How to find - Queues used by SVRCONNs (MQ Client Applications)

When IBM MQ for z/OS removed the client connection licensing restrictions, the number of applications connecting directly to MQ for z/OS has been on the rise. It has led to increasing numbers of reports of CHIN address space CPU cycles going up unexpectedly, often from changes that "could not possibly have done that!" When I was an application programmer, almost every time I said that I found that indeed my change could have done that and DID do that. Now when I hear that phrase, I shudder.

Questions we often have to resolve include:
1) What channels are SvrConns?

2) What tasks are associated with client connections?

3) What queues are being used by those tasks?

4) Has the client application behavior changed?

1) What channels are server connection channels?

   Finding the SvrConns in use requires the CLASS(4) accounting data. This record is defined in the hlq.SCSQMACS(CSQDQCST) macro. The channel type is a flag in those records called QCSTCHTY. When we use MQSMFCSV that field becomes the 'TYPE' in the QCST table and the value we search for is 'SvrConn'. Getting a list of channels with the SvrConn type is very simply done when these records are available. And this gives us a list of the client connections in use, answering our first question.

   If we are fortunate to have the SMF data 'before' and 'after' the application change, this may be all that is needed to see differences. For example, one customer was complaining about a significant rise in CPU we found that the number of client connections had gone from 25 to over 100 for the same SvrConn during similar SMF intervals before and after their changes. As it turned out, the application had responsiveness issues and the owners decided they were saturating the connections. This is a typical response to issues like this, adding more serving applications, but instead of improving the response times it made their problem significantly worse. As it turns out, the work was constrained by code additions that were only supposed

to occur for about 1/100th of the workload but was actually executed about 1/3 of the time due to a programming error.

2) What tasks are associated with client connections?

   If just the number of SvrConn instances does not appear to be the problem, we look for the associated task record(s) that are for client connections. This is a bit more complicated, as there is no direct correlation between the Channel Accounting data and the Task Accounting data, we have suggested creating some kind of linkage. What we had to do was to identify the common points between the SMF Channel Accounting records and the Task ID record, hlq.SCSQMACS(CSQDWTID). The common fields include the LPAR, queue manager, channel name, and channel connection name. Joining the QCST and WTID tables gives the list of Task records associated with client connections and answers the second question.

3) What queues are associated with client connections?

   We can then use the correlation identifier in the WTID records to identify the WQ records associated with the task. The WQ records are the individual queue records associated with the task identified by this correlation identifier. Note that this identifier is also used in the WTAS records, those records hold the task-level requests and counters. We now have the answer to the third question, what queues are in use by client applications?

We'll get to question 4 at the end. Right now, I'd like to explain the process and the SQL to get to this point.

The process begins with using MQSMFCSV to format the MQ SMF records (both 115s and 116s). We then use the DDL produced by MQSMFCSV to define our tables (one per SMF record type present), load each table from the csv files created by the program, and develop and run queries against those tables. We have some 'standard' queries that we run, to tell us things like the overall API counts, give us some consolidated data between related processing tables (the QMST message manager and QIST data manager tables), recovery log data, etc. Once we have looked into these results, we are better able to decide what deeper dive issues need to be investigated.

The following is a description of the views and queries we developed to help identify why the first customer we seriously investigated this for had such a

bump in CPU consumption, when there had only been a 'minor' change to the application.

Initially we did this with some elaborate joins between the QCST data, the WTID data, and the WQ data. The initial query built was almost unreadable, even to me (and I wrote it) and took quite some time to execute, but it did work. At that point though we had the answer to question 3 – which queues are being accessed via client connections.

For the customer we initially developed this for, getting the WQ data showed that there were more clients connecting, it looked like they were using more queues, and the client application was making more requests. What was so confusing to the customer was that the volume of workload really had not risen that much. When we looked at the WQ SMF data the number of valid gets against several queues was fewer than 1% of the MQGETs issued against the queue. Unfortunately, we did not have any current 'before' data, so could not get an accurate comparison, but this low a valid MQGET rate is not what we would expect for an efficient application. I typically do not get alarmed with as few as 50% valid hit rates on MQGETs, depending on how the applications are coded. Anything below 25% and I typically would ask if that is normal and expected. When it gets to 15% I assume there is something wrong that needs to be investigated. And when it gets to single digits, I start questioning everything.

In this case, like some others, there were multiple things that happened that contributed to the rise in CPU consumption. The four top reasons include:

1) More connections (8-10 more per queue manager) from a larger group of clients. This may have been rising over time until it became an issue, we do not know.
2) More queues being accessed and/or an increase in message sizes. In some cases we have seen sizes increased to the point where Db2 BLOBs were being used for large message storage on some Shared queues (more overhead).
3) The applications began using a more aggressive polling interval to see if new messages had arrived.
4) More instances of the new applications were deployed to help with the slowdowns, which of course triggered another cycle of slowdowns.

The queries and tools

I did not keep a copy of the first query I built, as I said it was almost unreadable. After completing our report, it was time to simplify the query.

The first, and most obvious, simplification was to set up a view for all the SvrConns, it had to include the 'identifier' fields we used to match the Task records back to channel records. To make the final query more readable, we also used the column names from the WTID records for the channel and channel connection name columns in our view.

```
CREATE VIEW MQSMF.QSGA_SVRCONN2
AS
SELECT DISTINCT LPAR AS LPAR, QMGR AS QMGR, CHL_NAME AS
CHANNEL_NAME, CONNECTION_NAME AS CHANNEL_CONNECTION_NAME
FROM MQSMF.QSGQSGA_QCST AS QCST
WHERE QCST.TYPE = 'SvrConn' ;
```

This effectively gives us a list of the channels that are server connection channels for the queue managers in this queue sharing group. If we needed to examine any other type of channels (sender receiver, cluster sender and cluster receiver pairs), we can create views for those channel types as well. We then looked to improving the query that extracts the matched rows.

A few things to note about this query:

1) 'SvrConn' was inserted as a literal as the channel type for convenience.
2) This query is made against a specific Queue Sharing Group view (QSGQSGA, where QSGA is the queue sharing group name).
3) This query also uses the 'IN' clause on the QSGQSGA_SVRCONN2 view.
4) At this stage the query does not include MQ API request counts, etc. We just wanted to verify that we were seeing the same results from this query as we did from the unreadable version.

```
Select
  WTID.DATE,
  WTID.TIME,
  WTID.LPAR AS LPAR,
  WTID.QMGR AS QMGR,
  WTID.WTAS_CORRELATOR,
  APPL_TYPE,
  WTID.CONNECTION_NAME,
  WTID.CHANNEL_NAME AS CHANNEL_ID,
  'SvrConn' AS CHL_TYPE,
  WTID.CHANNEL_CONNECTION_NAME AS CHL_CONNAME,
  WQ.OPEN_NAME AS OPEN_NAME,
  WQ.BASE_NAME AS BASENAME
  FROM MQSMF.WTID AS WTID, MQSMF.WQ AS WQ
```

```
  WHERE  (   (WTID.DATE = WQ.DATE AND
    WTID.TIME = WQ.TIME AND
    WTID.LPAR = WQ.LPAR AND
    WTID.QMGR = WQ.QMGR AND
    WTID.WTAS_CORRELATOR = WQ.CORRELATION) AND
  (WTID.LPAR, WTID.QMGR,  WTID.CHANNEL_NAME,
WTID.CHANNEL_CONNECTION_NAME)
    IN
    (SELECT LPAR,QMGR, CHANNEL_NAME, CHANNEL_CONNECTION_NAME
    FROM MQSMF.QSGA_SVRCONN2))
;
```

The output from this query looks as follows (note this is very limited test data from our environment):

DATE	TIME	LPAR	QMGR	WTAS CORRELATOR	APPL TYPE	CONNECTION NAME	CHANNEL ID	CHL
10/21/2024	10:31:44,1	MQS1	ZQS1	DFE0DC8F3D69986A5AF	CHIN	ZQS1CHIN	SYSTEM.ADMIN.SVRCONN	SvrC
10/21/2024	10:32:09,6	MQS1	ZQS1	DFE0DC8F3D69986A5AF	CHIN	ZQS1CHIN	SYSTEM.ADMIN.SVRCONN	SvrC

While this sample query does not include additional important details like the number of valid MQ API requests, we did add that to our final query. In customer data, figuring out which queues are used and how they are used by SvrConns can help identify application and volume changes that the MQ Admin staff may not have known about.

So our final query looked like this:

```
Select
  WTID.DATE,
  WTID.TIME,
  WTID.LPAR AS LPAR,
  WTID.QMGR AS QMGR,
  WTID.WTAS_CORRELATOR,
  APPL_TYPE,
  WTID.CONNECTION_NAME,
  WTID.CHANNEL_NAME AS CHANNEL_ID,
  'SvrConn' AS CHL_TYPE,
  WTID.CHANNEL_CONNECTION_NAME AS CHL_CONNAME,
  WQ.OPEN_NAME AS OPEN_NAME,
  WQ.BASE_NAME AS BASENAME,
  WQ.OPEN_COUNT AS OPEN_COUNT,
  wq.CLOSE_COUNT AS CLOSE_COUNT,
  WQ.GET_COUNT AS GET_REQUESTS,
```

```
  WQ.PUT_COUNT AS PUT_REQUESTS,
  WQ.PUT1_COUNT AS PUT1_REQUESTS,
  (WQ.PUT_COUNT + WQ.PUT1_COUNT) AS TOTAL_PUT_PUT1_REQUESTS,
  WQ.INQ_COUNT AS INQ_REQUESTS,
  WQ.SET_COUNT AS SET_REQUESTS,
  WQ.TOTAL_BYTES_PUT AS TOTAL_BYTES_PUT,
  WQ.TOTAL_BYTES_GET AS TOTAL_BYTES_GET,
  WQ.TOTAL_VALID_PUTS AS TOTAL_VALID_PUTS,
  WQ.TOTAL_VALID_GETS AS TOTAL_VALID_GETS

  FROM MQSMF.WTID AS WTID, MQSMF.WQ AS WQ
  WHERE  (   (WTID.DATE = WQ.DATE AND
    WTID.TIME = WQ.TIME AND
    WTID.LPAR = WQ.LPAR AND
    WTID.QMGR = WQ.QMGR AND
    WTID.WTAS_CORRELATOR = WQ.CORRELATION) AND
  (WTID.LPAR, WTID.QMGR,  WTID.CHANNEL_NAME,
WTID.CHANNEL_CONNECTION_NAME)
    IN
    (SELECT LPAR,QMGR, CHANNEL_NAME, CHANNEL_CONNECTION_NAME
    FROM MQSMF.QSGA_SVRCONN2))
;
```

And the output looks like this (again from our small test system):

BASENAME	OPEN COUNT	CLOSE COUNT	GET REQUESTS	PUT REQUESTS	PUT1 REQUESTS	TOTAL PUT PUT1 REQUESTS	INQ REQUESTS
ELKINSC.TEST.SUB.0WILDCARD	1	0	21	0	0	0	
ELKINSC.TEST.SUB.0WILDCARD	0	1	0	0	0	0	

Question #4 - Has the client application behavior changed?

The truth is that we (the WSC) often do not know that the application or infrastructure behavior has changed because we rarely have before and after sets of data to evaluate. When we do, we feel fortunate. Many MQ Administrators are often in the same position, trying to figure out what happened after a change 'that could not possibly have had impact on MQ' that was not fully vetted.

An excellent example of this is a customer who, we believe, changed their application from using a polling interval of once per second to once every 1/10th of a second, without realizing it. We cannot absolutely prove this, because we did

not have any ‘before’ data. However, based on the SMF data captured after the rise in CPU, there were extremely low levels of valid MQGET requests – less than 1% - for queues associated with many SvrConn tasks.

Having said that, it took several working days to gather this data and process it. It then took more time to review the possible causes of this behavior and find the applications that had recently been changed. Had we had SMF data from before the application change, it would have been a much faster process.

Another example is from a customer who carefully implemented their first couple of client applications, they were designed and built to connect and stay connected, to use triggering and get wait intervals consistently and correctly. They enforced careful testing and code reviews, and while their CHIN CPU use went up, but it was both manageable and predictable. The third application was developed without the involvement or initial knowledge by the MQ Admin team and went into production, reusing the same SvrConns for their connection. Their CHIN CPU immediately skyrocketed, and of course it was MQs fault. In this case, the customer had some statistics data from prior to the new application so we were able to see that the number of channels had gone up, and the load on both the dispatcher and adapter tasks in the CHIN had increased. We could not identify the specific channels (statistics data does not report at that level), but during their 7 am to 9 pm peak there were more channels attached to the dispatcher tasks and more adapter tasks busy.

It took us less than a day after we received the before and after data to identify the increase in connections and usage.