MODERN MESSAGING
FOR DISTRIBUTED SYSTEMS

Luca Magnoni
CERN/IT-SDC
MIG team

September 3rd, 2014
OUTLINE

- Why Messaging
  - *what’s wrong with socket?*
- Principles
  - *a bit of theory*
- Technology
  - *messaging software*
- Stories
  - *messaging at work*
- Summary
Tightly-Coupled Communication
Tightly-Coupled Communication

Information exchange based on several assumptions:

- Temporal dependency
- Location
- Data structure
- Platform technology (internal data representation)
A LOOSELY-COUPLED SOLUTION: MESSAGING
A LOOSELY-COUPLED SOLUTION: MESSAGING

To minimize producer and consumer dependencies

- Data sent via an intermediate channel
- Channel enhanced with functionality (e.g. queuing)
- **Message** as the information building block
  - body: immutable, structured data (e.g. JSON, Protobuffer)
  - header: key/value pairs, used for routing
**TYPICAL MESSAGING USE CASES**

- **Information Publishing**
  - An entity publishes volatile information with no a-priori knowledge about who is interested
  - e.g. sensor

- **Information Storing**
  - An entity collects information from multiple sources
  - e.g. log collector

- **Remote Procedure Call**
  - An entity sends request to one or more remote entity and expect reply
  - e.g. command distribution
MESSAGING TODAY

- Rich ecosystem of tools and services with years of development
  - message brokers & libraries
  - based on well-defined communication and integration patterns

- New technology recently appeared with focus on low-latency/high-performance

- Not the silver-bullet
  - important benefits in flexibility and scalability
  - more complex programming model and architecture
**Main Communication Models**

- **Point to Point (Queue)**
  - no consumers: message is kept in the channel
  - multiple consumers: message is delivered only once

- **Publish/Subscribe (Topic)**
  - no consumers: message is discarded
  - multiple consumers: message is delivered multiple times

- More complex delivery semantics exist
No standard way to interact with message brokers (yet)
- Many protocols, with different capabilities and QoS
- Protocol is bound in the application
- Messaging system can be polyglot
- AMQP *may* eventually unify
MESSAGING PROTOCOLS

- **AMQP = Advanced Message Queuing Protocol**
  - major contributors: Cisco, Microsoft, Red Hat, banks
  - designed for interoperability between different messaging systems
  - feature complete but complex (wire protocol + semantic definition)
  - binary protocol, in theory faster and compact
  - major transition: from AMQP 0.x to AMQP 1.0, no backward compatibility
  - brokers support still incomplete

- **STOMP = Streaming Text Orientated Messaging Protocol**
  - text-based, meant to be simple and widely-interoperable
  - extensive set of clients
  - supporting basic messaging features (e.g. transaction)
  - supported by many brokers
Messaging Protocols (II)

- MQTT (Message Queue Telemetry Transport)
  - originally from IBM, recently open sourced
  - low footprint: designed for low bandwidth, high-latency networks
  - Internet of Things style applications (mobile phones, etc.)
  - strength: simplicity, compact binary format, good fit for simple push scenario

- Many other broker-specific protocols exist (e.g. Openwire)

No clear best protocol, but the protocol choice is an important decision in respect to the messaging system coupling within the application
JAVA MESSAGE SERVICE (JMS) API

- Messaging standard that allows application components to create, send, receive, and read messages
- Define messaging main concepts
- Mature (1.1 spec. in 2002, 2.0 in 2013) and part of Java EE
  - It is (mainly) an API
  - Supports (mainly) two models:
    - Point-to-point
    - Publish/subscribe
- Many JMS frameworks/implementations available: commercial application servers, Spring, ActiveMQ, etc.
MESSAGING FUNCTIONALITY

- **Common features**
  - *Persistence, Fail-over, Guaranteed delivery, Ordering, Transaction, High-availability, Clustering*
  - shared by main brokers, with different interpretation

- **Many unique broker-specific features**
  - *e.g. Virtual Queues, Last Value Queues*

- Suggestion: focus on real requirements
MESSAGE BROKERS (*MQ)

- Many exist
- Most feature rich
- Different in features, protocols, implementation languages, platform support
- Focus here on open-source solutions
## Non-exhaustive Comparison

<table>
<thead>
<tr>
<th>Broker</th>
<th>Qpid</th>
<th>MRG</th>
<th>HornetQ</th>
<th>ActiveMQ</th>
<th>Apollo</th>
<th>RabbitMQ</th>
</tr>
</thead>
<tbody>
<tr>
<td>Language</td>
<td>Java</td>
<td>C++</td>
<td>Java</td>
<td>Java</td>
<td>Scala</td>
<td>Erlang</td>
</tr>
<tr>
<td>Main Protocols</td>
<td>AMQP</td>
<td>AMQP</td>
<td>proprietary STOMP</td>
<td>OpenWire STOMP AMQP MQTT</td>
<td>OpenWire STOMP AMQP MQTT</td>
<td>AMQP STOMP (MQTT)</td>
</tr>
</tbody>
</table>

| Owner (*) | Red Hat | Red Hat | FuseSource (Progress) | VMware |

- 2012 RedHat completed FuseSource acquisition
**APACHE ACTIVEMQ**

- *The most widely adopted open-source message broker*
- Written in Java
- Commercially supported by RedHat
- Solid and mature
- Many features (e.g. JDBC message stores, advanced clustering configuration)
- Weak point: complexity
- Future:
  - Apollo seemed the designed successor
  - RedHat may evolve ActiveMQ as unified messaging solution
RABBITMQ

- Open source
- Written in Erlang
- Modular design (e.g. AMQP, MQTT plugins)
- Development effort comes from SpringSource (a division of VMware)
- Embedded in several projects (e.g. Logstash, Sensu)
CONSIDERATIONS ON PERFORMANCE

- Beware of numbers!
  - msg/s has little value without context
  - in realistic scenario performance vary across 100K msg/s and few K msg/s (Broker comparison over STOMP)

- To handle with care:
  - message > 1KB payload
  - rate > 10K msg/s LAN & binary protocol
  - persistent message (at least x10 slower)
  - > 100 new sessions per second (in particular with security)
  - number of subscribers
  - slow subscribers!!!
CONSIDERATIONS ON PERFORMANCE (II)

- Messages go through intermediaries
  - multiple hops, delays, retries, failures
  - not a good fit for low-latency / high-throughput

- New projects tackled the problem from different perspective...
Kind of broker: APACHE KAFKA

- From LinkedIn, now Apache
- Written in Scala
- Focus on data movement
- Meant for real-time activity stream analytics (e.g. user data, metrics)
- Designed as distributed system easy to scale out (based on Zookeeper)
- Limited messaging features
  - (Mainly) Topic semantic
  - Filesystem based Persistence
  - Guaranteed ordering
  - Automatic balancing of consumer/producer/broker
**APACHE KAFKA - DESIGN**

- Kafka broker is stateless
  - consumer maintains its own state and pools data from the broker
- Messages are persisted independently from consumers
  - not removed on consumption, but by retention period
  - allow for re-consumption in case of problem
- Fast writes and batch-reads (zero-copy)
- # consumers is not relevant
- Used as *pipe* to different processing systems (e.g. Hadoop, Storm)
**Not a broker: ZEROMQ (ZMQ, 0MQ)**

- Despite the name, it’s **not a broker, but a library**
- **Socket on steroids**: expand the concept of socket
- 0MQ Socket built-in messaging patterns:
  - Request-Reply
  - Pub/Sub
  - Pipeline
  - Exclusive Pair
- Each socket’s connection has its own *queue*
**Zer0MQ - Examples**

- Inter-thread, inter-process or inter-node communication
- Work seamlessly with single or multiple clients
- Work with server started after the client

![Diagram showing communication between client, server, publisher, and subscribers.]

**Publisher**
- `PUB`
- bind
- updates

**Client**
- `REQ`

**Hello**

**World**

**Server**
- `REP`

**SUB**
- Subscriber

**SUB**
- Subscriber

**SUB**
- Subscriber
**ZeroMQ - overall**

- Lightweight but powerful
- Focus on high-performance low-latency communication
- Price to pay in application complexity
  - Comes with basic messaging features
- Good if you can invest sufficient effort to build a messaging solution on top of 0MQ as library
CERN Beam Control

- "No messaging, no beam"
  - Felix Ehm, CamelOne 2012

- Messaging for highly reliable control/monitoring/alarm applications for LHC
  - since 2005
CERN Beam Control - Technology

- Clusters of ActiveMQ brokers
  - safety system: 30 producers, 2MB/s each
  - 4.5 K msg/s, many consumers
  - store & forward

- Corba to 0MQ migration for control middleware
  - [Middleware Trends and Market Leaders 2011]
  - [The new CERN Controls Middleware 2012]
MESSAGING & ONLINE MONITORING

- ATLAS TDAQ
  - distribute operational alarms
  - from private TDAQ network to GPN
  - master/slave configuration
  - single outbound connection

- STAR Online framework
  - detector state processing, storage and monitoring
  - AMQP-based framework for flexible, loosely-coupled stream processing
  - investigation on MQTT for control framework
  - Dmitry Arkhipkin, ACAT 2014
WLCG Messaging Service

- Monitoring WLCG sites and services:
  - 50k clients all around the world
  - 100k msg/s or more

- STOMP on ActiveMQ, (RabbitMQ, Apollo)
- Messaging cluster made of independent brokers
  - produce to any, consume from all
SUMMARY

- **Messaging is fundamentally a pragmatic reaction to the problems of distributed systems**
  - asynchronous architecture
  - loose coupling
- It can bring many benefits in distributed applications flexibility and scalability
  - with implications in application and infrastructure complexity
- Messaging technology is still evolving
  - AMQP standardization effort in the good direction, but not yet mature
- New generation of systems promote messaging for low-latency / high-throughput communication
  - narrowing use cases and relaxing assumptions
REFERENCES

- CERN Messaging service:
  - [http://mig.web.cern.ch]
- *Enterprise Integration Patterns*, G.Hohpe and B.Woolf
  - [http://www.eaipatterns.com]
- Brokers comparison over STOMP
  - [http://hiramchirino.com/stomp-benchmark/ec2-c1.xlarge]
THANK YOU! ANY QUESTION?