Distributed Message Queues

CS199 - ACC

Prof. Robert J. Brunner

Ben Congdon
Tyler Kim
Bhuvan Venkatesh

How’s it Terraforming?

Remember the MP is due next Friday

What is a Distributed Message Queue?

- A distributed message queue (DMQ) is a fault tolerant program that takes and delivers messages
- Messages can be whatever you want, usually they are bytes
- DMQs can be very fast and have low memory overhead
- They can allow your application to be programming language agnostic
- They are the base for many cloud infrastructures nowadays

Pub-Sub

- The model for a distributed message queue is pub-sub (short for publisher subscriber).
- Distributed message queues have channels (sometimes sub-channels) and publishers publish to the channel
- Subscribers pull messages from the channel
- Messages may be repeated, but that is fine!

Advanced Message Queueing Protocol

- Developed by multiple companies (actually introduced by banks)
- Defines how messages are formatted, routed, delivered, and acknowledged
- It also allows for a type system, so there are type checks
- If a system is AMQP compliant, one can switch out the old system for the new system without too many problems
- The actual details of AMQP would require a little bit of depth into routing algorithms and whatnot

Big Distributed Message Queues

- Arguably the most used message queue
- Gives you a lot of flexibility in routing (You can attempt to route to a specific machine even though it is not a good idea)
- Well supported and errors are pretty prominent
- Fascinating article about using RabbitMQ as a mutual exclusion service
    - https://aphyr.com/posts/315-jepsen-rabbitmq

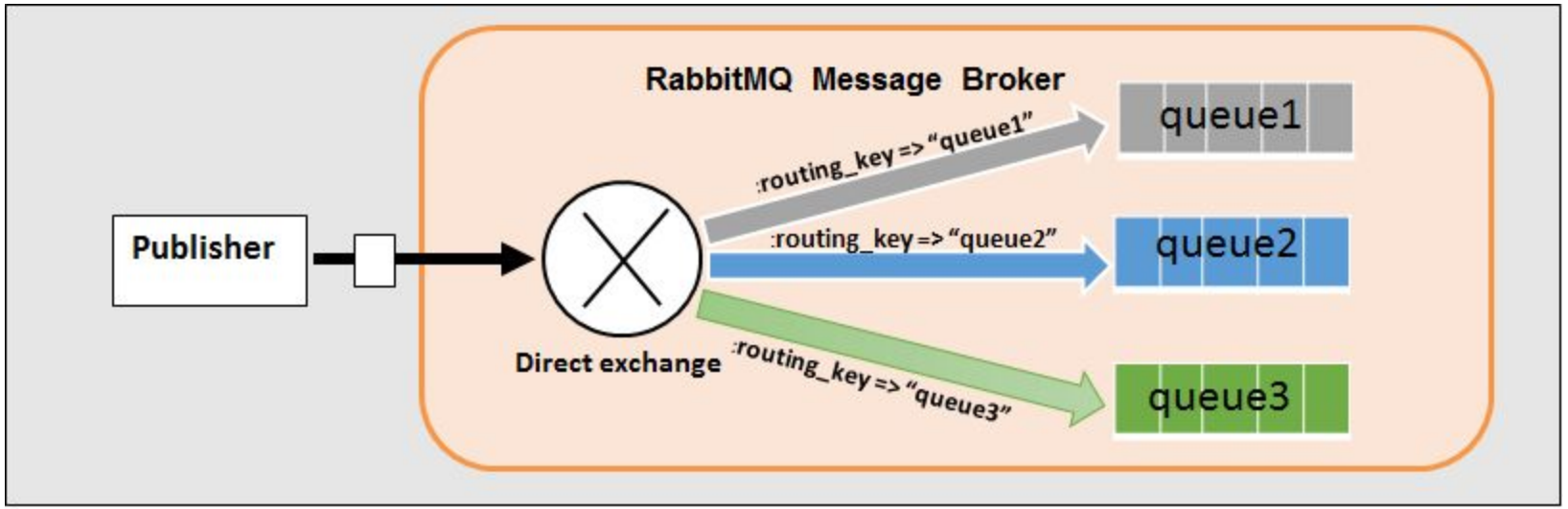

Fig: Direct Exchange Routing

Another One

- Same deal, but developed by Apache
- Nothing too amazing here
- So you can have a platform that is completely Apache driven which is generally good. ASF is a hallmark of quality, well-maintained software.

Cloud

Amazon has their messaging service (simple queue service) that can connect different parts of the cloud together

Google + Microsoft have their own but again amazon is king

Uses Cases

Streaming

- Remember last week?
- We can now use Distributed Message Queues to try and stream data
- Not always the best option because of having to come to a consensus about ordering
- Can be convenient if already in place
- Extraordinarily good when each of the messages have a lot of data (we can essentially batch operations)

Gatekeeper

- You can rate limit messages coming in
- This may be useful if you are doing a streaming experiment and believe that randomly picking samples is an effective way of doing so
- At fitbit, there was A/B testing with the distributed messaging service that sampled the entire application and re-routed to the new algorithm

Apache Thrift

- Apache thrift is a programming language agnostic way of sending objects
- You define a language agnostic API
- Thrift will then let you compile each of the modules to the native languages
- What thrift does under the hood is also generate a lot of boilerplate code
- The boilerplate can be yucky
- Thrift is one big dependency injection machine essentially

Thrift Example

```
typedef i32 int
service DatabaseService {
        void save(1:int id, 2:Object message),
}
struct Object {
       1: required i32 trackerNum;
       2: required string sleepInfo;
       3: required int lengthSleep;
}
```

Microservices!

- This is where the idea of microservices come into play
- If we can now define operations to be small, compact applications that fulfill the requirements in the thrift files
- You don’t even care what technologies the microservice uses because as long as you get your request back you are done

Cloud Architecture

- Now, we can formally put together a mock cloud architecture with all of the technologies that we have studied through in class
- A simple cloud architecture has a messaging queue at its hub
- Microservices handle each “territory” of the application
- One microservice learns about another microservice through zookeeper
    - This is what developer fear called “service discovery”
- The microservices call each other with thrift and process requests