PAUL SCHERRER INSTITUT

PSI

Michele Brambilla :: Scientific Software Developer :: Paul Scherrer Institut

Experiences with ØMQ

BrightnESS meeting, Lund

PAUL SCHERRER INSTITUT

What ØMQ is

Message library primary mean to scale that acts as concurrency framework

From the point of view off networking:

- broker free
- various transport process: in-process, inter-process,TCP, multicast
- N-to-N connections
- different socket pattern: pub-sub, request-reply,…
- asynchronous I/O

From the point of view of programming:

- many languages supported: C, C++, C#, Python, Java, Tcl (Clojure, Delphi, Erlang, Go, Haskell, Lisp, Obj-C, PHP, Ruby,…)
- open source
- fast to learn, fast to use
- huge online documentation

PAUL SCHERRER INSTITUT

ØMQ features

- Messages are blobs of 0 to N bytes
- No difference between text (attention!) and raw data
- Messages can consist of multiple parts
- Message queues at sender and receiver
- One socket can connect to many socket: receiver can filter messages
- Automatic TCP reconnect (I will discuss with SINQhm)
- Zero copy for large messages (increase performances)

```
import zmq

# step 1: create context
context = zmq.Context()

#step 2: create socket & connect
socket = context.socket(zmq.PUB)
socket.bind("tcp://*:5557")

#step 3: send
msg = "Hello"
socket.send(msg)
```

```
import zmq

# step 1: create context
context = zmq.Context()

#step 2: create socket & connect
socket = context.socket(zmq.SUB)
socket.connect("tcp://127.0.0.1:5557")

#step3: receive
msg = socket.recv()
```

PAUL SCHERRER INSTITUT

ØMQ features

- Messages are blobs of 0 to N bytes
- No difference between text (attention!) and raw data
- Messages can consist of multiple parts
- Message queues at sender and receiver
- One socket can connect to many socket: receiver can filter messages
- Automatic TCP reconnect (I will discuss with SINQhm)
- Zero copy for large messages (increase performances)

```
# nonblocking send & recv
socket.send(msg,flag=NOBLOCK)
socket.recv(msg,flag=NOBLOCK)

# multipart message
socket.send(msg0,flag=SNDMORE)
socket.send(msg1)

while more:
    msg.append(socket.recv())
    more = socket.getsockopt(zmq.RCVMORE)
```

```
# limit number of messages that can be buffered
socket.setsockopt(zmq.HWM, 1000)

# set a filter for incoming messages
topicfilter = "10001"
socket.setsockopt(zmq.SUBSCRIBE, topicfilter)

# set a timeout for the receiver
socket.setsockopt(zmq.RCVTIMEO, timeout)

# do not copy message in local buffer
socket.send(msg,copy=False)
socket.recv(msg,copy=False)
```

PAUL SCHERRER INSTITUT

Basic message patterns

ØMQ provides sockets which enables advantage of message patterns

- PAIR
- PUSH - PULL
- REQ - REP
- PUB - SUB

Each pattern defines constrain on the network topology

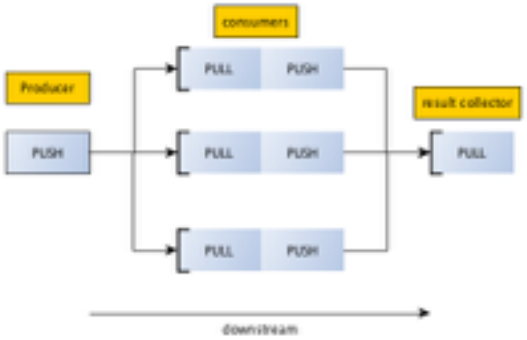
Failures

- Server dies
- With many connect/disconnect applications can leak memory and get slower
- Is network silence “good” or “bad”?
- If TCP connection stays silent for a while, network can disconnect

Techniques for reliability

PUSH-PULL

- Distribute messages between multiple workers in a pipeline
- One directional connection
- Round-robin distributor



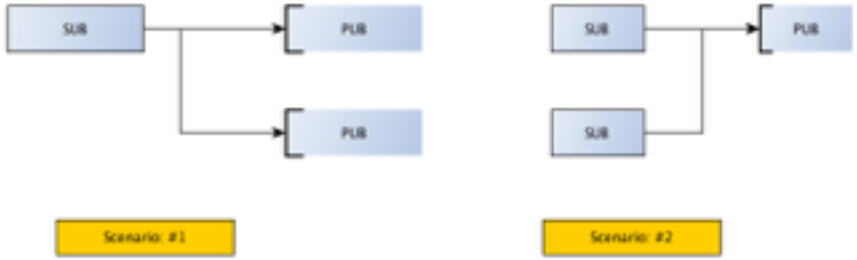
- Producer and collector stable part of architecture, consumers dynamic part

Possible issues

- If one consumer connects faster will receive more messages
- Large tasks requiring time to complete can cause unbalance
- If a consumer dies (while working) producer doesn’t know

PUB-SUB

- Addresses the “group messaging” problem
- Aimed for scalability: large volumes data sent rapidly to many recipients
- One-directional communications: no possibility to coordinate senders and receivers



- Messages are distributed without the knowledge of what or if any subscriber exist: a publisher with no connected subscribers drops the packages
- Filtering happens at the subscriber side

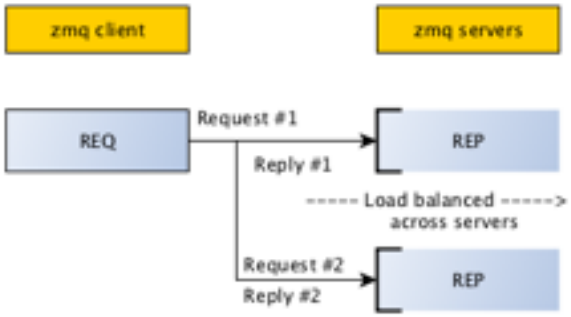
Possible issues

- Subs join late or drop off: messages are lost
- Subs slow fetching messages: queues overflow, pubs crash
- Network become overloaded and drop data

PAUL SCHERRER INSTITUT

REQ-REP

- Distribute messages between multiple workers
- Client-server model
- Synchronous request-reply dialog



Possible issues

- Server dies: client hangs forever
- Network loses request or reply: client hangs forever

PAIR

- 1-to-1 bidirectional connection
- No specific state stored within socket
- Sever listen (bind) on a certain port and client connects to it

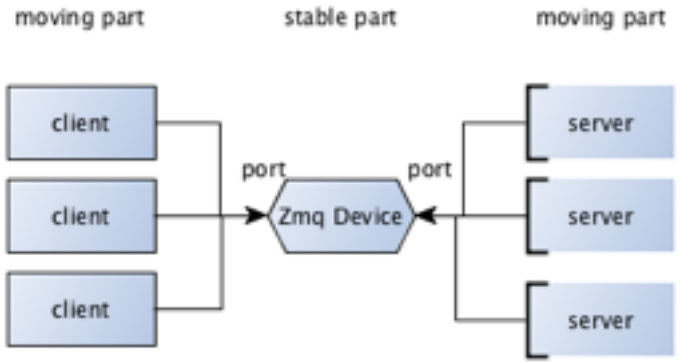
Multithreading & multiprocessing

- Design as message-driven application
- No need of locks and semaphores
- Transport: inproc:// , tcp://

0MQ devices

If both ends are dynamic it is not a good idea to provide well known ports.

ØMQ provides forwarding devices that became the stable point other components connect to



- QUEUE: forwarding device for request/reply communications
- FORWARDER: forwarding device for pub/sub communications
- STREAMER: streamer device for pipelined parallel communications

PAUL SCHERRER INSTITUT

Conclusion

ØMQ provides a communication library with turns out to be fast and reliable:

- the main communication patterns are already defined
- “devices” are in principle not required but can be useful, depending on the network structure.
- can handle different communication protocols and languages
- interoperable within different languages

Yes, it has some drawbacks:

- it provides only the communication layer, we have to build everything else on our own
- doesn’t provide serialisation
- requires developers take care of possible failures

Setup and results

- We are exploring different streaming solutions 0MQ, EPICS and shared memory
- Original data can be multiplied in order to reach ESS-like throughput
- 0MQ and EPICS: behaviour increasing message size / # clients