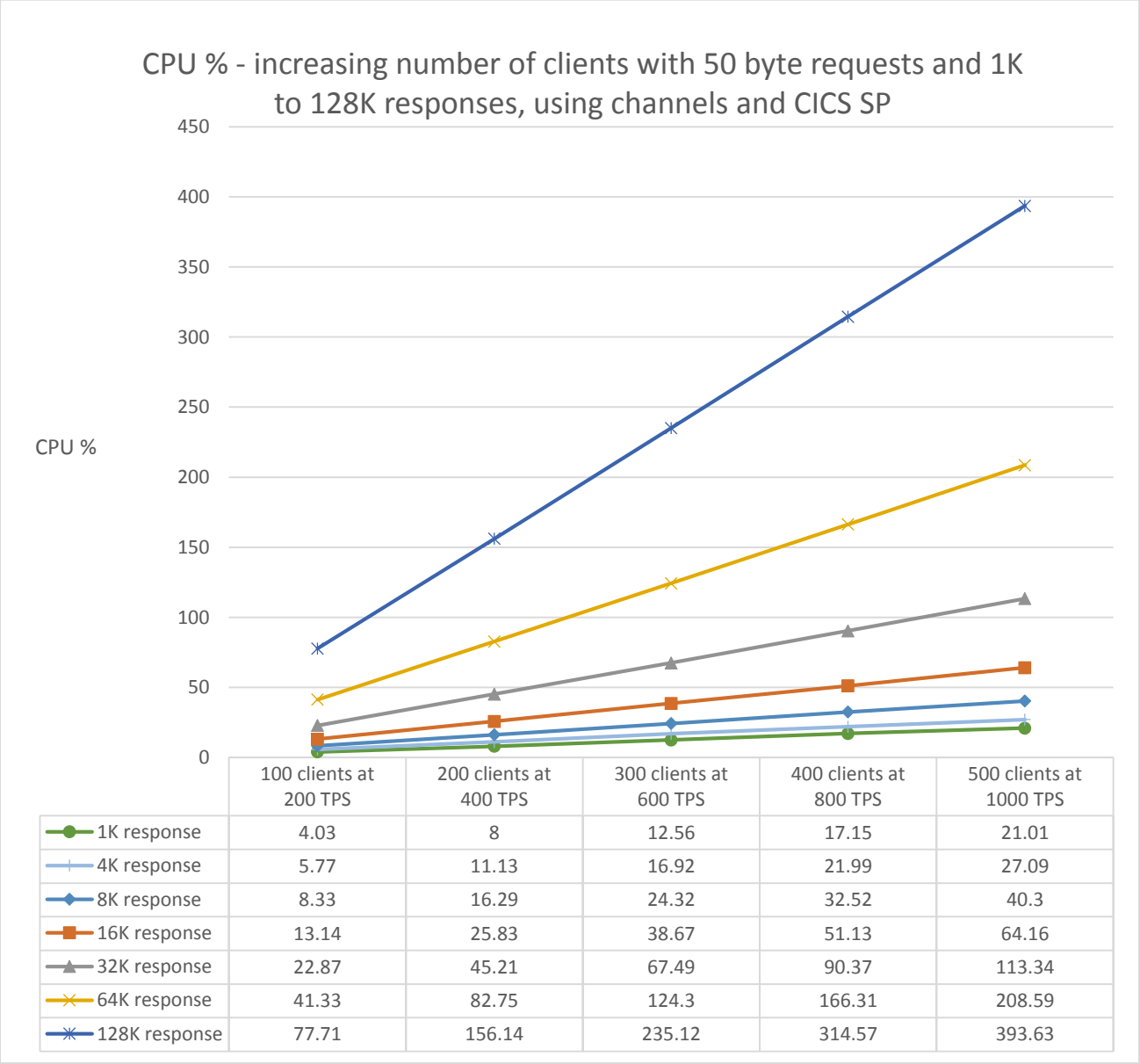
Observations

- ✓ z/OS Connect EE demonstrated good scalability with minimal increase in the CPU Cost Per Transaction as the number of clients increased.
- ✓ Increasing the number of clients to run in parallel from 100 to 500 clients resulted in little or no change in CPU Cost Per Transaction for this scenario.

10. CPU % usage

The CPU % usage will naturally increase as the number of clients running in parallel also increase. The CPU % includes all the GCPs (six in this environment), allowing a theoretical maximum of 600% for all products (including z/OS Connect EE, CICS and the workload driver, which all ran on the same LPAR).

Figure 15 shows the CPU % usage for the z/OS Connect EE server for 1K to 128K responses for different numbers of clients and transaction rates.



	100 clients at 200 TPS	200 clients at 400 TPS	300 clients at 600 TPS	400 clients at 800 TPS	500 clients at 1000 TPS
1K response	4.03	8	12.56	17.15	21.01
4K response	5.77	11.13	16.92	21.99	27.09
8K response	8.33	16.29	24.32	32.52	40.3
16K response	13.14	25.83	38.67	51.13	64.16
32K response	22.87	45.21	67.49	90.37	113.34
64K response	41.33	82.75	124.3	166.31	208.59
128K response	77.71	156.14	235.12	314.57	393.63

Figure 15 CPU% usage for increasing number of clients and transaction rates

Observations

- ✓ z/OS Connect EE demonstrated good scalability.
- ✓ The CPU % usage increased linearly as the number of clients running in parallel were increased.

11. zIIP Eligibility

As z/OS Connect EE V3.0 is almost entirely written in Java the zIIP eligibility was observed to understand the benefits of offloading work to one or more zIIPs.

The environment is configured with six GCPs. Although zIIPs have not been used, zIIP eligibility was captured and is shown alongside the actual GCP usage.

Figure 16 shows the GCP % usage and zIIP eligibility for 128K responses.

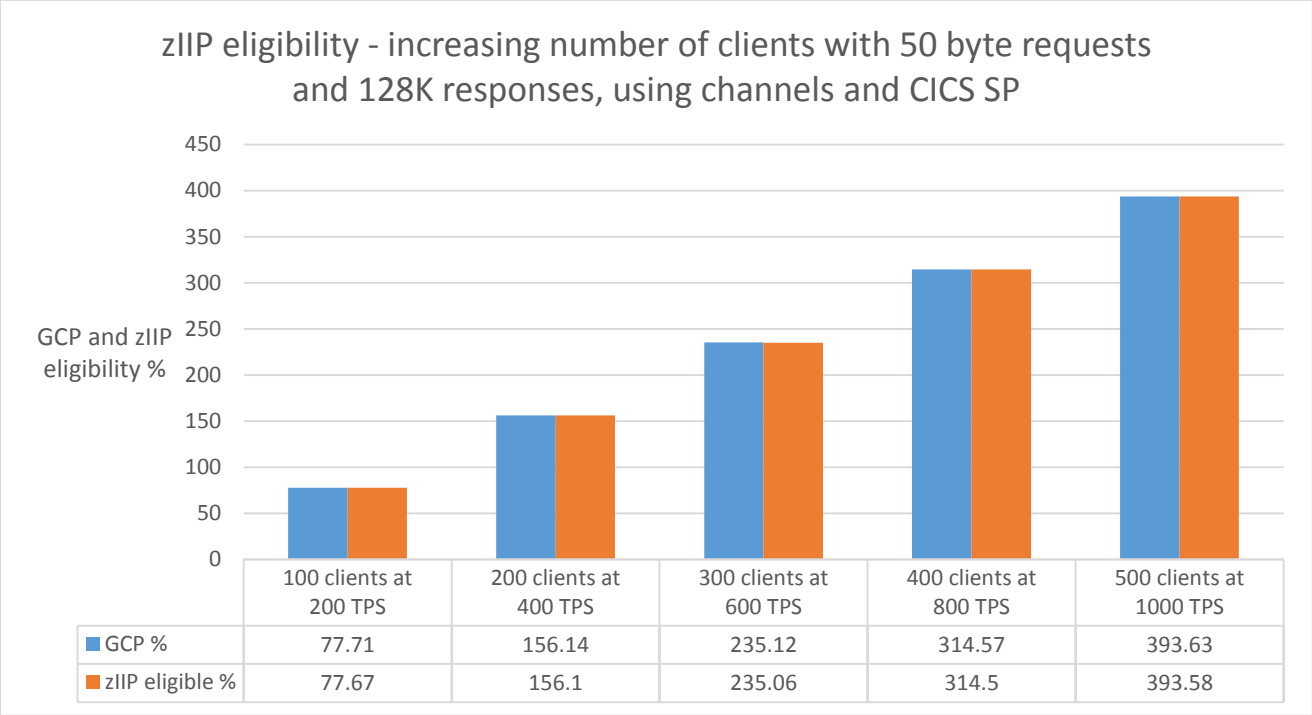


Figure 16 GCP CPU % and zIIP eligibility for increasing numbers of clients and transaction rates with 128K responses

Observations

- ✓ z/OS Connect EE is a Java-based product and so over 99% of the product is eligible to be offloaded to zIIP.
- ✓ The potential usage of zIIP processors scaled well with the increasing numbers of clients.
- ✓ Although not shown, smaller payloads exhibited the same results in that approximately 99% CPU is offloadable to zIIP.

12. Exceeding system capacity

All systems have their limits. The following graphs demonstrate the impact of CPU constraints during runs using 256K payload responses.

Note that the system has six GCPs which z/OS Connect EE shares with the other products running in the same LPAR including CICS and the (relatively light) workload driver.

12.1 Effect on TPS

Figure 17 shows good scalability when 100, 200 and 300 clients are receiving 256K payload responses, but that the system ran out of CPU when running with more than 300 clients on this LPAR.

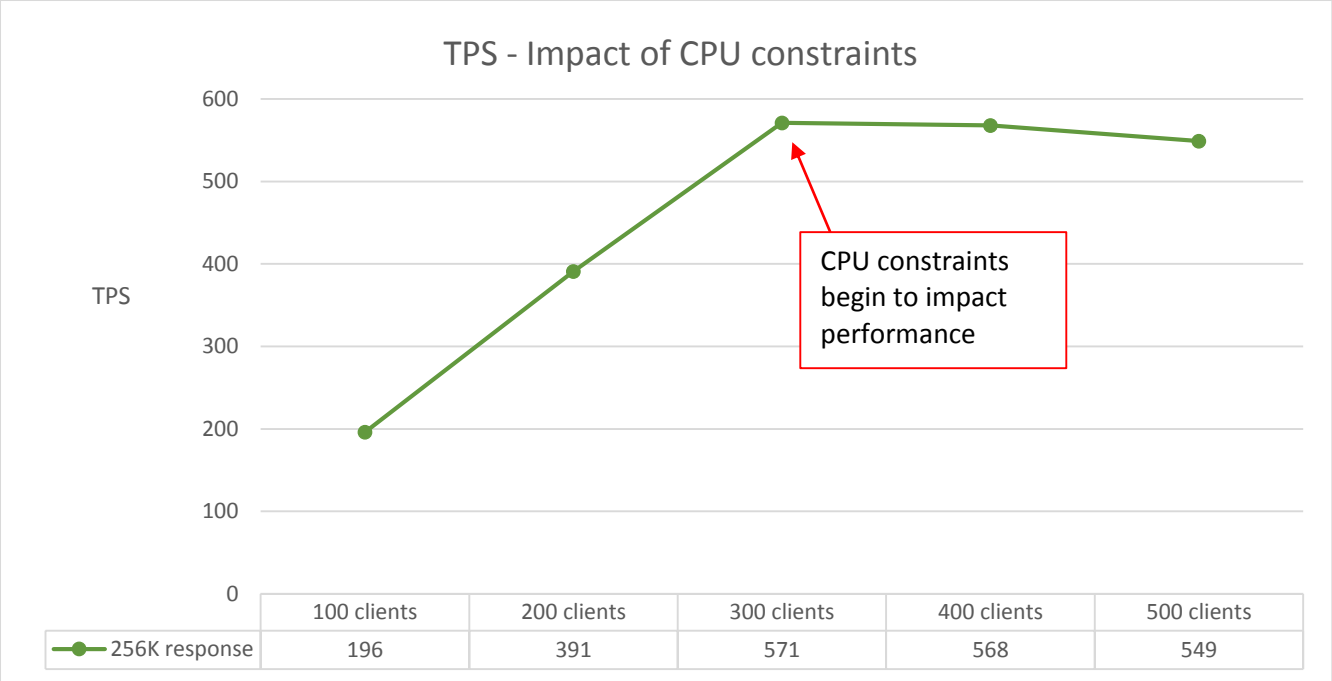


Figure 17 Impact on TPS when CPU constraints reached

12.2 Effect on Average Response

The average response is from the clients' perspective.

Figure 18 shows good scalability when 100, 200 and 300 clients are receiving 256K payload responses, but that the system ran out of CPU when running with more than 300 clients on this LPAR.

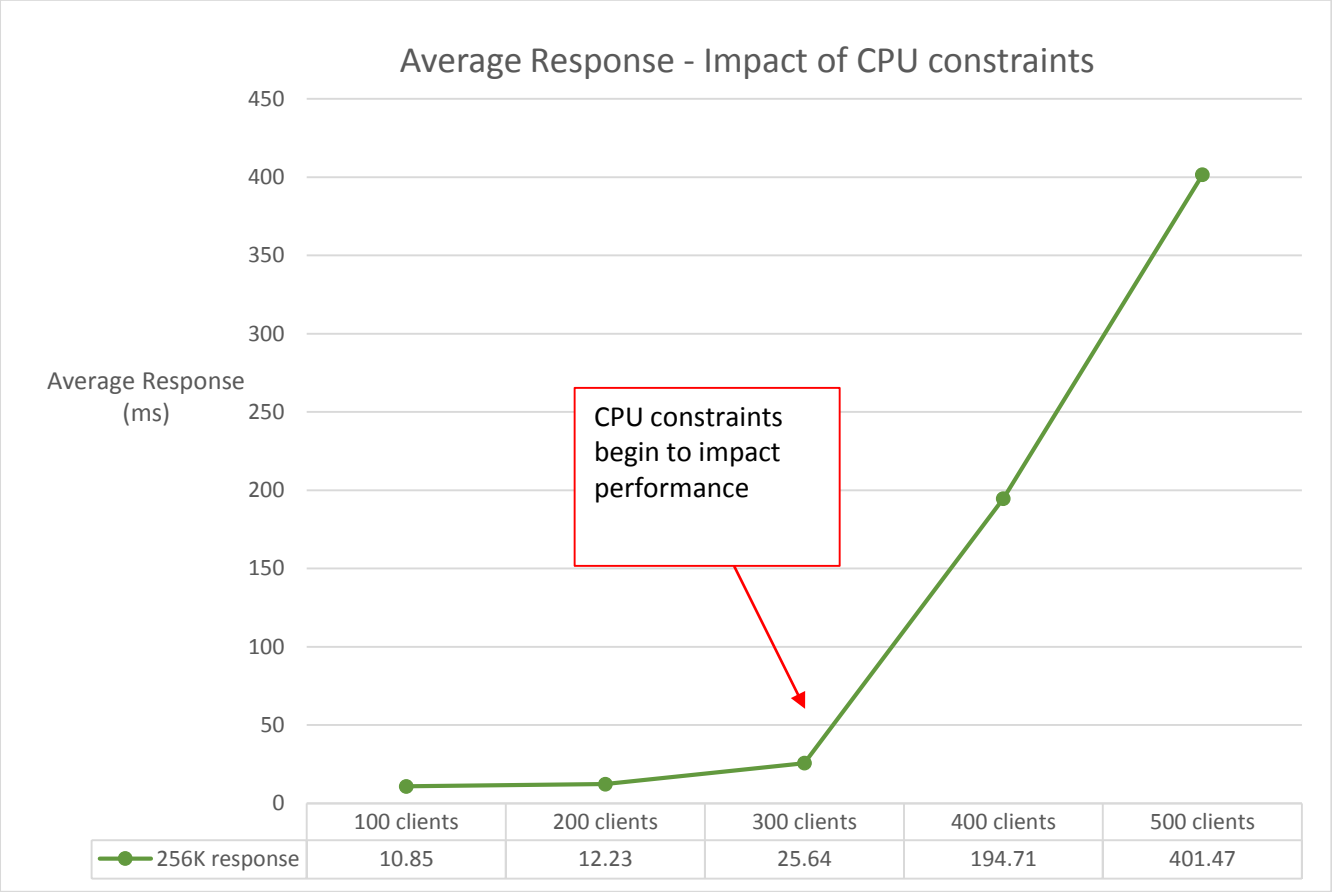


Figure 18 Impact on average response when CPU constraints reached

12.3 Effect on CPU Cost Per Transaction

Figure 19 shows good scalability when 100, 200 and 300 clients are receiving 256K payload responses, but that the system ran out of CPU when running with more than 300 clients on this LPAR.

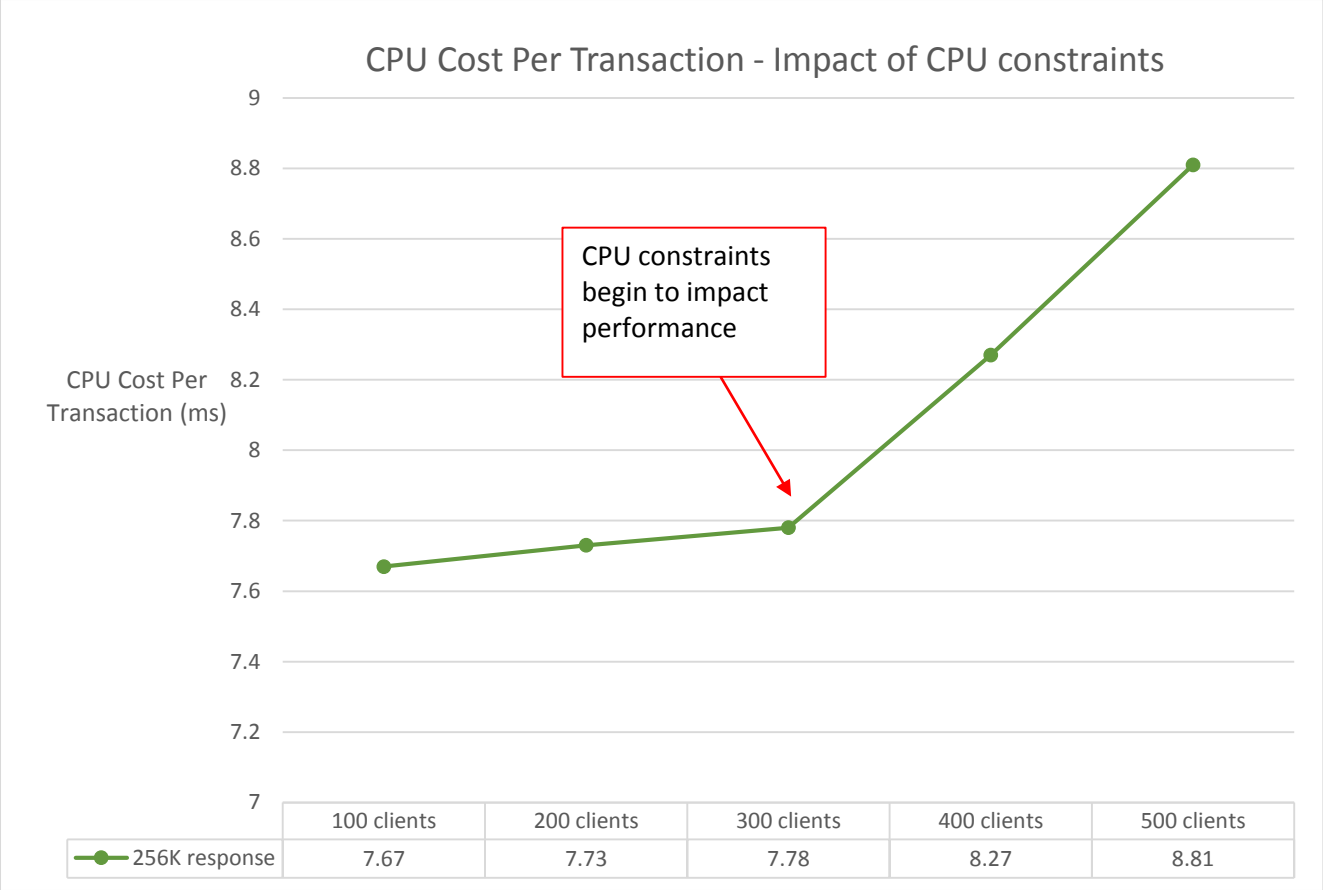


Figure 19 Impact on CPU cost per transaction when CPU constraints reached

12.4 Effect on CPU %

Figure 20 shows good scalability when 100, 200 and 300 clients are receiving 256K payload responses, but that the system ran out of CPU when running with more than 300 clients on this LPAR.

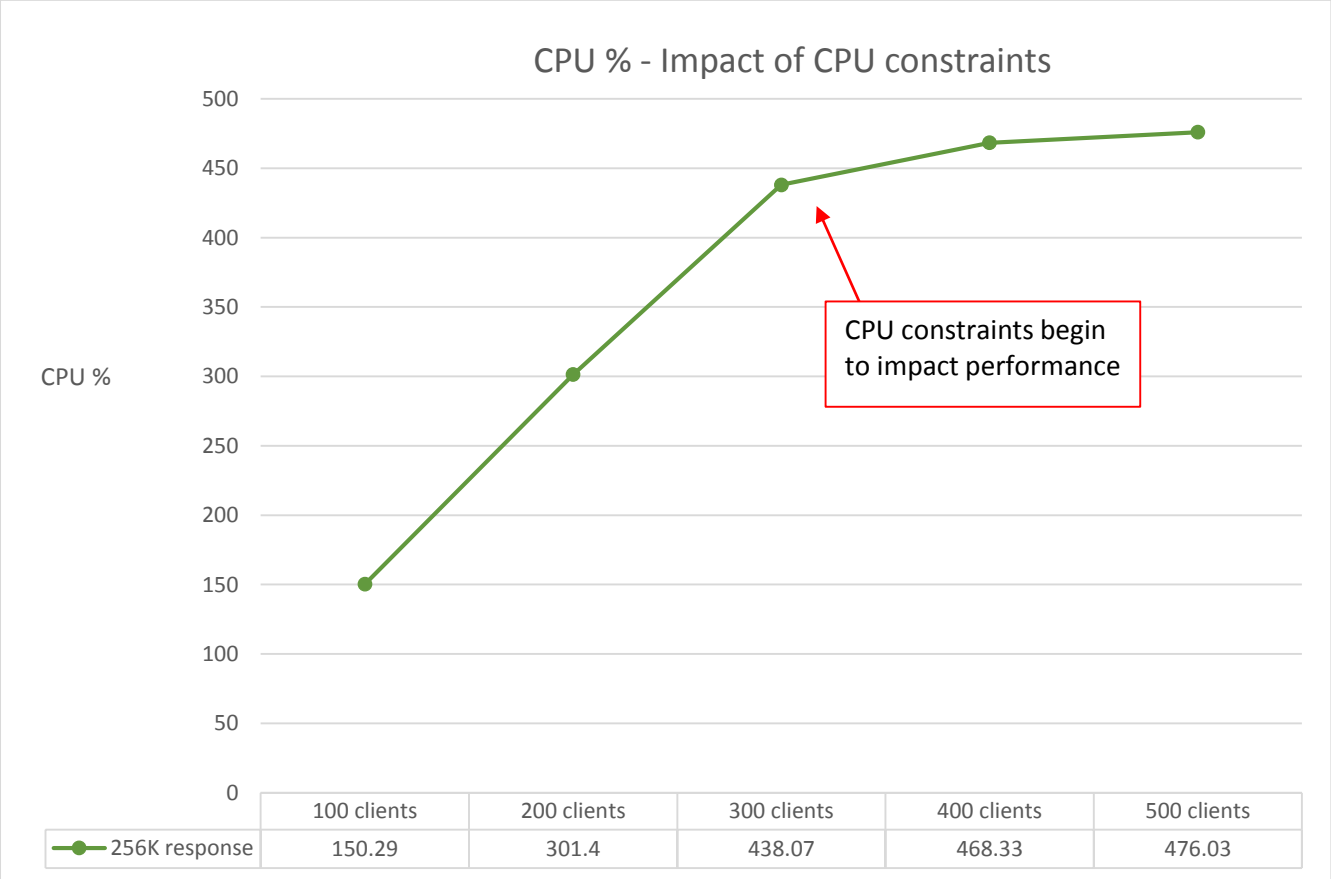


Figure 20 Impact on CPU % when CPU constraints reached

Observations

When systems become constrained, as shown in the examples above, careful monitoring is required to identify the cause.

- Use SMF data to monitor CPU usage.
- Attach the Health Center to observe garbage collection, and peaks and troughs of heap utilization.
- Understand your throughput, concurrent requests, and payload sizes.

13. Conclusions

The following conclusions can be drawn from this report:

1. A single z/OS Connect EE V3.0 server managing channel payloads to CICS Transaction Server using the CICS service provider demonstrated good scalability (in terms of number of clients and payload size) for
   - ✓ Transactions Per Second
   - ✓ Average Response time of requests
   - ✓ CPU Cost Per Transaction
   - ✓ CPU % Usage
   - ✓ Potential zIIP offload.
2. The CICS service provider scaled well.
3. z/OS Connect EE V3.0 is almost entirely written in Java and so would benefit from offloading work to one or more zIIPs.

Note:

1. The payload sizes for the responses in this report are up to 256K.
2. Analysis of other payload sizes, or different hardware, have not been completed at this time, so there is no guarantee that equivalent observations will be seen in other configurations.
3. Due to the effects on system performance of machine hardware, levels of software configuration and payload, equivalent observations might not be seen on other systems.