Distributed Enterprise Messaging with MantaRay

By Amir Shevat

As software developers, we often encounter the problem of communication between two or more applications we write. The problem gets more complex when the applications are not on the same computer; the alternative of implementing this communication on your own (over TCP or UDP) is appealing at first, but turns out to be a hard task that takes time to debug and stabilize. Using a more structured protocol like HTTP is easier, but you still have to handle failure and many other communication issues. Experienced programmers know to keep their development contained to the core business and utilize third-party, standardized products for issues like communication.

A very important communication standard in Java is Java Messaging Service (JMS). JMS is a set of Java interfaces and associated semantics that define a way for a Java-based client to access a messaging system. JMS provides a rich, yet simple, set of messaging facilities for creating powerful enterprise applications.

A JMS *provider* is the entity that implements JMS for a messaging product. JMS's feature set is broad enough to allow interfacing with existing, possibly non-Java, messaging systems. Traditionally, JMS vendors use a *broker* to communicate with all of the elements in the messaging system.

This article describes a unique distributed messaging solution and a JMS provider called MantaRay, and how it transformed a traditionally centralized and broker-based concept like JMS to a fully distributed system. It also shows what happens behind the scenes in a distributed system when performing JMS operations.

MantaRay

MantaRay is an open source, server-less transport layer designed specifically for heterogeneous, distributed, and high-traffic environments. It has a JMS API (1.1 and 1.02), as well as other proprietary APIs. Compared to traditional messaging systems, such as busses or brokers, MantaRay provides a much higher level of robustness, better performance, and faster implementation by eliminating the need for a broker and bringing all functionality to the edges. MantaRay eliminates the single point of congestion and single point of failure of traditional models by using a distributed, server-less architecture. The distributed application is lightweight and operating-system-agnostic.

Distributed versus Broker Communication

With a broker-based architecture, all of the applications in the network only "know" the broker, while the broker "knows" all the applications. When Application A wants to send a message to Application B, it sends it to the broker and the broker sends it to Application B, and vice versa.

Figure 1 show how applications communicate in a centralized environment.

Figure 1. Communication in a centralized environment

Because MantaRay is fully distributed and has no broker, the elements in the MantaRay network need to "know" one another. In order to achieve this, the system runs through a process of automatic discovery using a multicast channel. All of the MantaRay information, such as IP, port, and even role information (i.e., "Application A is a subscriber of topic 'fun'"), is published on a common multicast address. All MantaRay elements listen to this channel and from it, receive information about their peers. After Application A has discovered Application B, it is able to open a TCP/SSL connection to it for direct communication.

Figure 2 show how applications communicate in a distributed environment.

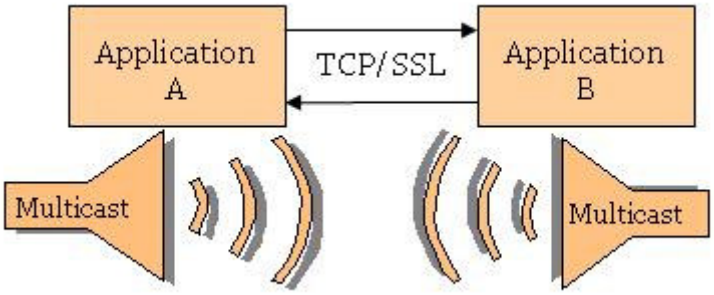

Figure 2. Communication in a distributed environment

Advantages of a Distributed Environment

This approach has its advantages and disadvantages.

No Single Point of Congestion

When communication is done via a centralized broker, the broker can easily become the single point of congestion. Let's say we have Applications A and B, which do not send many messages, but need their messages to be delivered quickly. On the same system, you have Applications C and D, which send millions of messages an hour. In this scenario, a centralized broker will have to manage all of these millions of messages, and despite its best efforts, the strain of handling millions of messages will impact its performance. Why should the heavy traffic between C and D interfere with communication between A and B?

In a distributed environment, A and B communicate using a direct communication, and are not bothered with any other communication that is done in the system. Thus, they do not suffer degradation of service because of the communication of C and D.

No Single Point of Failure

Even if you have a strong centralized server with hot backups and load balancing, you still face the possibility that the broker will crash. When a broker crashes, the whole system goes offline until the broker is brought up again.

Computers can, of course, crash in a distributed environment, but because the communication is distributed, only the elements on the crashed computer are affected. The other elements in the messaging system are not affected and continue to communicate seamlessly. When planned correctly, distributed environments can be very durable in comparison to centralized environments.

Simpler to Deploy and Cheaper to Maintain

When adding more and more applications and computers to your messaging system, a centralized environment forces you to reinforce the broker, in order to handle the additional traffic. This is not the case with MantaRay; here, you just need to install the MantaRay layer on the additional applications in the system, and you are good to go.

Disadvantages of a Distributed Environment

Learning the Environment

Elements in the distributed environment need to "learn" their environment; there is no "know-it-all" broker that manages everything. MantaRay uses multicast to discover elements in the system. On average, it takes about 1.5 seconds to learn the system, which could be a disadvantage for some systems.

The Flip-Flop Problem

Because every element in the distributed environment is its own small broker, there is a problem of *flip flop*. The flip-flop problem occurs when Application A wants to send a message to Application B, but Application B is down, and there is no time when Application A and B are up at the same time.

This flip-flop problem can be solved using a third element in the system that will route messages to their destinations. In MantaRay, there is an element called a *queue coordinator* that helps you solve this problem. I'll discuss queue coordinators in greater detail later.

Distributing the JMS Messaging Domains

JMS defines two domains of messaging: *point-to-point* and *publish/subscribe*.

- **Point-to-point** (PTP) is built around the concept of message queues. Each message is addressed to a specific queue; clients extract messages from the queue(s) established to hold their messages.
- **Publish/Subscribe** (Pub/Sub) defines how JMS clients publish messages to, and subscribe to messages from, a well-known node in a content-based hierarchy. JMS calls these nodes *topics*.

This section elaborates about these domains, how they are implemented in a traditional broker-based environment, and how MantaRay implements them.

Topics

Applications can communicate with each other using topics: one or more applications subscribe to a topic, and other applications publish data on the topic. There could be any number of publishers and subscribers for each topic, and any number of topics in the system.

A message published by a publisher will be received by all subscribers; a subscriber could even be offline (i.e., the process is down) and, after recovering, still get the messages that where published while it was down.

Figure 3 shows the notion of a topic.

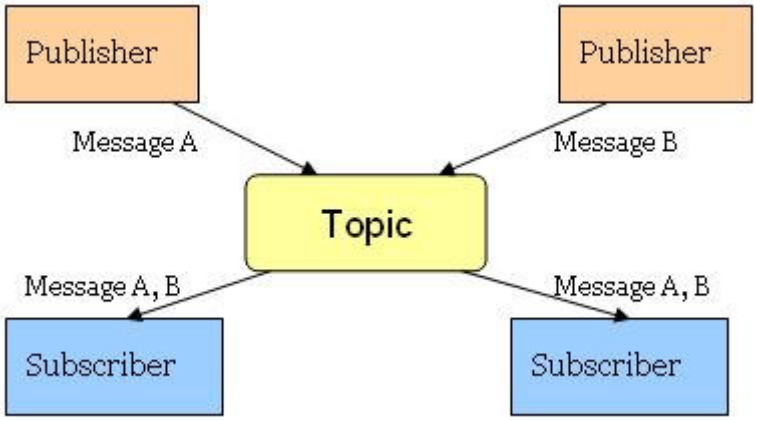

Figure 3. The notion of a topic

In a broker-based architecture, the topic messages and subscriptions are managed by the centralized broker. All published messages are sent to the broker and the broker is responsible for delivering the messages to all subscribers.

Figure 4 shows how topics work in a centralized environment.

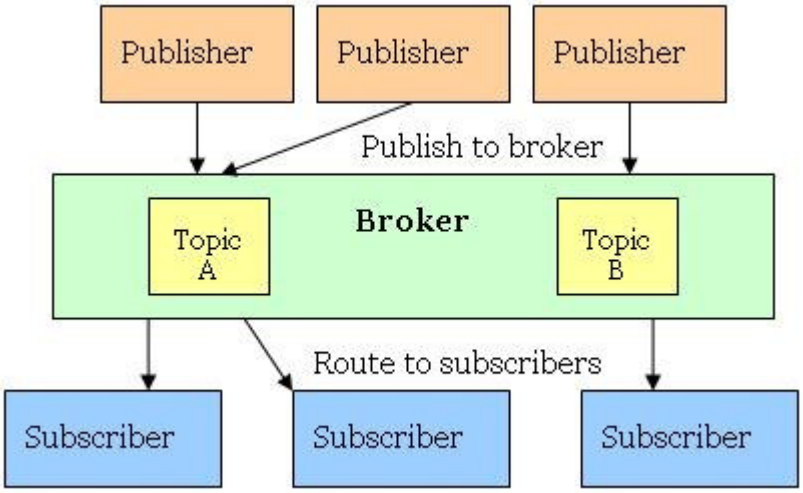

Figure 4. Topics in a centralized environment

MantaRay removes the broker from the architecture, so Publishers send the messages directly to the subscribers. The communication is TCP-based and can be encrypted using SSL.

Figure 5 shows how topics work in a distributed environment.

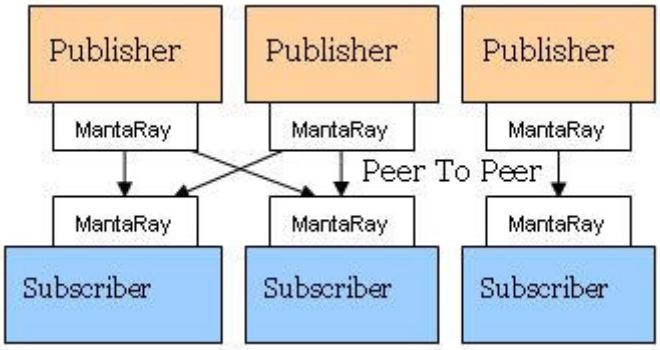

Figure 5. Topics in a distributed environment

Queues

Applications can also communicate with each other using queues of messages. One application sends messages to the queue and another receives the messages from the queue. There could be any number of message producers and consumers on a given queue and there could be any number of queues in the system.

Messages are kept in the queue in the order in which they were sent until a consumer requests a message. Each message will be delivered to one and only one consumer. This delivery algorithm is the main difference between queues and topics.

Figure 6 shows the notion of a queue.

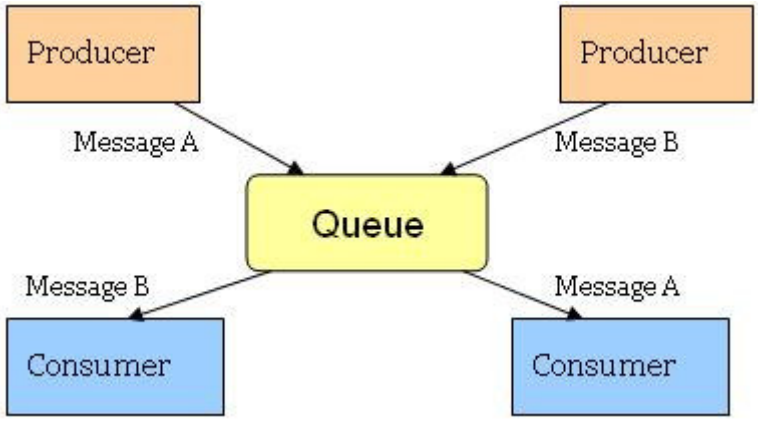

Figure 6. The notion of a queue

Consumers can register themselves as queue listeners. By doing so, they eliminate the need to request a message every time. In the queue listener mode, a message that is received by the queue will be delivered immediately to one of its listeners.

In a broker-based architecture, all queues are managed by the centralized broker in a manner very similar to topics.

Queues are centralized by their nature, because of the order requirement and the "once and only once" delivery constraint. In MantaRay, we kept the queue as a centralized logical entity and still removed the need for a broker. We have introduced a third role (beyond consumer and producer) called a *queue coordinator*, which can reside on any MantaRay element in the network. Different MantaRay elements can

coordinate different queues. The only rule that has to be kept is that there is only one queue coordinator per queue.

Figure 7 shows how queues work in a distributed environment.

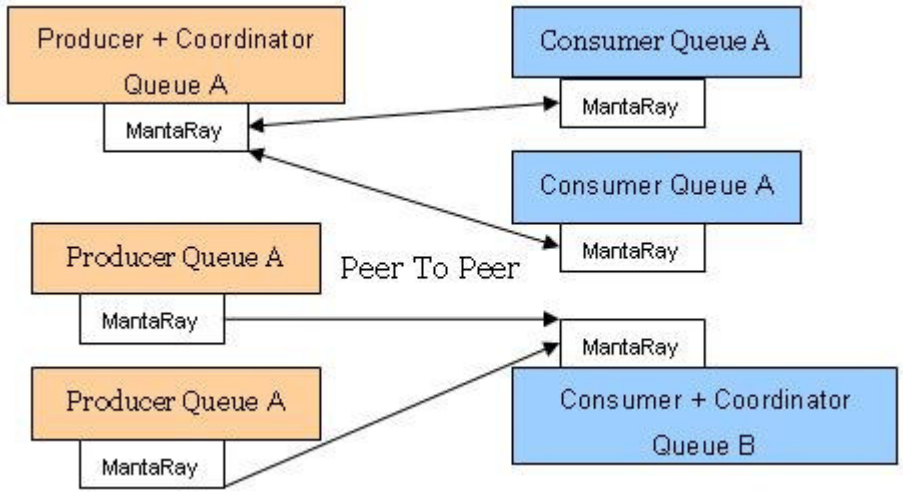

Figure 7. Queues in a distributed environment

The coordinator for each queue is a configurable attribute of MantaRay. Once a MantaRay element is configured to be the coordinator of a queue, it will automatically manage that queue. The queue coordinator can even be a dedicated standalone MantaRay process, which does not act as a consumer or producer at all.

If planned correctly, this queue architecture can be much faster than a queue on a broker. Let's say we have only one consumer of messages from the queue; this means we can configure the queue coordinator on the same MantaRay instance. This gives us a very fast queue--one that does not require network traffic. On the other hand, if you have only one producer and multiple consumers, you can configure the queue coordinator to be on the same process as the producer and queuing a message into the queue will not require network traffic.

Inter-Process Messaging

Messaging can also be useful for inter-process communication; different logical components and different tiers in an application can communicate with each other using JMS. This abstraction helps you decouple the tiers and more easily separate them into different applications when that time comes.

MantaRay enables inter-process communication with no network traffic. If MantaRay "knows" that the destination of a message is in the same JVM, it skips all of the network stuff (serialization, network send and receive, and deserialization), making the communication much faster.

Using Standards

While writing a proprietary code can be fun, using standards is very important. Standards have been proven and accepted by the community and usually cover all bases. MantaRay tries to use standards as much as possible:

- JMX (Java Management Extensions) infrastructure lets you easily manage and configure the MantaRay layer.
- Commons-logging lets MantaRay logs plug into the logs of the application that uses MantaRay.
- MantaRay keeps the persistent *world view* (of all of the other MantaRays in the system) in a well-formed, easy-to-read XML file.

Conclusion

When considering a messaging solution, there are many things to take into account: stability, robustness, high availability, and speed. Up until now, you could only solve these problems by buying a stronger computer and a stronger network infrastructure. This meant you'd need your brokers to be backed up, clustered, and load-balanced. While these options are still valid, MantaRay provides a different point of view on the messaging world; one that is less costly, and one that is easy to deploy and manage.

Issues of scaling and cost should be considered, as well. Until recently, scaling up was very costly and sometimes a big pain. Now you have the option to start small and easily scale up.

Because it is a lightweight solution and easy to deploy, and because of its distributed nature, MantaRay can be an ideal solution for your Java messaging needs.