可靠、可伸缩、实时分布式消息中间件

李碧[1,2,3]，刘俊朋[1,2]，沈岩[1,2],张鹏[1,2]，吴潇[4,*]，李高超[4]

(1. 中国科学院信息工程研究所，北京 100093； 2. 信息内容安全技术国家工程实验室，北京 100093；

3. 网络空间安全学院，中国科学院大学， 北京 100049; 4. 国家计算机网络应急技术处理协调中心,，北京 100029)

摘 要：

随着互联网的快速发展，业务数据规模的不断扩大，后台支撑的服务不断扩容，从单机到多机，从多机到集群。在面向集群服务的架构中，如何保证分布式环境下各业务系统之间关键数据的稳定传输和共享已成为非常迫切的问题。一个分布式中间件凭借其本身的平台无关性，松耦合性可以以有限的成本满足高可用、易扩展等需求，因此消息中间件必然是明天分布式架构的选择。针对目前无论商用的、还是开源的消息中间件都不能保证消息的可靠性，即消息的顺序和精确一次的消息传输语义，本文提出了一种可靠、实时的消息分发机制；针对目前大部分消息中间件只能支持水平扩展，不能支持垂直扩展的问题，本文设计并实现了一种既支持水平扩展又支持垂直扩展的可伸缩消息中间件，以不断满足大规模复杂网络环境下海量数据传输、高并发的用户实时处理等在线服务需求；针对多层次、有状态的存储节点管理，本文设计了一种自适应的全局资源调度算法，并基于此实现了一个全局资源管理模块。由此可知，我们设计并实现了一个高可靠、高性能、易伸缩的分布式消息中间件 Minos。最后的实验测试结果表明 Minos 系统的有效性。

关键词：消息中间件；分布式；高可靠；高性能；可伸缩

Reliable, scalable, real-time distributed message middleware

Bi Li[1,2,3], JunPeng Liu[1,2], Yan Shen[1,2], Peng Zhang[1,2],Xiao Wu[4,*], GaoChao Li[4,]

(1. Institute of Information Engineering, Chinese Academy of Sciences, Beijing 100093, China;

2. National Engineering Laboratory for Information Security Technologies, Beijing 100093, China;

3. School of Cyber Security ， University of Chinese Academy of Sciences, Beijing 100049, China;

4. National Computer Network Emergency Response Technical Team/Coordination of China, Beijing 100029)

Abstract: With the rapid development of Internet and then the continuous expansion of business data, the architecture storing and computing massive data expands unceasingly, from single to multiple machines, and then the cluster. In the cluster service oriented architecture, how to ensure the stable transmission and sharing of key data among different business systems in the distributed environment has become an urgent problem. A distributed middleware with its platform independent, loosely coupled can use limited cost to meet the demand of high availability and high salability, therefore the message middleware is the inevitable choice of distributed architecture tomorrow. Aiming at the fact that most of the message middleware cannot guarantee the reliability of the message, in other words, the order consistency and the exactly-once semantics of the message delivery, the paper presents a reliable and real-time message transmission mechanism; As for the fact that whether commercial or open source message middleware can only support broker's horizontal expansion, but cannot support its vertical expansion, this paper designs and implements a highly scalable distributed message middleware supporting horizontal expansion and vertical expansion ,to meet the data transmission demand of real-time online service with massive data and high concurrent user in the large-scale complex network environment; as for the management of multi-level storage nodes named broker, this paper designs an adaptive global resource scheduling algorithm, and realize a global resource management module based on it. Therefore, the paper designs and implements a reliable, real-time and scalable distributed middleware named Minos. The final experimental results show the effectiveness of the Minos system.

Key words: message middleware; distributed; high reliability; high performance; high scalability

本课题得到国家重点研发计划（2016YFB0801304）,国家自然科学基金项目（No.61402474,No.61402464），国家 242 信息安全计划（No.2017A018）资助。

1. 引言

过去几年间，伴随着互联网数据爆炸性增长，大数据迅速发展成为学术界、工业界甚至世界各国政府关注的热点。任何电子设备都是数据之源，只要连上网络，任何物品都可以实时向云端传输数据。因此大数据的时代已然来临。面对海量数据以及不断增长的业务规模，单机服务器存在瓶颈，逐渐不能满足海量日志存储、高并发用户访问的需求。因此后台架构不断扩容，分布式系统备受青睐。而分布式消息中间件[1][2][3][4]凭借其平台无关、松耦合、可扩展的特性，从无到有，如今已是分布式架构的一个关键组成部分。

在分布式系统架构中，消息中间件[3][4]主要有以下几个特性（1）快响应：提高某些紧急服务的响应时间，例如用户的订单提交请求，而对于某些从容业务，例如数据持久化、离线分析等，可以借用消息中间件异步提供服务；（2）流量削峰：面对电商等领域高并发的服务请求，消息中间件就像一个变压器，无论前端流量多大，消息中间件都可以输出平缓的流量，避免短时间内高并发流量压垮应用，导致服务不可用。（3）松耦合：时间解耦-服务调用者和提供者不需要建立连接，服务调用者和服务提供者可以不同时存在；空间解耦-服务调用者和服务提供者可以跨机房、跨城、甚至全球实时通信；应用解耦-应用松耦合的设计模式增加了服务的可靠性，松耦合的应用不会因为一个应用崩溃而导致另一个应用拒绝服务。

然而，目前大多数消息中间件是专为高吞吐量的消息传递、海量的消息堆积而设计的，并不能满足严格的消息高可靠[5][6][7]需求。而在某些特定领域，后端系统需要可靠、实时的消息传输。例如面对电商领域、金融领域等交易类场景，消息中间件必须保证消息传输的可靠性，例如订单消息不能丢失，也不能重复。因此本文提出了一种高实时、高可靠的消息分发机制。

随着数据源不断增加，消息中间件下游的消费者数量不断增长，业界的消息中间件通过水平扩展broker 节点来满足海量数据传输的业务需求，而且目前大部分开源系统[3][5][7][8]不支持垂直扩展。因此对于生产者与消费者之间距离较远，需要跨运营商、甚至全球通信的应用场景，如果使用目前的分布式消息中间件，即单层的 broker 架构，消费者访问单层的 broker 服务器获取数据有较大的延迟，不能保证消费者的服务质量。其次随着访问数据量的不断增加，骨干网的带宽容易产生瓶颈，影响消息传输服务。因此面对大规模的网络传输，目前消息中间件延迟高，不能满足实时传输的需求，所以本文设计了一个支持水平扩展、垂直扩展的可伸缩架构。为了支持对多层扩展后的有状态存储节点进行有效的资源管理，本文提出了一种资源调度算法，并基于此实现了一个全局资源管理模块，对存储节点的 CPU，磁盘等资源执行自适应的调度。

因此面对跨运营商、全球通信的场景，本中间件通过部署多层有状态的 broker 节点，并且最后一层 broker 节点部署在网络边缘，每一个消费者的请求通过全局资源管理模块智能路由到最合适的 broker 节点，由最合适的 broker 节点提供消息传输服务。所以可伸缩的架构设计不仅消除了骨干网络带宽瓶颈、而且节约了高成本的骨干服务器资源，最终降低了消费者的访问延迟，达到大规模网络环境下的实时消息传输。由此可知：本文设计并实现了一个一个高可靠、高性能、易伸缩的分布式消息中间件 Minos。而且本文的性能测试结果表明：单生产者时，Minos 消息传输的速度性和实时性最高。Minos 严格保证了消息传输过程中的顺序性和精确一次的传输语义。

本文的其他结构组织如下:第二部分讨论相关工作，第三部分描述了 Minos 的设计和实现，第四部分是性能评估结果，第五部分是结论。

2. 相关工作

我们研究了目前开源消息中间件的代表 Kafka、RocketMQ 和商用消息中间件的代表 Amazon SQS。这一小节从可靠性、可扩展性等方面对比这几个消息中间件。

Amazon SQS[4]是目前商业界应用广泛的消息中间件，但是 SQS 不能保证消息的可靠性。首先 SQS 不保证消息的顺序[9][10][11][12]，消费者接收的顺序不一定是生产者发送的顺序；其次 SQS 只提供至少一次的语义保障，不保证精确一次的消息传输语义，因此消费者可能收到重复的消息。而且 SQS 对消息的大小有限制，消息大小不能超过

本课题得到国家重点研发计划（2016YFB0801304）,国家自然科学基金项目（No.61402474,No.61402464），国家 242 信息安全计划（No.2017A018）资助。

256 KB。因此 Amazon SQS 不适合对数据可靠性要求较高，而且资金受限的应用场景。

Apache Kafka[3]是开源界高吞吐量的分布式消息中间件。但是 Kafka 同 SQS 一样，不能保证消息的可靠性。Kafka 只能保证一个分区内的消息顺序，但是不能保证分区间消息的顺序完全一致性，消费者接收到分区间的消息可能是乱序的。Kafka 支持最多一次和至少一次的消息传输语义[13][14][15]，但是不支持精确一次的消息传输语义。Kafka 只能通过水平增加 broker 节点的数目来支撑大规模数据的传输需求，而对于生产者与消费者之间距离较远，跨运营商、甚至全球的应用场景，表现出较大的延迟。因此 Kafka 适合于小规模网络下日志处理的应用场景，不适合对数据可靠性要求较高，而且消息传输网络规模较大的应用场景。

RocketMQ[2]是在阿里巴巴特定业务需求的驱动下应运而生的开源实时分布式消息中间件。但是 RocketMQ 和 Kafka 一样，为了追求消息的高吞吐和低延迟舍弃了消息的可靠性，不能达到消息的顺序一致性和精确一次的传输语义。而且 RocketMQ 和 Kafka 一样，不支持 broker 节点的垂直扩展[16][17][18]。因此 RocketMQ 和 kafka 一样，只能支持不关注乱序，不关心重复的应用场景，也难以支撑大规模网络下的实时消息传输[19][20]。

3. 系统设计与实现

本文对应的名词解释如表 1，分布式中间件架构如图 1。如图所示，该中间件既支持水平扩展，也支持垂直扩展。当数据量越多时，中间件水平扩展 broker 节点；当网络规模较大时，中间件垂直扩展 broker 节点，系统中每一个区域组成一个消费者集群，一个区域内可以有上千个消费者。假设根据消费者规模需求，本系统需要 3 层 broker 节点，4 个区域，1 个区域包含了 2 个消费者，则系统的逻辑拓扑如图 1 所示。

术语	业务	备注
Producer	消息中间件的生产者	Producer 生产和发送消息
Consumer	消息中间件的消费者	Consumer 接收和消费消息
Broker	消息中间件的存储节点	Broker 服务器提供了消息的接收、存储和转发服务，实现生产者和消费者的解耦
region	消费者集群	区域，根据消费者所在的位置划分

表1 名词解释

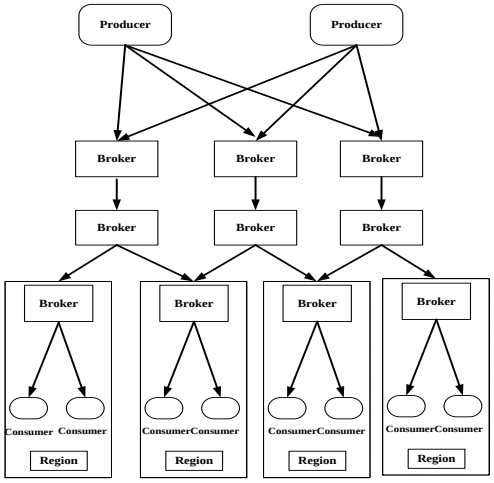

图1 系统架构图

3.1 系统框架总览

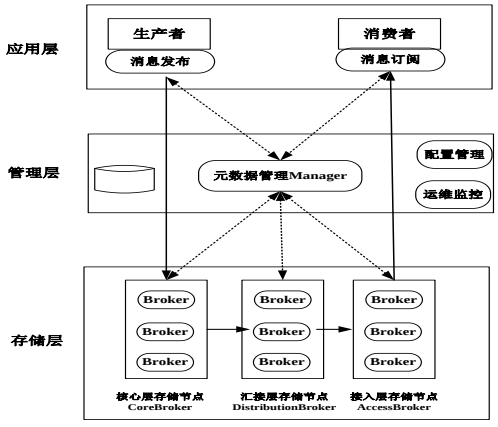

图2 系统框架图

如图 2 所示，本系统结构分为 3 个层次，下面对每个层次进行简单介绍：

应用层

本系统提供一个轻量级的客户端，包括发布、订阅接口。生产者发布消息；消费者订阅接口。

管理层

系统的管理者，监控应用层节点和存储层节点，保证系统正常稳定运行。

存储层

存储层是消息中间件的核心。为了实现垂直扩展，本架构设计了 3 种有状态的存储节点：核心层存储节点、汇接层存储节点、接入层存储节点。

1) 核心层存储节点：连接生产者。
2) 汇接层存储节点：为了区分与生产者建立连接的核心层存储节点和与消费者建立连接的接入层存储节点，本系统将中间可以任意扩展多层的存储节点命名为汇接层存储节点。如果网络传输规模较小，汇接层存储节点可以不部署。
3) 接入层存储节点：连接消费者。

3.2 详细模块设计

本小节详细介绍 Minos 的主要模块设计。

3.2.1 可伸缩的存储节点设计

如图所示，本小节介绍 3 种有状态存储节点的模块设计，其中汇接层存储节点的层次根据消费者规模、网络规模垂直扩展。

核心层存储节点设计

消息接收接口监听指定端口，等待生产者建立连接。一旦与生产者建立连接，消息接收模块便开始接收消息，将消息缓存至消息缓存模块。消息接收模块会对收到的消息进行哈希计算，并与传来的哈希值进行校验，如果两个哈希值不一样，则消息接收接口请求消息转发接口重发，通过消息校验保证消息在网络传输过程中的完整性。消息存储模块从消息缓存模块中取出消息，持久化消息至 KV 数据库。消息转发模块监听指定端口，等待下层存储节点建立连接。

汇接层存储节点设计

消息拉取模块通过资源管理模块（Manager）查询资源合适、网段合适的上层存储节点，然后向合适的存储节点请求建立连接，一旦建立连接，便开始与上层存储节点传输消息，并且对接收到的每条消息进行哈希校验。其中上层存储节点可能是核心层存储节点，也可能是汇接层存储节点，这取决于消费者规模和网络规模的需求。消息缓存模块、消息存储模块，消息转发模块与核心层的模块设计大同小异，因此不做具体介绍。

接入层存储节点设计

消息拉取模块、消息缓存模块、消息存储模块与汇接层的模块设计大同小异，因此不做具体介绍。消息转发接口监听指定端口，等待消费者建立连接。

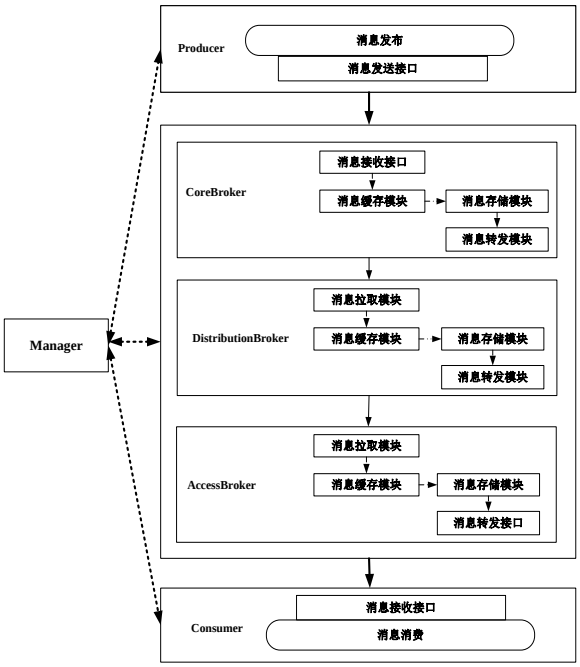

图3 系统详细模块设计图

3.2.2 全局资源管理模块设计

资源管理模块基于 zookeeper 开发，根据网络拓扑对多层次的存储节点进行统一资源调度，智能切换消息传输路径。本小节基下假设对全局资源管理模块流程进程详细说明：

- 本系统定义核心层存储节点的层次为 L，L 默认为 0。
- 本系统定义汇接层存储节点的层次为 L+1，随着网络规模扩大，汇接层存储节点将越来越多，每增加一层，层次加 1。

- 本系统定义接入层存储节点的层次为最后一层汇接层存储节点层次加 1，假设为 H。

本系统的资源管理模块如图 4 所示，生产者、消费者、存储节点与全局资源管理模块建立长连接，定时发送心跳，以便监控系统运行时的状态，评估系统的健康；资源管理模块支持运维人员对元数据的查询，包括消息的生产情况、消费情况等；资源管理模块支持异常报警，间接保证系统正确稳定的运行；全局资源管理模块存储存储节点的网络拓扑，资源持有情况等信息，然后根据存储的元数据信息，将存储节点和消费者的请求智能重定向到网段合适、资源合适的存储节点进行数据传输。其资源调度算法简述如下：

- 如果资源请求者是生产者，则请求与 level=L 的核心层存储节点建立连接。
- 如果资源请求者是中间的汇接层存储节点，假设存储节点自身的 level 为 S，则请求与 level=S-1 的存储节点建立连接。
- 如果资源请求者是消费者，则与 level 值最大的存储节点，即 level=H 的接入层存储节点建立连接。

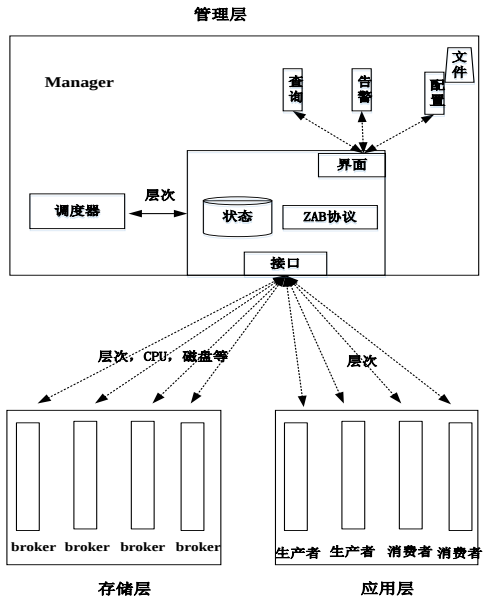

图4 全局资源管理模块

3.3 高可靠设计

可靠、实时的消息传输机制

为了达到高吞吐，本系统使用批的概念，将大量的读写操作聚合成一个批后再提交，消息的发送、接收、确认都是以批为单位。本系统基于批通信减少网络 IO 操作，压缩批消息等达到实时消息传输。本系统采用基于 ACK 的推拉结合的通信方式。消息发送者与消息接收者的通信过程如图 5。消息接收模块向消息发送模块请求建立连接，然后通过 ACK 进行可靠消息传输。如果消息接收模块收到并消费了 offset 为 0 到 ACK-1 号的批消息，而没有收到 offset 为 ACK 号的批消息或者消费 ACK 号的批消息不成功时，消息接收模块发送 ACK 消息，表示消息接收模块没有收到或者没有成功消费批号为 ACK 的批消息，此时需要消息转发模块重发该消息，则消息转发模块会重传 offset 为 ACK 号的批消息。如果消息转发模块没有收到 ACK 确认，消息转发模块不会继续发送批号为 ACK 的消息，以避免网络不可靠导致的消息顺序不一致的可能性。因此基于 ACK 的批消息重传机制可以保证消息接收模块接收的消息顺序与消息转发模块发送的消息顺序完全一致，而且通过基于 ACK 的批消息重传机制保证了至少一次的消息传输机制。

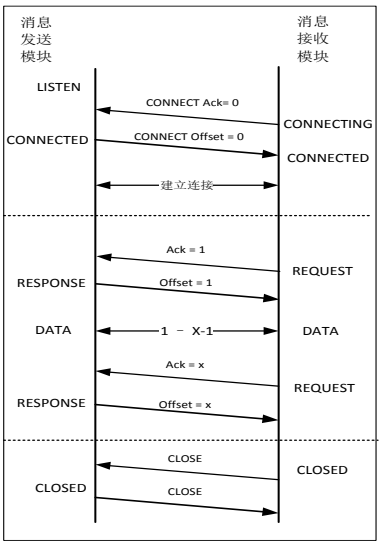

图5 基于 ACK 的推拉结合的通信方式

因为当消息接收模块收到批消息时根据消息 ID 查询 KV 型数据库，如果数据库中没有一样 ID 的消息，则插入 KV 型数据库，持久化该消息。如果收到的是重复的消息，则丢弃消息。因此通过基于 KV 可以保证消息不重复，而基于 ACK 的批消息重传机制保证了至少一次的消息传输机制，因此本系统可以支持精确一次的消息传输语义。

3.4 高可用设计

基于热备的容错机制

每一个存储节点部署主节点与从节点，主从节点之间使用同步复制。主从节点随机选择管理层节点中的任意一个节点建立长连接，并定时发送心跳。消息中间件的客户端（生产者和消费者）与消息中间件的会话状态转换如图 6。其会话的状态可以简述为：NOT_CONNECTED, CONNECTING, CONNECTED, CLOSED。一个会话从 NOT_CONNECTED 状态开始，客户端初始化后转换到 CONNECTING 状态（箭头 1）。正常情况下，客户端成功与 broker 建立连接，会话转为 CONNECTED 状态（箭头 2）。当客户端与 broker 断开连接或者无法收到 broker 的响应时，他就会转换为 CONNECTING 状态（箭头 3）。客户端尝试重新连接从服务器，当从服务器确定会话有效后，状态又回到 CONNECTED 状态。否则，会话超时，转换到 CLOSED 状态（箭头 4）。当然客户端如果显示关闭连接，也会转换到 CLOSED 状态（箭头 5）。

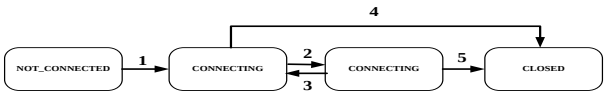

图6　会话状态转换图

由此可知，当主节点的 **CPU** 或者主板等关键设备损坏，发生单点故障时，从节点自动接管主节点的服务，秒级切换成为主节点。因此本系统可以容忍存储节点的单点故障，有强大的容错机制。而对于管理节点，因为管理节点是无状态的，所以管理节点可以集群部署，以保证服务的可用性。

4　性能评估

基于本文的架构，本文实现了一个单层 broker 的原型系统，从消息及时性、消息吞吐量、以及服务的可靠性三个方面与 kafka、RocketMQ 进行性能测试。服务端为单机部署，服务器硬件配置如表 2，服务器运行在操作系统 Red Hat Enterprise Linux Server release 6.4（Santiago）上。其中 Kafka 版本为 0.8.2，分区数为 1，副本为 1。RocketMQ 版本为 3.4.6，逻辑队列为 1，副本为 1，kafka、RocketMQ 都采用异步刷盘策略，生产者等所有的副本同步完毕后返回，三个消息中间件都只有一个消费者。

消息及时性

这一小节对比 Kafka、RocketMQ、Minos 发送并消费消息的及时性。三个中间件各自发送并消费 400 万条 100 字节的消息，分别记录 Kafka、RocketMQ、Minos 在不同的发送端并发数下，从生产者发送第一条消息到消费者接收完最后一条消息所消耗的时间。

如图 7 所示，横轴依次表示 1 个生产者并发，50 个生产者并发，200 个生产者并发时，Kafka、RocketMQ、Minos 完成 400 万条 100 字节的消息传输所对应的时间。当发送端的并发数较小时，Minos 的消息传输时间最短，消息及时性强于 Kafka、RocketMQ，其传输速度接近于 kafka 的 2 倍，RocketMQ 的 3 倍。但是随着发送端并发数的增加，kafka，RocketMQ 的消息传输速度有所提高，但是 Minos 并没有显著提高。因此当单生产者时，Minos 的消息及时性优于 Kafka、RocketMQ；当多个生产者并发时，Minos 和 kafka 不相上下，RocketMQ 的消息及时性最强。实验也间接证明：Minos 为了追求消息高可靠设计，损失了部分性能。

硬件	服务器配置
CPU	Intel Xeon CPU E5-4620 0 @ 2.20GHz（64 核）
内存	DDR3 126GB
硬盘	SATA 822GB

表2　服务器配置

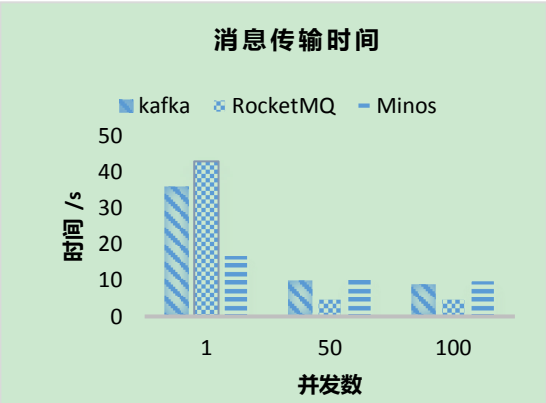

图7　消息及时性

消息吞吐量

这一小节对比 Kafka、RocketMQ、Minos 的最佳吞吐量。消息转发模块不断发送大小为 100 字节的消息，直到系统吞吐量不再上升，响应时间拉长，这时模拟服务端已达到性能瓶颈，分别记录 Kafka、RocketMQ、Minos 的最大 TPS 条数。如图 8 所示，Kafka 的吞吐量高达 19w/s，远超 RocketMQ 和 Minos，因此 Kafka 的消息吞吐量最大，Minos 次之，RocketMQ 最差。

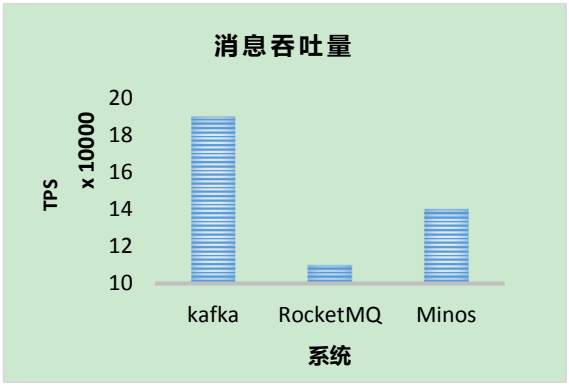

图8 **消息吞吐量**

消息的可靠性

这一小节测试 Kafka、RocketMQ、Minos 的消息可靠性。在本次实验中，消费者每接收一条消息睡眠 2ms 来模拟消费者的处理耗时，通过拔掉 Broker 所在的宿主机电源来模拟机器掉电故障，然后重启 Broker，查看消息的消费情况。

实验证明： kafka、RocketMQ 均出现了消息丢失和消息重复的情况，因为 kafka、RocketMQ 设置为异步刷盘策略，而 Minos 不会丢失消息，因为 Minos 消息存储模块采用同步刷盘策略。kafka、RocketMQ 会收到重复的消息，而 Minos 不会收到重复的消息，因为 Minos 的每层存储节点都实现了基于 KV 数据库的语义处理机制来消除消息重复的问题，保证了消息精确一次的语义。因此在 broker 宿主机掉电的场景下，Minos 的消息可靠性优于 kafka 和 RocketMQ。

5 结论

随着互联网的快速发展，消息中间件降低了业务系统和处理系统之间的耦合性，保证系统稳定运行，已经是分布式架构的一个关键组成部分。面对需要可靠、实时传输的电商、金融领域，本文设计了一种高实时，高可靠的消息分发机制，不仅保证消息实时传输，而且保证了消息的顺序一致性和精确一次的传输语义；面对大规模网络的消息传输架构，本文设计了一个支持水平扩展、垂直扩展的可伸缩架构达到实时消息传输。由此可知本文设计了一个消息高可靠、易扩展的实时分布式消息中间件 Minos，其与 SQS、kafka、RocketMQ 比较汇总如表 3。

特点	SQS	kafka	RocketMQ	Minos
持久化	是	是	是	是
传输语义	至少一次	至少一次 至多一次	至少一次	精确一次
消息顺序	不提供	不提供	不提供	提供
垂直伸缩	否	否	否	是
批量	是	是	是	是

表3 SQS、kafka、RocketMQ、Minos **比较汇总**

本文最后的性能测试结果表明：单生产者时，基于本文架构实现的单层消息中间件消息传输速度接近于 Kafka 的 2 倍，RocketMQ 的 3 倍，保证了消息实时性；严格保证了消息传输过程中的顺序性和精确一次的消息传输语义，保证了消息高可靠；对存储节点通过主从热备来支持容灾，保证了服务高可用。由于受到实验环境的限制，本论文只测试了单机存储节点的性能，而对于系统可伸缩，本文只是从设计原理上进行说明，并没有仿真跨城、跨运营商的大规模网络环境进行测试。因此未来的工作包括本系统和业界常用的消息中间件在大规模网络下的性能对比。

本系统对于并发度的支持不够，当多个生产者并发生产时，服务器存在瓶颈，因此未来的工作包括增加逻辑分区和设计可靠的负载均衡策略，提高生产者和消费者的并发度，消除发送端和消费端的瓶颈。

6 致谢

本课题得到国家重点研发计划（2016YFB0801304）,国家自然科学基金项目（No.61402474,No.61402464），国家 242 信息安全计划（No.2017A018）资助。

参考文献

[1]. https://kafka.apache.org/

[2]. https://github.com/alibaba/Rocket 消息中间件

[3]. Garg N. Apache Kafka[M]. Packt Publishing Ltd, 2013.

[4]. Mathew S, Varia J. Overview of amazon web services [J]. Amazon Whitepapers, 2014.

[5]. Zhang C, Liu X. HBaseMQ: A distributed message queuing system on clouds with HBase[C]//INFOCOM, 2013 Proceedings IEEE. IEEE, 2013: 40-44.

[6]. Kleppmann M, Kreps J. Kafka, Samza and the Unix philosophy of distributed data [J]. Bulletin of the IEEE CS Technical Committee on Data Engineering, 2015.

[7]. Tran N L, Skhiri S, Zim E. Eqs: An elastic and scalable message queue for the cloud[C]//Cloud Computing Technology and Science (CloudCom), 2011 IEEE Third International Conference on. IEEE, 2011: 391-398.

[8]. Patel D, Khasib F, Sadooghi I, et al. Towards in-order and exactly-once delivery using hierarchical distributed message queues[C]//Cluster, Cloud and GrID Computing (CCGrID), 2014 14th IEEE/ACM International Symposium on. IEEE, 2014: 883-892.

[9]. Sadooghi I, Wang K, Patel D, et al. Fabriq: Leveraging distributed hash tables towards distributed publish-subscribe message queues[C]//Big Data Computing (BDC), 2015 IEEE/ACM 2nd International Symposium on. IEEE, 2015: 11-20.

[10]. Kim B C, Jang C. A Distributed Instant Message System Architecture using Media Control Channel[J]. Journal of the Korea Institute of Information and Communication Engineering, 2016, 20(5): 979-985.

[11]. Wang G, Koshy J, Subramanian S, et al. Building a replicated logging system with Apache Kafka[J]. Proceedings of the VLDB Endowment, 2015, 8(12): 1654-1655.

[12]. Tarre M S, Rantzau R, Dutta D, et al. Multi-datacenter message queue: U.S. Patent Application 15/154,141[P]. 2016-5-13.

[13]. Liang Y, Tang X, Bing L, et al. Study on Service Oriented Real-Time Message MIDdleware[C]//Semantics, Knowledge and GrIDs (SKG), 2015 11th International Conference on. IEEE, 2015: 207-211

[14]. Hernández S, Fabra J, Álvarez P, et al. A reliable and scalable service bus based on Amazon SQS[C]//European Conference on Service-Oriented and Cloud Computing. Springer Berlin HeIDelberg, 2013: 196-211.

[15]. Sadooghi I, Palur S, Anthony A, et al. Achieving efficient distributed scheduling with message queues in the cloud for many-task computing and high-performance computing[C]//Cluster, Cloud and GrID Computing (CCGrID), 2014 14th IEEE/ACM International Symposium on. IEEE, 2014: 404-413.

[16]. Gross J N, Albert P. Message distribution system and method: U.S. Patent 9,171,089[P]. 2015-10-27.

[17]. Upadhyay U C, Katzer R D, Holmes G A, et al. Data message queue management to IDentify message sets for delivery metric modification: U.S. Patent 9,385,974[P]. 2016-7-5.

[18]. Li T, Keahey K, Wang K, et al. A dynamically scalable cloud data infrastructure for sensor networks[C]//Proceedings of the 6th Workshop on Scientific Cloud Computing. ACM, 2015: 25-28.

[19]. Bellows G H, Dale J N. Managing an out-of-order asynchronous heterogeneous remote direct memory access (RDMA) message queue: U.S. Patent 8,904,064[P]. 2014-12-2.

[20]. Dixon S, Huband S T, McKenna L R. Message queue transaction tracking using application activity trace data: U.S. Patent 8,683,489[P]. 2014-3-25.