Messaging Middlewares

Mario Andrés Serruya

1 Introduction

The current document aims to perform a comparison between the different state-of-the-art messaging oriented middleware solutions. The concerned tools will be analysed highlighting multiple characteristics of interest for the implementation of the In-Memory Storage System.

The following messaging libraries conform the ones evaluated: ZeroMQ, RabbitMQ, Mosquitto, Apache Qpid, YAMI4, EVpath and Kafka.

The criteria followed by the analysis performed will be based in a set of points, from which the following could be accentuated:

- Programming language in which the analysed tools have been implemented.
- Scalability.
- Availability.
- Message persistence.
- Implemented protocols in the application. A differentiation will be made between *AMQP* (Advanced Message Queuing Protocol), *MQTT* (Message Queuing Telemetry Transport) and the corresponding ones to the messaging system.
- Communication patterns implemented by the tool. The following models will be taken into account: Publish-Subscribe, Peer to Peer, *ACTive* (Availability for Current Transactions), *Request-Response*, *pipeline* and *survey.*
- Messaging broker implementation.

2 Analyzed Tools

Now, a brief summary of every tool is presented describing each one of them.

2.1 ZeroMQ

ZeroMQ implements its own communication protocols. ZeroMQ constitutes a messaging socket based library [2] which implements communication based on different channels: TCP, PGM (*Pragmatic General Multicast*), IPC (*Inter-Process Communication*) and inproc communication (*Intra-Process Communication*). It is written in the *C* programming language and lacks of messaging broker. The forthright connection of the elements involved in the system supposes a reduction in maintenance costs and justifies the triviality of an intermediate broker. Nevertheless, ZeroMQ does not provided message persistence methods nor availability ones. In any case, facing it with a non persistent implementation of RabbitMQ, ZeroMQ ends up performing a faster and more scalable execution, as it is justified by Estrada and Astudillo in [2].

2.2 RabbitMQ

Facing ZeroMQ, RabbitMQ takes a stand. The corresponding middleware is developed in the Erlang programming language. In this case, the surveyed messaging system implements an intermediate broker in charge of dealing with the messages through queues. RabbitMQ provides message persistence methods as it takes into account the possibility of creating queues in disk. The system itself also considers redundancy and techniques in order to offer a high availability service.

Moreover, scalability constitutes another characteristic of certain RabbitMQ implementations as Rostanski, Grochla and Seman justify in [5].

2.3 Mosquitto

Mosquitto conforms a lightweight messaging system designed to work over TCP/IP transport protocol. Mosquitto implements *MQTT* protocol, which provides a *pub-sub* communication pattern. In relation to message persistence, developing queues at broker level constitutes a possibility in order to avoid message loss in case of failure [4].

The prioritization of Mosquitto as messaging tool takes place when devices limited by poor resources are involved in the system. Nowadays, Mosquitto is mainly used in IoT environments.

2.4 Apache Qpid

In opposition to Mosquitto, the middleware messaging system Apache Qpid exclusively implements *AMQP* protocol. The current possibility considers message persistence through the usage of databases (Apache Derby and Oracle Berkeley DB) and the implementation of queues in disk. As RabbitMQ does, it is also possible to develop an implementation of Apache Qpid in a clustered-based architecture.

The method followed by the studied tool in order to achieve high availability is based in the maintenance of multiple brokers. Initially, clients will be connected to one of the set. Those acting as copies will be also connected to it. In case of failure, client nodes as well as the remaining will be redirected to an auxiliary one successfully increasing the systems's availability [4].

2.5 YAMI4

YAMI4 is a lightweight messaging system based in the well-known master-slave model through which events are managed. It provides a broker and broker-less possibilities, and it implements its own protocol. The previous resource reaches high scalability and performance environments [4].

As the previously referenced systems, YAMI4 also supports a clustered-based implementation among an active set of brokers in order to deal with possible failures. Through those resources, YAMI4 reaches a balanced distribution of the workload as well as high resiliency and availability characteristics.

2.6 Kafka

Apache Kafka is a *streaming* distributed system whose low latency in its messaging queues conforms a key advantage. Additionally, conducted experiments state that the number of dealt messages by unit of time is also a factor to highlight. The justification to it remains behind the sacrifice Kafka performs in relation to reliability in exchange of performance [3]. Besides, Kafka provides *pub-sub* messaging as well as *Point-to-point*.

From a persistent viewpoint, Kafka provides a solution to avoid data leaks due to failures. The previous method is based in the usage of a persistent file system.

2.7 EVpath

EVpath constitutes an event driven middleware system that acts as a transport layer. It was specifically designed to ease the development of overlapped networks. EVpath itself does not implement a communication method, but it provides the key components of the development of different communication paradigms.

EVpath holds up a high performance in the execution of applications through its *stones* system, those correspond to the basic tools which implement each communication paradigm. The previous capability allows answering to higher workloads [1].

3 Comparison

Table 1

Middleware/ Characteristics	Mosquitto	Kafka	EVpath
Development language	C	Scala	C
Relase year	2009	2011	2009
Implements Broker	Yes	Yes	No
Supported Messaging Patterns	Exclusively Pub-Sub	Pub-Sub, Point to Point	Implementation based
Message persistance	Yes	Yes	No
Lightweight	Yes	Yes	Yes
Considered protocols	MQTT, Websocket	Kafka	-
Availability Methods	Tries to achieve it linking brokers	Yes	Yes

Table 2

Middleware/ Characteristics	ZeroMQ	RabbitMQ	Apache Qpid	YAMI4
Development language	C++	Erlang	Java, C++	C++, Objective C
Release year	2007	2007	2005	2010
Implements Broker	No	Yes	Yes	Could be used with or without one
Considered protocols	Request-Reply, Pub-Sub, workload distribution	Request-Reply, Pub-Sub	Request-Reply, Pub-Sub	Request-Reply, Pub-Sub
Message persistance	No	Yes	Yes	No
Lightweight	Yes	No	Yes	Yes
Considered protocols	ZMTP	AMQP, MQTT, REST, STOMP, STOMP over websockets, XMPP over getaway	AMQP	YAMI4 - a WIRE level protocol
Availability Methods	No	Yes	Yes	Yes

References

[1] Greg Eisenhauer, Hasan Abbasi, Matthew Wolf, and Karsten Schwan. *Event-based Systems: Opportunities and Challenges at Exascale*. 2009. `https://dl.acm.org/citation.cfm?id=1619261`.

[2] Nicolás Estrada and Hernán Astudillo. *Comparing scalability of message queue system: ZeroMQ vs RabbitMQ*. 2015. `https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=7360036`.

[3] Vineet John and Xia Liu. *A Survey of Distributed Message Broker Queues*. 2017. `https://arxiv.org/pdf/1704.00411.pdf`.

[4] Suman Patro, Manish Potey, and Amit Golhani. *Comparative Study of Middleware solutions For Control and Monitoring systems*. 2017. `https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=8117808`.

[5] Maciej Rostanski, Krzysztof Grochla, and Aleksander Seman. *Evaluation of highly available and fault-tolerant middleware clustered architectures using RabbitMQ*. 2014. `https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=6933108`.