Министерство науки и высшего образования Российской Федерации

Федеральное государственное бюджетное образовательное учреждение
высшего образования «Петрозаводский государственный университет»

Институт математики и информационных технологий
кафедра информатики и математического обеспечения

Оценивание производительности брокера сообщений EMQ X

Харзия Н., Беседный Н., Филиппова Е.

Петрозаводск — 2020

Содержание

1 Введение

MQTT - это упрощенный сетевой протокол обмена данными, созданный для передачи данных на удаленных локациях, где требуется небольшой размер кода и есть ограничения по пропускной способности канала. Протокол работает поверх TCP/IP, и ориентирован на обмен сообщений по принципу издатель-подписчик.

Брокер EMQ X - это полностью масштабируемый, распределенный брокер обмена сообщениями MQTT с открытым исходным кодом, который может выдерживать до миллиона параллельных подключенных клиентов.

Цель: проверить брокер сообщений EMQ X на соответствие требованиям в системе промышленного мониторинга.

Задачи:

1. Зафиксировать целевую платформу;
2. Зафиксировать требования(ограничения) к брокеру сообщений EMQX;
3. Определить тестируемые метрики и план тестирования;
4. Развернуть систему на целевой платформе;
5. Подготовить ПО для сбора данных о работе системы;
6. Провести расчеты и сделать вывод.

2 Архитектура тестируемой системы

Программные агенты связываются с датчиками один к одному, их количество зависит напрямую от количества датчиков в системе. Кол-во сервисы умного мониторинга зависит от набора предоставляемых системой функций, но не более 30 шт.

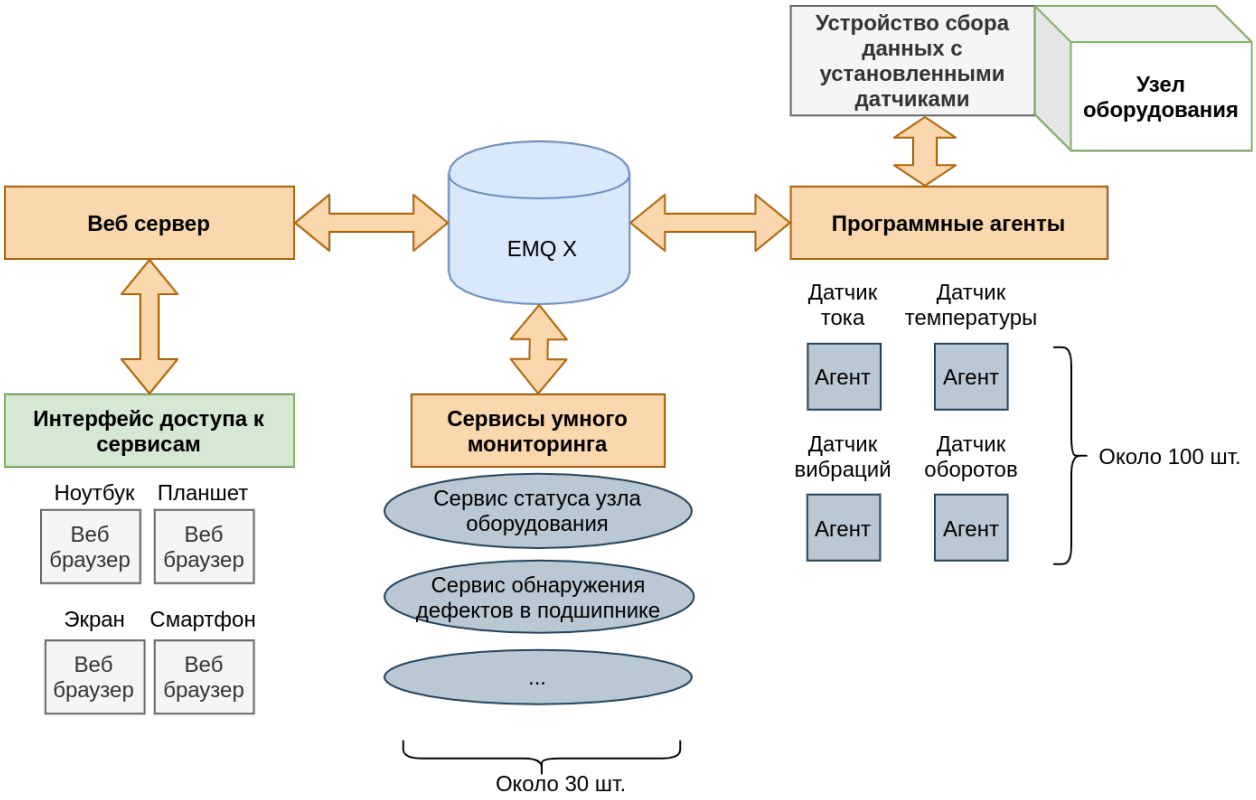

Рис. 1: Архитектура тестируемой системы

3 Оценка количества сообщений отправляемых брокеру

Программные агенты:

- Раз в секунду отправляет 1 сообщение
- Когда происходит событие – отправляет 1 сообщение

Сервисы умного мониторинга:

- Когда происходит событие – отправляет 1 сообщение

Таблица 1: Оценка количества сообщений программных агентов в системе

Кол-во программных агентов	10	20	50	70	100
Кол-во сообщений в секунду	от 10 до 20	от 20 до 40	от 50 до 100	от 70 до 140	от 100 до 200
Итого сообщений в секунду	20	40	100	140	200

Таблица 2: Оценка количества сообщений сервисов в системе

Кол-во сервисов	30
Кол-во сообщений в секунду	не более 30
Итого сообщений в секунду	30

Целевой платформой является Lenovo ThinkPad T540p со следующими характеристиками:

1. i7 4710MQ
2. 16 Гб RAM
3. 1 Тб HDD
4. ОС Ubuntu 20.04

4 Метрики проекта

1. Потребление CPU

   Поскольку брокер является лишь вспомогательным инструментом для основных вычислительных систем, нагрузка брокера на процессор должна составлять менее 50

2. Потребление оперативной памяти

   Поскольку брокер является лишь вспомогательным инструментом для основных вычислительных систем, потребление оперативной памяти должно быть менее 1.5 Гигабайт.

3. Количество недоставленных сообщений

   Все сервисы системы общаются через брокер. Существуют как сообщения, отправляемые через фиксированный промежуток времени, так и сообщения которые могут быть отправлены в любой момент. Недоставленное сообщение может сильно повлиять на точность работы системы.

4. Среднее время доставки сообщения

   Сервисам, для которых важно время выполнения алгоритма, требуется минимальное время отправки/получения сообщений от других сервисов через брокер.

5 План тестирования

Подсчет будет производится независимо для 10, 20, 50, 70 и 100 программных агентов по следующему сценарию:

1. Подготовка инструментов сбора данных: клиенты (аналоги реальных модулей), системные мониторы CPU и RAM;
2. Запуск брокера EMQ X;
3. Запуск множества клиентов имитирующих работу агентов + 30 сервисов;
4. Испытание производительности в соответствии с планом тестирования;
5. Выводы на основе собранных данных.

Тестирование метрик:

1. Потребление CPU
   - Запуск системного монитора top/htop для мониторинга ресурсов CPU
2. Потребление оперативной памяти

- Запуск системного монитора top/htop для мониторинга ресурсов CPU

3. Количество недоставленных сообщений

   - Каждый клиент ведет счетчик отправленных и полученных сообщений, счетчики обновляются и записываются в файл созданный клиентом.
   - Каждое сообщение содержит поле с временной меткой, в которой содержится время отправки в формате Unix-time.
   - Получатель сообщения вычисляет текущее время в формате Unix-time и считает разницу между текущим временем и временем в полученном сообщении.
   - Полученную разницу времени назовем “Время доставки сообщения”.
   - Если время доставки сообщения больше 1 секунды, то сообщение считается недоставленным, и счетчик полученных сообщений не обновляется.
   - Время доставки накапливается в одной переменной для всех полученных сообщений и обновляется в файле.

4. Среднее время доставки сообщения

   - Каждый клиент ведет счетчик отправленных и полученных сообщений, счетчики обновляются и записываются в файл созданный клиентом.
   - Каждое сообщение содержит поле с временной меткой, в которой содержится время отправки в формате Unix-time.
   - Получатель сообщения вычисляет текущее время в формате Unix-time и считает разницу между текущим временем и временем в полученном сообщении.
   - Полученную разницу времени назовем “Время доставки сообщения”.

- Если время доставки сообщения больше 1 секунды, то сообщение считается недоставленным, и счетчик полученных сообщений не обновляется.
- Время доставки накапливается в одной переменной для всех полученных сообщений и обновляется в файле.

6 Проведение тестирования

Для тестирования на целевую платформу устанавливался брокер EMQ X. Отправление сообщений брокеру производилось с других устройств через локальную сеть. Тестирование производилось скриптами, написанными на python с библиотекой Paho MQTT. Для отслеживания CPU и RAM использовалась утилита psrecord позволяющая отрисовывать график потребления CPU и RAM на протяжении времени.

6.1 Publisher.py

Скрипт отвечающий за отправку сообщений брокеру.

```
import random
import time
import numpy as np
import sys
from paho.mqtt import client as mqtt_client


broker = '172.17.172.10'
port = 1883
topic = "/"
if (len(sys.argv) > 2):
    topic = sys.argv[2]
# generate client ID with pub prefix randomly
client_id = f'python-mqtt-{random.randint(0, 1000)}'
```

```
def connect_mqtt():
    def on_connect(client, userdata, flags, rc):
        if rc == 0:
            print("Connected to MQTT Broker!")
        else:
            print("Failed to connect, return code %d\n", rc)

    client = mqtt_client.Client(client_id)
    client.on_connect = on_connect
    client.connect(broker, port)
    return client

def publish(client):
    msg_count = 0
    while True:
        time.sleep(1)
        timestamp = int(time.time())
       msg = "{\"m\":" + str(msg_count) + ", \"t\":" + str(timestamp) + "}"
        result = client.publish(topic, msg)
        # result: [0, 1]
        status = result[0]
        if status == 0:
            a = 0
        else:
            print(f"Failed to send message to topic {topic}")
        msg_count += 1

        p = np.random.sample()

        if (p < 0.20):
          msg = "{\"m\":" + str(msg_count) + ", \"t\":" + str(timestamp) + "}"
```

```
            result = client.publish(topic, msg)
            status = result[0]
            if status == 0:
                a = 0
            else:
                print(f"Failed to send message to topic {topic}")
            msg_count += 1
    f = open("/home/demo/experiments/pub/pub_" + sys.argv[1] + ".txt", 'w')
    f.write(str(msg_count))
    f.close()

def run():
    client = connect_mqtt()
    if len(sys.argv) > 2:
        client.loop_start()
        publish(client)
    else:
        print("Добавьте уникальный номер клиента")

if __name__ == '__main__':
    run()
```

6.2 Subscriber.py

Скрипт отвечающий за подписку к брокеру и получение сообщений.

```
 import random
import time
import json
import sys
from paho.mqtt import client as mqtt_client
```

```
broker = '172.17.172.10'
port = 1883
topic = "/"
if (len(sys.argv) > 2):
    topic = sys.argv[2]
# generate client ID with pub prefix randomly
client_id = f'python-mqtt-{random.randint(0, 100)}'
recv_msg_count = 0
delivery_time = 0

def connect_mqtt() -> mqtt_client:
    def on_connect(client, userdata, flags, rc):
        if rc == 0:
            print("Connected to MQTT Broker!")
        else:
            print("Failed to connect, return code %d\n", rc)

    client = mqtt_client.Client(client_id)
    client.on_connect = on_connect
    client.connect(broker, port)
    return client



def subscribe(client: mqtt_client):
    def on_message(client, userdata, msg):
            parsed_msg = json.loads(msg.payload)
            curr_t = int(time.time())
            global recv_msg_count
            t = parsed_msg["t"]
            global delivery_time
```

```
            if (curr_t - t <= 1):
                delivery_time += curr_t - t
                recv_msg_count += 1
             f = open("/home/demo/experiments/sub/sub_" + sys.argv[1] + ".txt", 'w')
                f.write(str(recv_msg_count) + " " + str(delivery_time))
                f.close()

    client.subscribe(topic)
    client.on_message = on_message

def run():
    client = connect_mqtt()
    subscribe(client)
    if len(sys.argv) > 2:
        client.loop_forever()
    else:
        print("Добавьте уникальный номер клиента")

if __name__ == '__main__':
    run()
```

6.3 Read_files.py

Скрипт отвечающий за подсчет результатов.

```
 import os
import numpy as np
import matplotlib.pyplot as plt

send_amount = 0
get_amount = 0
time_amount = 0
```

```
all_files = os.listdir("/home/demo/experiments/pub/")
pub_amount = len(all_files)
for i in all_files:
  my_data = np.genfromtxt('/home/demo/experiments/pub/'+i)
  send_amount+=my_data
all_files = os.listdir("/home/demo/experiments/sub/")
sub_amount = len(all_files)
for i in all_files:
  my_data = np.genfromtxt('/home/demo/experiments/sub/'+i)
  get_amount+=my_data[0]
  time_amount+=my_data[1]
print("Кол-во отправителей: "+str(pub_amount)+
    "\nКол-во получателей: "+str(sub_amount)+
    "\nКол-во отправленных сообщений: "+str(send_amount)+
    "\nКол-во полученных сообщений: "+str(get_amount)+
   "\nСреднее время доставки: "+str(time_amount/get_amount/sub_amount)+
    "\nКол-во недоставленных сообщений: "
    +str(int((1 - get_amount/sub_amount/send_amount)*100))+"%\n")
```

6.4 Run_test.sh

Скрипт отвечающий за запуск сервера.

```
#!/bin/bash

if [ -n "$1" ] && [ -n "$2" ]
then
echo "Запуск теста для $1 публикаторов и $2 подписчиков."
else
echo "No parameters found. "
fi

rm -rf /home/demo/experiments/pub/* /home/demo/experiments/sub/*
```

```
for (( i=1; i <= $2; i++ ))
do
    echo "start sub $i"
    python3 subscriber.py $i "service/events" &
done

for (( i=1; i <= $1; i++ ))
do
    echo "start pub $i"
    python3 publisher.py $i "service/events" &
done

sleep 30

pids=$(ps aux | grep -E "python3 publisher" | grep -v "grep" | grep -v
"awk" | awk  '{print $2}')
echo $pids
kill -15 $pids

pids=$(ps aux | grep -E "python3 subscriber" | grep -v "grep" | grep -v
"awk" | awk  '{print $2}')
echo $pids
kill -15 $pids

python3 read_files.py
```

7 Результаты

7.1 Запуск 40 клиентов

Среднее время доставки сообщения < 0.03 и % недоставленных сообщений 0. Использование CPU и RAM на рисунке 2. % CPU измеряется аналогично программе top: 100% на 1 ядро

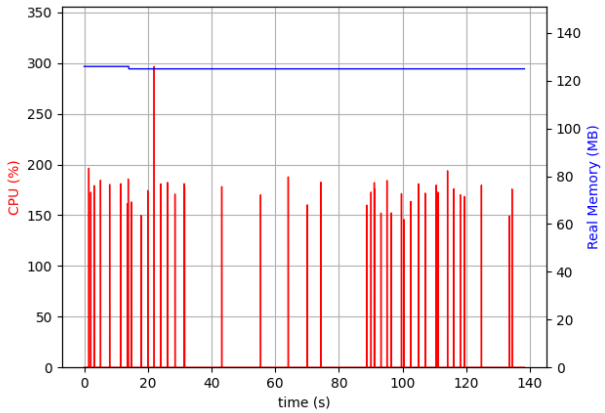

Рис. 2: % CPU и RAM

7.2 Запуск 50 клиентов

Среднее время доставки сообщения < 0.14 и % недоставленных сообщений 0. Использование CPU и RAM на рисунке 3.

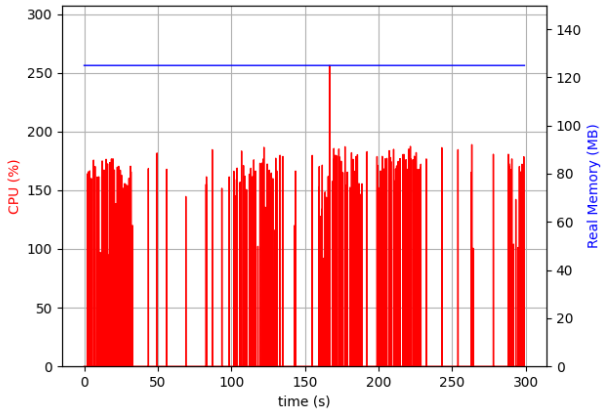

Рис. 3: % CPU и RAM

7.3 Запуск 80 клиентов

Среднее время доставки сообщения < 0.18 и % недоставленных сообщений 0. Использование CPU и RAM на рисунке 4.

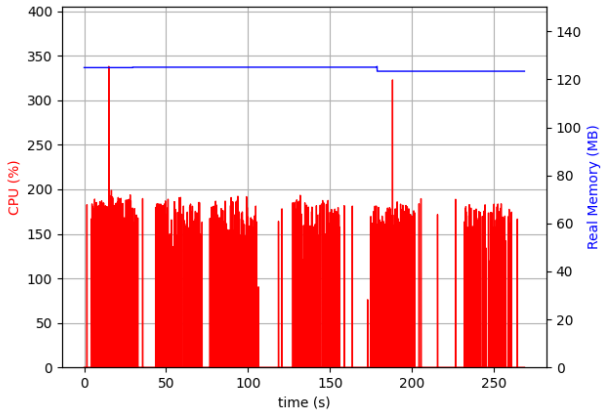

Рис. 4: % CPU и RAM

7.4 Запуск 100 клиентов

Среднее время доставки сообщения < 0.20 и % недоставленных сообщений 0. Использование CPU и RAM на рисунке 5.

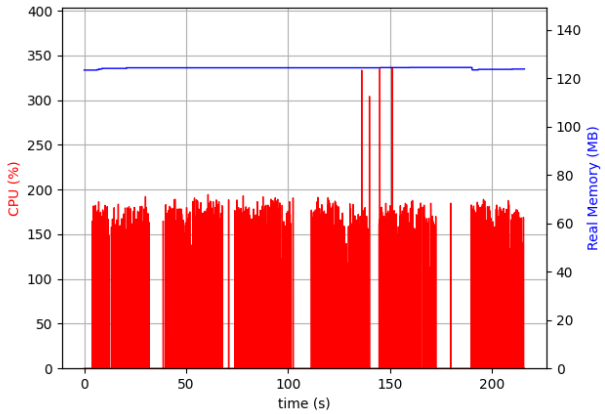

Рис. 5: % CPU и RAM

7.5 Запуск 130 клиентов

Среднее время доставки сообщения < 0.20 и % недоставленных сообщений 0. Использование CPU и RAM на рисунке 6.

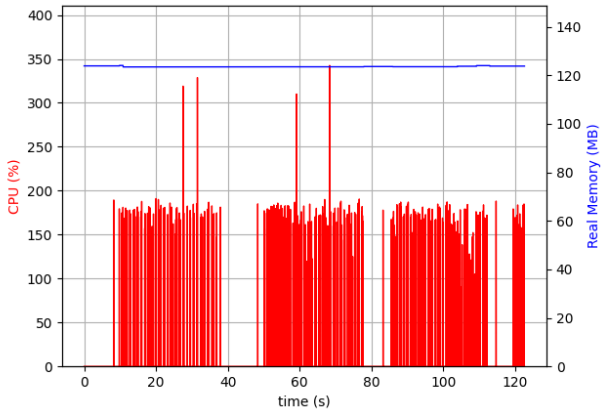

Рис. 6: % CPU и RAM

Вывод

При тестировании брокера сообщений с разным количеством клиентов от 10 до 100, отслеживаемые метрики находились в допустимых пределах. Брокер удовлетворил всем заявленным требованиям.