**Performance Experiment - MQ - Actor Count and Message Size Blend**

The goal of this experiment is to benchmark how this technology performs when hardware resources are plentiful, the number of actors (publishers and subscribers) in the system varies, and the size of the message varies.

Contact [@jmbeach](https://github.com/jmbeach) or [@laynemoseley](https://github.com/laynemoseley) for clarifications.

**Preliminary Steps**

**Producer Program**

Design a **python** program utilizing your assigned MQ technology that produces as many messages per second as possible (For example: while true: produce\_message())

For MQ’s that run under multiple operating modes, run under the mode that is most reliable. An example of this would be RabbitMQ which can use “at most once” and “at least once” delivery by choosing whether or not to use acknowledgements. [Reference](https://www.rabbitmq.com/reliability.html). You should choose “at least once”.

For data collection, the program should output (to standard output) a CSV of message ID’s, the time they were sent, and the time the acknowledge (ACK) response was received (if applicable).

Ex output:

1,1617516000,1617516005

2,1617516010,1617516015

...

**Note:** The reason we are using standard out is to prevent using file operations that could slow down the overall performance.

The program should send 1,000 messages and then terminate once all messages are sent and all acknowledgements are received.

Link to your program’s usage here:

[Replace me](Replace me)

**Consumer Program**

Design a **python** program utilizing your assigned MQ technology that consumes as many messages per second as possible (For example: while true: consume\_message())

For MQ’s that run under multiple operating modes, run under the mode that is most reliable. An example of this would be RabbitMQ which can use “at most once” and “at least once” delivery by choosing whether or not to use acknowledgements. [Reference](https://www.rabbitmq.com/reliability.html). You should choose “at least once”.

The consumer doesn’t have to do anything with the message; only retrieve it.

The program should terminate once all messages are consumed.

For data collection, the program should output (to standard output) a CSV of message ID’s, the time they were received, and the time they were acknowledged (if applicable).

Ex output:

1,1617516000,1617516005

2,1617516010,1617516015

...

Link to your program’s usage here:

[Replace me](Replace me)

**Data Processor**

The experiment designers will provide a program that accepts a list of producer csv files and a list of consumer csv files. It should output 2 CSV’s that marry up the matching ID’s of messages and contain calculated results.

The first csv should contain the following durations:

* Processing latency - total duration in milliseconds from the time the message is sent to the time the message is received
* Send time - total duration in milliseconds from the time the message is sent to the time the message arrives at the broker/queue/system (this is likely the ACK received time)

Ex:

1,15,8

2,8,4

...

The above output indicates that message 1 took 15ms to be fully processed and 8ms to be sent to the broker.

The second csv should output the number of messages sent in each second of the experiment and the number of messages received in each second of the experiment. This is the throughput.

1,15,30

2,8,16

...

The above output indicates that the end to end throughput in second 1 was 15 and the send to broker throughput in second 1 was 16.

Link to program pending.

**Hardware Configuration**

Each test should be ran using Ubuntu 18.04 chameleon cloud VM’s with 8 CPU and 32 GB of RAM. If these hardware specs aren’t attainable, use the nearest available and document the configuration.

**Experiment Configurations**

These are the configurations your program should be ran in.

Refer to [MessageSize](https://github.com/CS-6381/FinalProject/blob/main/DesignOfExperiments/MQ-MessageSize.md) for the content of the messages to be sent.

| **Producer Machines** | **Producer Instances** | **Consumer Machines** | **Consumer Instances** | **Message Size** |
| --- | --- | --- | --- | --- |
| 1 | 1 | 1 | 1 | tiny |
| 1 | 1 | 1 | 5 | small |
| 1 | 5 | 1 | 1 | medium |
| 3 | 8 | 3 | 8 | large |
| 4 | 25 | 4 | 25 | x-large |

If possible, implement these configurations with redundancy enabled and document the configuration.

These tests are described in detail below and have placeholders for results.

**Redundancy implementation Details**

Place the redundancy implementation details here.

**1 Producer - 1 Consumer - Tiny Message Size**

Run one instance of your producer program on one VM. Configure your producer to always send the [tiny](https://github.com/CS-6381/FinalProject/blob/main/DesignOfExperiments/messages/tiny.txt) message.

Ensure you run your programs in such a way that the output is not lost. For example: python main.py > results.csv.

Run your consumer program with a matching configuration.

Fill this table with the appropriate links:

| **File Name** | **Link** |
| --- | --- |
| producer1.csv |  |
| consumer1.csv |  |
| latencies.csv |  |
| throughput.csv |  |

Calculate the following data using Excel or a custom program by analyzing the data in latencies.csv and throughput.csv

| **Metric** | **Value** |
| --- | --- |
| Processing Latency Min |  |
| Processing Latency Max |  |
| Processing Latency Average |  |
| Processing Latency Standard Deviation |  |
| Send Time Min |  |
| Send Time Max |  |
| Send Time Average |  |
| Send Time Standard Deviation |  |
| Processing Throughput Min |  |
| Processing Throughput Max |  |
| Processing Throughput Average |  |
| Processing Throughput Standard Deviation |  |
| Send Throughput Min |  |
| Send Throughput Max |  |
| Send Throughput Average |  |
| Send Throughput Standard Deviation |  |

**1 Producer VM - 1 Consumer VM: 5 Instances - Small Message Size**

Run one instance of your producer program on one VM. Configure your producer to always send the [small](https://github.com/CS-6381/FinalProject/blob/main/DesignOfExperiments/messages/small.txt) message.

Ensure you run your programs in such a way that the output is not lost. For example: python main.py > results.csv.

Run five consumer program instances on one VM.

Fill this table with the appropriate links:

| **File Name** | **Link** |
| --- | --- |
| producer1.csv |  |
| consumer1.csv |  |
| latencies.csv |  |
| throughput.csv |  |

Calculate the following data using Excel or a custom program by analyzing the data in latencies.csv and throughput.csv

| **Metric** | **Value** |
| --- | --- |
| Processing Latency Min |  |
| Processing Latency Max |  |
| Processing Latency Average |  |
| Processing Latency Standard Deviation |  |
| Send Time Min |  |
| Send Time Max |  |
| Send Time Average |  |
| Send Time Standard Deviation |  |
| Processing Throughput Min |  |
| Processing Throughput Max |  |
| Processing Throughput Average |  |
| Processing Throughput Standard Deviation |  |
| Send Throughput Min |  |
| Send Throughput Max |  |
| Send Throughput Average |  |
| Send Throughput Standard Deviation |  |

**1 Producer VM: 5 Instances - 1 Consumer VM: 1 Instance - Medium Message Size**

Run five producer program instances on one VM. Configure the producers to always send the [medium](https://github.com/CS-6381/FinalProject/blob/main/DesignOfExperiments/messages/medium.txt) message.

Ensure you run your programs in such a way that the output is not lost. For example: python main.py > results.csv.

Run one instance of the consumer program on one VM.

Fill this table with the appropriate links:

| **File Name** | **Link** |
| --- | --- |
| producer1.csv |  |
| consumer1.csv |  |
| latencies.csv |  |
| throughput.csv |  |

Calculate the following data using Excel or a custom program by analyzing the data in latencies.csv and throughput.csv

| **Metric** | **Value** |
| --- | --- |
| Processing Latency Min |  |
| Processing Latency Max |  |
| Processing Latency Average |  |
| Processing Latency Standard Deviation |  |
| Send Time Min |  |
| Send Time Max |  |
| Send Time Average |  |
| Send Time Standard Deviation |  |
| Processing Throughput Min |  |
| Processing Throughput Max |  |
| Processing Throughput Average |  |
| Processing Throughput Standard Deviation |  |
| Send Throughput Min |  |
| Send Throughput Max |  |
| Send Throughput Average |  |
| Send Throughput Standard Deviation |  |

**3 Producer VM's: 8 Instances - 3 Consumers VM's: 8 Instances - Large Message Size**

Run eight instances of your producer on three different machines (24 instances total). Configure your producers to always send the [large](https://github.com/CS-6381/FinalProject/blob/main/DesignOfExperiments/messages/large.txt) message.

Ensure you run your programs in such a way that the output is not lost. For example: python main.py > results.csv.

Run eight instances of your consumer program on three different machines as well.

Fill this table with the appropriate links:

| **File Name** | **Link** |
| --- | --- |
| producer1.csv |  |
| producer2.csv |  |
| producer3.csv |  |
| consumer1.csv |  |
| consumer2.csv |  |
| consumer3.csv |  |
| latencies.csv |  |
| throughput.csv |  |

Calculate the following data using Excel or a custom program by analyzing the data in latencies.csv and throughput.csv

| **Metric** | **Value** |
| --- | --- |
| Processing Latency Min |  |
| Processing Latency Max |  |
| Processing Latency Average |  |
| Processing Latency Standard Deviation |  |
| Send Time Min |  |
| Send Time Max |  |
| Send Time Average |  |
| Send Time Standard Deviation |  |
| Processing Throughput Min |  |
| Processing Throughput Max |  |
| Processing Throughput Average |  |
| Processing Throughput Standard Deviation |  |
| Send Throughput Min |  |
| Send Throughput Max |  |
| Send Throughput Average |  |
| Send Throughput Standard Deviation |  |

**4 Producer VM's: 25 Threads - 4 Consumer VM's: 25 Threads - X-Large Message Size**

Run 25 instances of the producer program on four different VM's (100 instances total). Configure your producers to always send the [x-large](https://github.com/CS-6381/FinalProject/blob/main/DesignOfExperiments/messages/xlarge.txt) message.

Ensure you run your programs in such a way that the output is not lost. For example: python main.py > results.csv.

Run 25 instances of the consumer program on four different VM's (100 instances total).

Fill this table with the appropriate links:

| **File Name** | **Link** |
| --- | --- |
| producer1.csv |  |
| producer2.csv |  |
| producer3.csv |  |
| producer4.csv |  |
| consumer1.csv |  |
| consumer2.csv |  |
| consumer3.csv |  |
| consumer4.csv |  |
| latencies.csv |  |
| throughput.csv |  |

Calculate the following data using Excel or a custom program by analyzing the data in latencies.csv and throughput.csv

| **Metric** | **Value** |
| --- | --- |
| Processing Latency Min |  |
| Processing Latency Max |  |
| Processing Latency Average |  |
| Processing Latency Standard Deviation |  |
| Send Time Min |  |
| Send Time Max |  |
| Send Time Average |  |
| Send Time Standard Deviation |  |
| Processing Throughput Min |  |
| Processing Throughput Max |  |
| Processing Throughput Average |  |
| Processing Throughput Standard Deviation |  |
| Send Throughput Min |  |
| Send Throughput Max |  |
| Send Throughput Average |  |
| Send Throughput Standard Deviation |  |