分布式消息系统研究综述

吴 璨[1,2] 王小宁[1] 肖海力[1] 曹荣强[1] 赵一宁[1] 迟学斌[1,2]
(中国科学院计算机网络信息中心 北京 100190)[1] (中国科学院大学 北京 100049)[2]

摘 要 随着大数据时代的到来,各类软硬件系统的高并发访问、海量数据处理等需求越来越多,系统的高可用、易伸缩、可扩展成为系统研发的首要目标,分布式系统应运而生,提供了满足高性能需求的解决方案。然而,系统分布式地部署在不同的计算机上,使得系统间的消息通信成为重要问题。文章综述了 4 种流行的开源分布式消息系统,对比分析了 RabbitMQ,Kafka,ActiveMQ 和 RocketMQ 的架构及性能,为科研人员和系统开发者选择分布式消息系统提供了参考意见。

关键词 分布式消息系统,RabbitMQ,Kafka,ActiveMQ,RocketMQ
中图法分类号 TP311 **文献标识码** A

Survey on Distributed Message System

WU Can[1,2] WANG Xiao-ning[1] XIAO Hai-li[1] CAO Rong-qiang[1] ZHAO Yi-ning[1] CHI Xue-bin[1,2]
(Computer Network Information Center, Chinese Academy of Sciences, Beijing 100190, China)[1]
(University of Chinese Academy of Sciences, Beijing 100049, China)[2]

Abstract With the advent of the Big Data Era, there have been increasing demands for the high concurrent access and mass data processing of all kinds of hardware and software systems. High availability, extensibility and scalability are the main driving forces for system development. Distributed systems emerge as new age comes, providing solutions for higher performance requirements. However, as distributed systems are deployed on different computers, message communication between the systems has become an important problem. This paper provided an overview of the research progress of four popular open source distributed message systems: RabbitMQ, Kafka, ActiveMQ and RocketMQ. The architecture and performance of them are compared and analyzed, providing information and references for researchers and developers when choosing distributed message systems as an option.

Keywords Distributed message system, RabbitMQ, Kafka, ActiveMQ, RocketMQ

分布式系统[1]是一个硬件或软件组件分布在不同的网络计算机上,彼此之间仅通过消息传递进行通信和协调的系统。分布式消息系统[2-3]是分布式系统之间进行消息传递的重要组件,它利用高效、可靠的消息传递机制进行平台无关的数据交流,并基于数据通信进行分布式系统的集成。目前,分布式消息系统已经在企业中广泛应用,甚至有些企业已自主研发更加适合企业应用场景的消息系统,如阿里巴巴等。为了向科研人员和系统开发者提供更多参考信息,本文对比分析了 4 种流行的开源分布式消息系统的架构及性能。

分布式消息系统中的消息传递主要有两种模式,分别是点对点模式[4]和发布-订阅模式[5-6]。

点对点模式(Point to Point,PTP)的结构如图 1 所示。

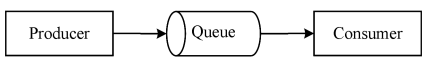

图 1 点对点模式结构图

消息生产者生产消息,将消息发送到队列中,消息消费者从队列中取出并且消费消息。消息被消费后,将其从队列中移除,一条消息只能被一个消费者消费。

发布-订阅模式(Publish-Subscribe,Pub-Sub)的结构如图 2 所示。消息生产者生产消息,将消息发布到 Topic 中,消息消费者订阅 Topic,从 Topic 中取出并且消费消息。消息一直保存在 Topic 中,一条消息可以被所有订阅该 Topic 的消费者消费。多数分布式消息系统采用发布-订阅模式进行消息传递,有些消息系统也支持点对点模式的消息传递。

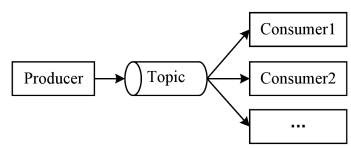

图 2 发布-订阅模式结构图

1 系统描述

1.1 RabbitMQ

1.1.1 概述

RabbitMQ[7-10]是一个基于 AMQP 协议[11]可复用的企业

本文受国家重点研发计划课题(2018YFB0204001),中国科学院青年创新促进会(2017216)资助。

吴 璨(1992-),女,博士生,主要研究方向为高性能计算、可视化与网格技术,E-mail:wucan@sccas.cn;王小宁(1981-),女,博士,副研究员,主要研究方向为网格计算、云计算和分布式系统,E-mail:wxn@sccas.cn(通信作者);肖海力(1978-),男,博士,研究员,主要研究方向为网格技术和分布式系统;曹荣强(1982-),男,博士,副研究员,主要研究方向为网格计算和云计算;赵一宁(1983-),男,博士,助理研究员,主要研究方向为分布式系统与大数据分析;迟学斌(1963-),男,博士,研究员,博士生导师,主要研究方向为高性能计算、可视化与网格技术。

级消息系统。其使用 Erlang 语言开发，支持多种客户端，如 Python，Ruby，Java，JMS，C，PHP 等，可以运行在 Erlang 语言所支持的平台之上，如 Solaris，BSD，Linux，MacOSX，TRU64，Windows NT/2000/XP/Vista/Windows7/Windows8，Windows Server 2003/2008/2012，Windows 95，Windows 98，VxWorks 等。

1.1.2 架构分析

RabbitMQ 架构如图 3 所示，主要包含 Broker，Queue，Exchange，Binding，Virtual Host，Connection 和 Channel 等基本概念。

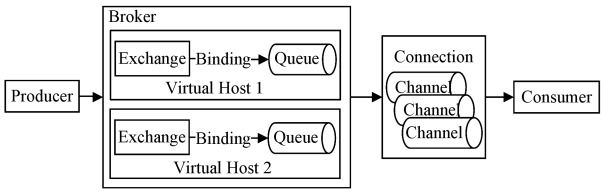

图 3 RabbitMQ 架构图[12]

Broker：消息队列服务器，用于接收和分发消息。每一个 RabbitMQ Server 是一个 Broker，一个 RabbitMQ 集群由一个或多个 Broker 组成。

Queue：用于存储消息，一条消息可以被转发到多个 Queue 中。

Exchange：用于转发消息，根据分发规则，将消息转发到 Queue 中。

Binding：Exchange 和 Queue 之间的虚拟连接，Binding 中包含 RoutingKey。Binding 信息保存在 Exchange 的查询表中，用作消息的分发依据。根据 RoutingKey 绑定的 Exchange 与 Queue 的对应规则，决定消息发送的方向。

Virtual Host：用于区分不同用户权限。把 AMQP 的基本组件划分到一个虚拟的分组中，当多个不同的用户使用同一个 Broker 时，可以划分多个 Virtual Host，每个用户在自己的 Virtual Host 上创建 Exchange，Queue 等。Broker 中可以有多个 Virtual Host。

Connection：Producer，Consumer 和 Broker 之间的 TCP 连接。除出现网络故障、Broker 宕机情况外，Broker 不会断开连接，只有客户端可以断开连接。

Channel：Connection 内部建立的逻辑连接。每个 Channel 代表一个会话任务，所有操作都在 Channel 中进行，如果应用程序支持多线程，通常每个线程创建单独的 Channel 进行通信。Channel 作为轻量级的 Connection，减少了系统建立 TCP Connection 的开销。Channel 之间是完全隔离的，Connection 中可以建多个 Channel。

RabbitMQ Broker 提供 4 种 Exchange 模式，分别为 Direct Exchange，FanoutExchange，Topic Exchange 和 Headers Exchange。Direct Exchange 是 RabbitMQ Broker 默认的 Exchange 模式。此模式处理 RoutingKey，Queue 与 Exchange 绑定，将发布到 Exchange 的消息转发至与 RoutingKey 完全匹配的 Queue 上。FanoutExchange 模式不处理 RoutingKey，Queue 与 Exchange 绑定，将发布到 Exchange 的消息转发到与 Exchange 绑定的所有 Queue 上，类似于广播，FanoutExchange 模式转发消息速度最快。Topic Exchange 模式处理 RoutingKey，Queue 与 Exchange 绑定，将发布到 Exchange 的消息转发至与 RoutingKey 模糊匹配成功的 Queue 上。符号“*”匹配一个词，符号“#”匹配一个或多个词。例如，“example.*”只会匹配到“example.a”，而“example.#”能够匹配到“example.a.b”。Headers Exchange 模式不处理 RoutingKey，Queue 与 Exchange 绑定，绑定时 Queue 提供一组键值对，如果消息的键与其完全匹配，则 Exchange 将消息转发到 Queue。此模式与 Direct Exchange 的实现思路是一样的，但是性能差很多，因此不实用。

Producer 发送消息至 Exchange，Exchange 根据 Exchange 模式匹配到对应的 Queue，将消息存入 Queue 中，Consumer 从 Queue 中取出消息进行消费。

1.1.3 性能分析

(1)负载均衡

RabbitMQ 通过设置预取数目(Prefetch Count)进行负载均衡。如果有多个 Consumer 同时订阅同一个 Quque 中的消息，Quque 中的消息会被平均分配给多个 Consumer。若每条消息的处理时间不同，就会导致某些 Consumer 一直很忙，而另一些 Consumer 很快处理完工作，处于空闲状态。可以设置 Prefetch Count=1，则 Quque 每次给每个 Consumer 发送一条消息；Consumer 处理完这条消息后，Quque 会再给 Consumer 发送一条消息，以实现负载均衡。

(2)可靠性

RabbitMQ 通过消息回执(Message Acknowlegment)机制、消息持久化(Message Durability)机制和消息跟踪机制保证系统的可靠性。

Consumer 在消费完消息后，发送一个回执给 Broker，Broker 收到消息回执后，才将该消息从 Queue 中移除。如果 Broker 没有收到回执，并检测到消费者的 Broker 连接断开，则 Broker 会将该消息发送给其他 Consumer 进行处理。

当将 Queue 设置为可持久化(Durability)时，数据会保存在硬盘上，保证了数据的可靠性。

同时，RabbitMQ 提供消息跟踪机制，如果出现消息异常的情况，用户可以通过跟踪机制找出异常。

另外，RabbitMQ 支持集群模式，多台服务器的 Queue 数据同步，若一台服务器丢失消息则可从其他服务器上获取，以保证数据不会丢失。

(3)可扩展性

RabbitMQ 提供了许多插件，用户可以从多方面进行扩展，也可以根据需求开发自己的插件。

1.2 Apache Kafka

1.2.1 概述

Apache Kafka[13-16]是一个高吞吐量的分布式消息系统，采用发布-订阅模式传递消息，使用 Scala 语言开发。其最初由 LinkedIn 公司开发，之后成为 Apache 软件基金会的一个开源项目，于 2011 年初开源。

Kafka 客户端语言丰富，支持 Scala，Java，C++，C#，Go，PHP，Python，Ruby，PHP，.net 等多种语言。基于 ZooKeeper 协调负载均衡，Kafka 具有实时处理大量数据的能力，支持在线水平扩展，适用于批处理、实时处理、访问日志、消息服务等场景。

1.2.2 架构分析

如图 4 所示，Kafka 架构中有 5 个重要概念，分别为 Broker，Topic，partition，Producer 和 Consumer。

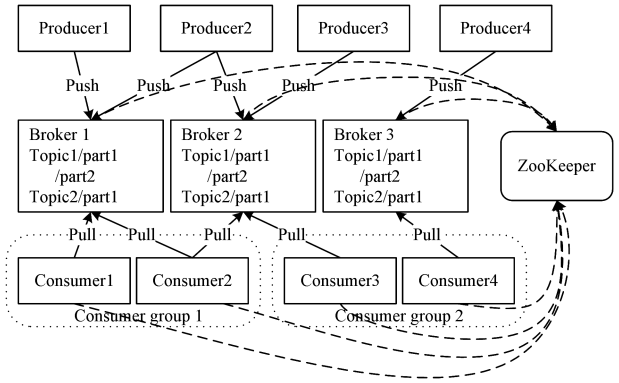

图 4　Kafka 架构图[17]

Broker:用于持久化和备份消息。每一个 Kafka Server 是一个 Broker,一个 Kafka 集群由一个或多个 Broker 组成。

Topic:消息按照类别区分,每个类别是一个 Topic,一个 Broker 可以容纳多个 Topic。

partition:用于存储消息数据和索引文件。partition 是一个有序的队列,每条消息在 partition 中拥有一个偏移量(offset),保证了消息的有序性,一个 Topic 可以包含一个或多个 partition。

Producer:生产消息,产生的消息将被发送到 Broker 的 Topic。

Consumer:消费消息,消费的消息内容来自 Broker 的 Topic。Kafka 在创建 Consumer 时,设置其属于某个 Consumer Group,一个 partition 的数据可以同时被多个Consumer Group 消费,但是只能同时被 Consumer Group 内的一个 Consumer 消费。当一个 Topic 被多个 Consumer Group 订阅时,如果每个 Consumer 属于不同的 Consumer Group,则实现了消息的广播;如果所有 Consumer 属于相同的 Consumer Group,则实现了消息的单播。

Producer 产生消息,向 Broker 的 Topic 发布;Consumer 订阅 Broker 的 Topic,主动从 Broker 中拉取消息进行消费,通过指定 offset 可以消费任意位置的消息。

1.2.3　性能分析

(1)效率

Kafka 通过 partition 设计和 consumer group 设计提高了系统的吞吐量和并发处理能力。

Kafka 在创建 Topic 时设置 partition 数量,所有的 partition 被均匀地分布在集群的所有节点上。Producer 向 Broker 的 Topic 发布消息时,可以通过随机、哈希、轮训等策略选择将某条消息发布到某个 partition,消息被并行地发布到不同节点的 partition 中。Producer 和 Consumer 可以同时从多个 partition 上读写数据,提高了系统的并行处理能力。

Kafka 在创建 Consumer 时,设置其属于某个 Consumer Group。Consumer Group 之间完全独立,不同的 Consumer Group 可以同时消费一个 partition 的数据;Consumer Group 内的多个 Consumer 可以交错地消费一个 Topic 的所有 partition 的数据,减少了每个 Consumer 连接 Broker 的通信开销,Consumer Group 的设计有效地提高了消费消息的效率。

另外,Kafka 采用发布-订阅模式传递消息,Producer 只负责向 Broker 发布消息,Consumer 只负责从 Broker 消费消息,消息的发布与消费是异步的,极大地提高了 Kafka 的吞吐量。

(2)负载均衡

Kafka 使用 ZooKeeper 进行集群管理。Broker,Consumer和 Consumer Group 都在 ZooKeeper 中注册,增加或减少 Broker 和 Consumer 均会触发负载均衡。此外,Topic 与 Broker 的对应关系以及 partition 在集群内的分配也由 ZooKeeper 进行维护。

(3)可靠性

Kafka 从多个层面考虑了可靠性。

Producer 通过确认机制保证发布消息的可靠性。Producer 发布消息后,如果在等待时间内未收到确认消息,则重新发布消息。

Kafka 将消息持久化到本地磁盘,消息被消费后不会立即删除,通过设置消息存储时间或存储消息最大值,自动处理消息。消息在存在期间可以被多次消费,若 Consumer 发生故障,没有正常消费消息,可以根据 offset 查找到上一条消息,重新处理。

Broker 通过备份 partition 提高系统的可靠性。每个 partition 有多个备份,其分布在不同的服务器上,即使某个服务器发生故障,数据也不会丢失。同时,ZooKeeper 查找可用 Broker,并通知生产者和消费者,生产者和消费者转而使用其他 Broker,不影响其正常工作。

(4)可扩展性

Kafka 使用 ZooKeeper 保存元数据信息,进行负载均衡,提高系统的健壮性,在增加 Broker,partition 时,直接向 ZooKeeper 注册即可,不影响系统正常运行,使 Kafka 支持在线水平扩展,具有高可扩展性。

1.3　ActiveMQ

1.3.1　概述

ActiveMQ[18-23] 是 Apache 软件基金会的一个开源的消息中间件,由 Java 语言开发,是 JMS 消息通信规范的一个实现。ActiveMQ 的目标是在尽可能多的平台和语言上提供一个标准的,消息驱动的应用集成。ActiveMQ 支持多种语言和协议的客户端,支持的语言有 Java,C,C++,C#,Python,Ruby,PHP,Perl;支持的协议有 OpenWire[24],WS Notification[25],XMPP[26],AMQP 等。ActiveMQ 可以运行在 Java 语言所支持的平台之上。

1.3.2　架构分析

如图 5 所示,ActiveMQ 架构主要包括 3 层 5 个模块:传输协议、消息域、消息存储、集群(Cluster)和监控(Monitor)。

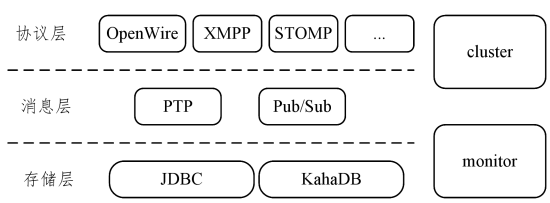

图 5　ActiveMQ 架构图[27]

传输协议:消息之间的传递需要协议进行沟通。ActiveMQ 支持多种通信协议,包括 TCP,NIO,STOMP 等;ActiveMQ 默认使用的应用协议是 OpenWire,端口号为 61616。

消息域:ActiveMQ 默认提供了 PTP 和 Pub-Sub 两种消息传输模式。

消息存储:消息存储到数据库或文件系统中,遇到机器故障时,信息不会丢失。

Cluster(集群):最常见的集群方式包括 network of brokers 和 Master Slave。

Monitor(监控):ActiveMQ 一般由 JMX 来进行监控。

ActiveMQ 支持 PTP 和 Pub-Sub 两种消息传递方式。ActiveMQ 提供通信接口、消息保存接口和网络服务接口。通信接口负责网络连接和消息传输;消息保存接口将数据持久化;网络服务接口支持存储转发、集群和命名服务等。

1.3.3 性能分析

(1)性能

ActiveMQ 支持集群模式,支持多种传送协议,如 TCP,SSL,NIO,UDP 等,且支持 JDBC 消息持久化和高效的日志系统。

(2)可靠性

ActiveMQ 支持内存、文件、内嵌数据库和外部数据库 4 种消息持久化方式,还提供失败重连机制、容错机制和多种恢复机制。另外,有很多种方法可以监控 ActiveMQ 不同层面的数据,包括 JConsole,JMX 等,保证了 ActiveMQ 消息的可靠性。

1.4 RocketMQ

1.4.1 概述

Apache RocketMQ[28-30] 是 MetaQ 的 3.0 版本,MetaQ 的全称为 Metamorphosis,是阿里巴巴公司开源的分布式的消息中间件。MetaQ 思路起源于 Kafka,MetaQ 1.x 和 MetaQ 2.x 均依赖于 ZooKeeper,但 RocketMQ 去掉了 ZooKeeper 依赖,采用自主研发的轻量级名称服务(NameServer)进行元数据管理。RocketMQ 由 Java 语言开发,采用发布-订阅模式传递消息,具有高效的水平扩展能力和亿级消息堆积能力,是一款高性能、高吞吐量的分布式消息中间件。RocketMQ 可以运行在 Java 语言所支持的平台之上。

1.4.2 架构分析

RocketMQ 架构如图 6 所示,主要由 NameServer,Broker,Producer 和 Consumer 4 个模块构成。NameServer 和 Broker 构成服务端,Producer 和 Consumer 构成客户端。

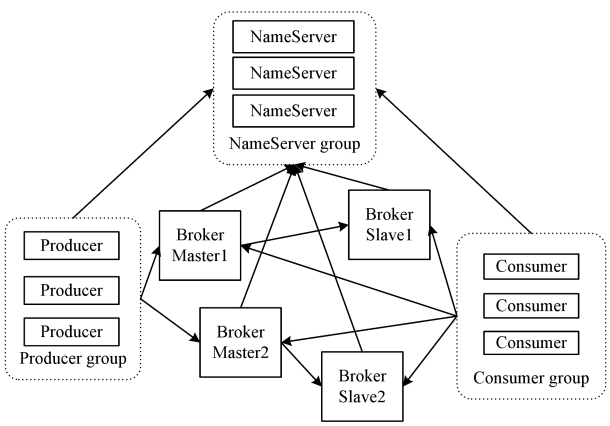

图 6 RocketMQ 架构图[31]

Broker:提供关于消息的管理、存储、分发等功能,是消息队列的核心组件。Broker 有 3 种集群部署方式:单 Master 部署、多 Master 部署、多 Master 多 Slave 部署。采用多 Master 多 Slave 部署方式时,Master 和 Slave 可以采用同步复制和异步复制两种方式。

Producer:即消息生产者,每个生产者都有一个 ID,多个生产者实例可以共用同一个 ID,同一个 ID 下所有实例组成一个 Producer group,每个 Producer 都属于一个 group。

Consumer:即消息消费者,每个消费者都有一个 ID,多个消费者实例可以共用同一个 ID,同一个 ID 下所有实例组成一个 Consumer group。在集群消息的配置下,集群内各个服务平均分配消息,当其中一台 Consumer 宕机,分配给它的消息会继续分配给其他的 Consumer。

NameServer:即注册中心,存储当前集群的所有 Broker 信息以及 Topic 与 Broker 的对应关系。Producer,Broker 和 Consumer 在启动时均向 NameServer 进行注册,NameServer 之间不通信。

NameServer 监听端口,等待 Broker,Producer,Consumer 连接。Broker 启动,与所有 NameServer 保持长连接,定时发送心跳包。心跳包中包含当前 Broker 信息、存储的所有 Topic 信息。Producer 启动,与 NameServer 集群中的一台 NameServer 建立长连接,获取当前发送的 Topic 与 Broker 的映射关系,然后与对应的 Broker 建立长连接,向 Broker 发消息。Producer 发送消息时,自动轮询当前所有可发送的 Broker。若消息发送成功,再次发送消息时,选择下一个 Broker,以实现将消息平均分布在所有 Broker 上。Consumer 启动,与 NameServer 集群中的一台 NameServer 建立长连接,获取当前订阅的 Topic 与 Broker 的映射关系,然后与对应的 Broker 建立长连接,开始消费消息。

1.4.3 性能分析

(1)效率

所有 Topic 数据同时只写一个文件,一个文件满 1G 后再写新文件,顺序写盘,使得发消息吞吐量大幅提高,为系统提供了的高并发读写服务。

(2)可靠性

Broker 将消息持久化到磁盘文件,每天定时清理消息,文件默认保留时间为 72h。Broker 集群采用主从模式部署时,备机实时从主机同步消息,如果其中一台主机宕机,备机提供消费消息服务,但不提供发布消息服务,不支持主从自动切换。若某个 Broker 宕机,Producer 仍然会向宕机的 Broker 发送消息,当消息发送失败后,会自动重发 2 次。如果仍发送失败,则抛出发送失败异常,自动轮询另外一个 Broker 重新发送。

NameServer 之间没有通信,没有频繁的读写,性能开销较小,稳定性高,单台 NameServer 宕机不影响其他 NameServer 和集群的正常工作。

(3)可扩展性

集群中增加 Broker 时,首先启动 Broker,然后向 NameServer 注册信息。Producer 和 Consumer 通过NameServer 获取 Broker 信息,与 Broker 建立连接,进行收发消息,不影响系统中其他 Broker 正常工作。

2 系统对比

经过对上述 4 种分布式消息系统的架构及性能分析,本节总结各类参考文献的要点,将其性能进行对比,结论如表 1 所列。

表 1　分布式消息系统的性能对比

系统名称	RabbitMQ	Kafka	ActiveMQ	RocketMQ
源码语言	Erlang	Scala	Java	Java
授权方式	开源	开源	开源	开源
支持的协议	AMQP	TCP	OpenWire,STOMP,REST,XMPP,AMQP	自己定义
客户端语言	Java,C,C++,Python,PHP,Perl 等	C,C++,Erlang,Java,Perl,PHP,Python,Ruby 等	Java,C,C++,Python,PHP,Perl,.net 等	Java,C++
负载均衡	支持	支持	支持	支持
持久化	内存、文件	磁盘文件	内存、文件、数据库	磁盘文件
事务	支持	不支持	支持	支持
集群	支持	支持	支持	支持
单机吞吐量	万级	十万级	万级	十万级
消息批量操作	不支持	支持	支持	支持
社区活跃度	高	高	高	中
管理界面	有	有	有	有
应用场景	对实时性、可靠性要求较高的场景	处理大量活跃的流式数据的场景	集成不同平台、不同语言应用的场景	对性能、事务性要求较高的场景

由于 Erlang 语言的特性,RabbitMQ 并发性能较好;单条消息无大小限制,能够保证消息顺序,支持异步发送消息;跨平台、支持多种客户端语言;有消息确认机制和持久化机制,存储可以采用内存或者文件,可靠性高;支持事务,不支持批量的操作;管理界面较丰富,在互联网公司也有较大规模的应用;社区活跃度高。RabbitMQ 在吞吐量方面稍逊于 Kafka;开发语言为 Erlang 语言,不利于做二次开发和维护;需要学习比较复杂的接口和协议,学习和维护成本较高。其适用于实时的、对可靠性要求比较高的消息传递。

Kafka 性能卓越,具有较高的吞吐量,单机写入 10 个字节大小的消息,TPS 约在百万条/秒;支持集群模式,支持批量操作,支持持久化,消息存储在磁盘上,拥有较高的可靠性;集群可以透明地扩展,对业务无影响;Kafka 每个 partition 的数据都会复制到多台服务器上,具有较好的容错性;有第三方管理界面 Kafka-Manager;在日志领域比较成熟,被多家公司和多个开源项目使用。Kafka 主要用于处理大量的活跃流式数据。

ActiveMQ 支持消息持久化,可以将消息存入文件或者数据库;支持对消费消息的反馈确认;支持多种 JMS 协议,如 OpenWire,Stomp REST 等;相关管理工具比较丰富,故障切换处理比较完善,水平扩展比较方便;开发活跃,使用范围广泛;提供 Web 管理界面。但是其社区活跃度不及 RabbitMQ 高,相比其他队列,性能并不是太好,不适合用于上千个队列的应用场景。

RocketMQ 已经在阿里巴巴公司大规模应用,具有高吞吐量、高可用性、适合大规模分布式系统应用的特点;对消息的可靠性、事务性做了优化;模型简单,接口易用;支持根据时间、消息 ID、消息内容查询消息;基于磁盘持久化存储,支持多主多从方案。但是其支持的客户端语言不多,目前支持 Java 和 C++,其中 C++不成熟;没有 Web 管理界面。

结束语　在服务端发送消息的性能上,Kafka>RocketMQ>RabbitMQ>ActiveMQ。RabbitMQ 对路由、负载均衡、数据持久化都有很好的支持,但是由于其支持多种协议,使得 RabbitMQ 变为一个重量级软件,更适合于企业级的开发。Kafka 是高吞吐量的消息系统,支持持久化,适合于处理活跃的流式数据、产生大量数据的互联网服务的数据收集业务。ActiveMQ 在集成不同平台、不同语言编写应用时,拥有巨大优势。RocketMQ 思路来源于 Kafka,改成了主从结构,在事务性、可靠性方面做了优化,在阿里巴巴集团内部有大量的应用,经受多次天猫双十一海量消息传输的考验。总体来说,电商、金融等对事务性要求很高的企业可以考虑 RabbitMQ 和 RocketMQ,对性能要求高的企业可考虑 Kafka,需要集成不同平台、不同语言的应用的企业,可以考虑 ActiveMQ。

除本文所述 4 种分布式消息系统外,ZeroMQ[32-34],Redis[35,36]也可以用于分布式系统中的消息传递。ZeroMQ 号称"史上最快的消息队列",其实质是一种基于消息队列的多线程网络库,对套接字类型、连接处理、帧、路由等底层细节进行抽象,提供跨越多种传输协议的套接字。Redis 是一个开源的、高性能的 key-value 形式的数据库,同时也提供了发布-订阅消息的功能,可以用于消息的传输。相比上述 4 种成熟的分布式消息系统,ZeroMQ 和 Redis 较为简洁,系统开发人员可根据其提供的发布-订阅消息传递机制,进行定制开发。

参 考 文 献

[1] Distributed Systems Concepts and Design . George Coulouris [M]. America:Addison-Wesley,2012.

[2] WOOD I. Distributed Message Transmission System and Method:WO,EP1477034[P]. 2004.

[3] GE Y,LIANG X X,PAN Z,et al. MESSAGE PARSING IN A DISTRIBUTED STREAM PROCESSING SYSTEM:U. S. Patent Application 15/258,629[P]. 2018-3-8.

[4] CONSULTANT M P C T. Chapter 3. Point-to-Point(PTP) and Point-to-Multipoint(PMP) Wireless Systems and Antennas[M]// Wireless Access Networks: Fixed Wireless Access and WLL Networks-Design and Operation. John Wiley & Sons,Ltd,2001:41-56.

[5] EUGSTER P T. The many faces of publish/subscribe[J]. ACM Computing Surveys(CSUR),2003,35(2):114-131.

[6] GUPTA A,SAHIN O D,AGRAWAL D,et al. Meghdoot:Content-Based Publish/Subscribe over P2P Networks[M]// Middleware 2004. Springer Berlin Heidelberg,2004:254-273.

[7] SURHONE L M,TIMPLEDON M T,MARSEKEN S F,et al. RabbitMQ[M]. Betascript Publishing,2010.

[8] VIDELA A,WILLIAMS J J W. RabbitMQ in action:distributed messaging for everyone[M]. Manning Publications Co.,2012.

(下转第 34 页)