MQ-CPH Introduction

What Is It?

The MQ-CPH Performance Harness (hereafter referred to as MQ-CPH), is a native MQI interface (C/C++), performance test tool, based largely on the function and externals of the JMSPerfHarness tool (also on GitHub at https://github.com/ot4i/perf-harness). It provides an extensive set of performance messaging functionality, as well as many other features such as throttled operation (a fixed rate and/or number of messages), multiple destinations and live performance reporting. It is one of the many tools used by the IBM MQ performance team at Hursley, for tests ranging from a single client to more than 10,000 clients.

What Types of Application Can be Simulated?

Each MQ-CPH process can start 1-n worker threads of the same type (module). The types of module that a worker that can specify are:

1. **Sender** : Sends messages to a named queue destination.
2. **Receiver** : Receives messages from a named queue destination. This can be used in conjunction with the Sender module.
3. **PutGet** : Sends a message to queue then retrieves the same message (using CorrelationId).
4. **Requestor** : Sends a message to a queue then waits for a corresponding reply (using CorrelationId) on a second queue.
5. **Responder** : Waits for a message on a queue then replies to it on another queue. This can be used in conjunction with the Requestor module.
6. **Publisher** : Sends messages to a named topic destination.
7. **Subscriber** : Subscribes and receives messages from a named topic. This can be used in conjunction with the Publisher module.

How Do I Use It?

MQ-CPH is a command line tool which can be configured in a variety of ways to run multiple MQ application worker threads, specifying persistence, transactionality, pacing etc. The simplest way to understand its use is by means of an example. Consider the following simple message flows between two MQ applications, where 'Requester' is a client bound application on a remote machine, and 'Responder' is an application using local binding (so resides on the same machine as the queue manager).

Worker module	**MQI Operation**
Requester	MQPUT to queue 'REQUEST1' (on remote host)
Responder	MQGET from queue 'REQUEST1' (on local host)

Responder	MQPUT to queue 'REPLY1', with the correlid of the message got from queue 'Reque (on local host)
Requester	MQGET (by Correlid) from queue 'REPLY1' (on remote host)

If we run the requester in a loop, then from that application's perspective, the requests are:

This is a self-regulating application, as for each requester thread, the next PUT will not be executed, until the previous reply has been received. The maximum number of messages being processed will therefore be limited by the number of requester threads started (often increased iteratively, by adding more requester processes over time, in a long test).

We can use MQ-CPH to run such a 'requester-responder' scenario, simulating multiple requesters, and multiple responders, spreading the workload across multiple pairs of request/reply queues. Diagram1, below, show a scenario with requesters and responders utilising 3 pairs of request/reply queues.

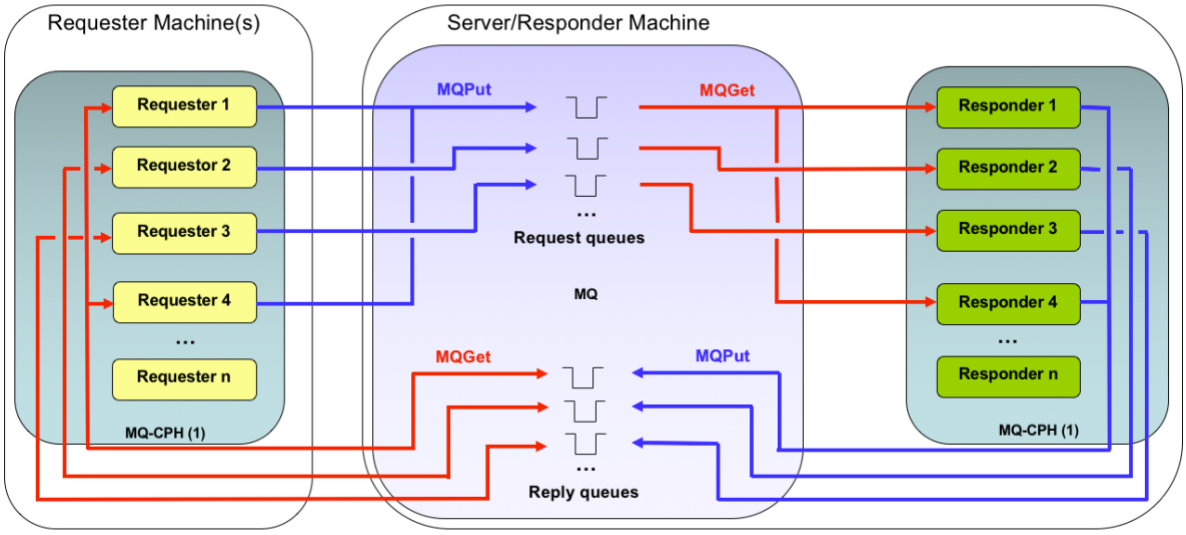


Diagram 1 : Requester/Responder Application, Utilising 3 Pairs of Request/Reply Queues.

MQ-CPH can be started with any number of requester or responder threads, and can be specified to use a single pair of request/reply queues, or a multiple set of queues (i.e. three pairs of queues, as in the diagram above). Typically, we round-robin the request and reply queues so if we were to specify 6 requesters, across three sets of queues, we would get the following:

Requester1 & requester4: PUT to 'REQUEST1' and GET from 'REPLY1'
Requester2 & requester5: PUT to 'REQUEST2' and GET from 'REPLY2'
Requester3 & requester6: PUT to 'REQUEST3' and GET from 'REPLY3'

Responders are typically configured to round-robin as well. It's important to understand that we are *opening the queues* in a round robin fashion. Once a requester, or responder starts using a particular request/reply pair of queues, it stays on that queue pair. It is for this reason, that if we specify a queue range of n, then we start a multiple of n requesters (and responders), to keep the load balanced across the queues, unless you are trying to model an unbalanced workload, of course!

In diagram 1, above, the first 4 requesters are shown, so you can see that requester4 & responder4 go back to using queues request1 & reply1, as this is the range being used, in the example shown. Any number of further requesters and/or responders can be specified however, and the range of queue pairs used can be (and typically is) larger.

Note: request/reply queues are specified as input queues/output queues respectively when used by MQ-CPH.

input queue (request queue)	: The queue that receives messages from the requester (PUT queue). Specified as responder
output queue (reply queue)	: The queue that holds reply messages for the requester (GET queue). Specified a responder

Let's Go!

Enough of the example, let's run it! The topology will be like that shown in diagram 1, and we will be specifying the following main parameters:

Queue pair range :10 (we'll use queues REQUEST1-REQUEST10 and REPLY1-REPLY10)
Number of responders: 200
Number of requesters: 100
Message Size: 2KiB
Quality of Service: Persistent/Transacted.

1. Download and make MQ-CPH

MQ-CPH can be downloaded from GitHub here: https://github.com/ibm-messaging/mq-cph
Once you have downloaded and unzipped the tool onto your Linux machine. You can build and test the tool as follows:

```
export installdir ~/cph
make
```

After which, you will have the cph executable and the required properties files in the ~/cph directory.

2. Setup the queue manager

Sample scripts to set-up the queue manager used in this tests scenario (rr_1) are in the mq-cph/samples/rr_1/qm_setup directory, previously downloaded from GitHub.

Simply review, and run the setup_mq_server.sh script there to setup the 'PERF0' queue manage used, with the required objects defined and tuning applied.

You need to have Perl installed o your system. The version this was tested on is:

Perl v5.16.3

setup_mq_server.sh may be edited, to change the location of MQ data and log files etc.

3. Start the responders.

Issue the following command to start 200 responders across 10 pairs of request/reply queues:

```
cph -nt 200 -vo 3 -wi 0 -rl 0 -id ResP1 -tx -pp -tc Responder -ss 5 -to -1
-oq REPLY -iq REQUEST -cr  -dn 10 -jb PERF0 -jt mqb
```

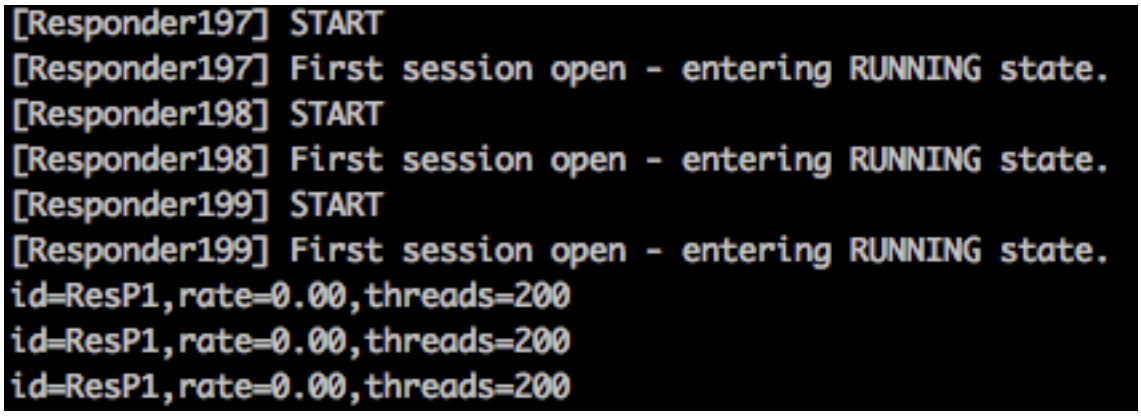
The parms used above are:

-nt	Number of worker threads
-vo	Verbosity of messages to stdout
-wi	WorkerThread start interval (ms)
-rl	Run length in seconds (0 = run forever).
-id	Process identifier, printed in statistics reporting.
-tx	Use transactions
-pp	Use persistent messages
-tc	Worker thread type (module).
-ss	Report statistics to stdout every 5 seconds
-to	Polling timeout on receiving messages. We set this to -1 to wait indefinitely (so we can start th leisure).
-oq	Get destination prefix (output/reply queue)
-iq	Put destination prefix (input/request queue)
-cr	Copy request message to response message (a different sized response message can be specifi ms instead)

-dn	Multi-destination numeric range.
-jt	Connection type (mqb = use local bindings)
-jb	MQ Queue manager.

The range of request/reply queues to use is specified by -oq & -iq (the 'output' and 'input' queue suffixes), and and -dn (the suffix range). The range can start at a different index (e.g. REQUEST11-REQUEST20), using other parameters (this can be useful when starting multiple cph processes) . We'll keep it simple here, using just -dn.

Hint: Add the -h (brief help) or -hf (full help) flags to the cph command to see an essential, or full list of parameters pertinent to the command. cph -hm <module>, will list the parameters for just one specific MQ-CPH module (e.g. cph -hm Responder will list the parameters pertinent to just the Responder module).



Screen output of Responder process, showing the startup of the last three 'Responder' worker threads, and the first three, five-second statistics intervals. The Responders are polling the request queues, waiting for work, so the total rate (across the 200 worker threads) is 0.

The 'id' specified is arbitrary but can be useful when merging results from multiple cph processes.

4. Start the Requesters

Issue the following command to start 100 requesters across 10 pairs of request/reply queues:

```
cph -vo 3 -nt 100 -ss 5 -ms 2048 -wt 10 -wi 0 -rl 0 -id ReqP1 -tx -pp -co -
tc Requester -to 30 -iq REQUEST -oq REPLY -dn 10 -jh mqhost1 -jp 1420 -jc
SYSTEM.DEF.SVRCONN -jb PERF0 -jt mqc -us <mq userid> -pw <mq userid
password>
```

Some of the parameters will be familiar from starting the requesters (e.g. -rl 0, specifying the test should run until we terminate it, and the specification of the queues to be used). Note the additional parameters here (including client connection credentials).

-ms	Message Size
-jt	Connection type (mqc = use client bindings, which additionally requires the next three parms)
-jh	Machine hosting the queue manager.
-jp	MQ Listener port on host machine.
-jc	WMQ Channel to connect to. (though the default is SYSTEM.DEF.SVRCONN anyway).
-us	Client connection userid
-pw	Client connection password

Once the requesters have started up, we have 'closed the loop' and messages will begin to flow as seen in the windows for the requester process (left), and the responder process (right):

```
[Requester97] START
[Requester97] First session open - entering RUNNING state.
[Requester98] START
[Requester98] First session open - entering RUNNING state.
[Requester99] START
[Requester99] First session open - entering RUNNING state.
id=Req1,rate=22026.80,threads=100
id=Req1,rate=22321.80,threads=100
id=Req1,rate=22205.60,threads=100
```

```
id=ResP1,rate=0.00,threads=200
id=ResP1,rate=3434.00,threads=200
id=ResP1,rate=20165.40,threads=200
id=ResP1,rate=22196.60,threads=200
id=ResP1,rate=22304.80,threads=200
id=ResP1,rate=22292.40,threads=200
```

The test here is running at a rate of around 22K 'round trips'/sec. Since each round trip involves a PUT/GET by the requester, and a GET/PUT by the responder, the total MQPUT rate serviced by the queue manager is ~44K/sec, with a similar rate for MQGET.

5. Monitor the Application

That's it, we're done! The test here is self-regulating so will run as fast as resources allow (typically, the limit will be the speed of the disk writes for a persistent test like this, but larger numbers of threads, to a point, will allow MQ to reach that disk I/O limit by combining log writes).

You can configure MQ-CPH in a multitude of ways, to compare one system with another, or look at disk capabilities, or model simple scenarios etc. Once the test is running, use your favourite tools to monitor the system (e.g. vmstat, iostat, SAR, etc).