# 跨域环境下的实时数据分发服务

李碧 [1), 2),3)] 张鹏 [2),3)] 杨嵘 [2),3)] 刘俊朋 [2),3)]

[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)

**摘 要** 随着互联网的快速发展，如何保证分布式环境下，各业务子系统间关键数据的可靠传输已经成为非常迫切的问题。针对复杂网络环境下的数据传输，OMG（Object Manage Group）提出了以数据为中心的发布/订阅模型-数据分发服务（Data Distribution Service）。本文研究了 DDS 规范，引入无状态的 BrokerSet，设计了一个既支持横向扩展又支持纵向扩展的数据分发服务；针对跨域场景下，多个 BrokerSet 集群的资源调度，本文设计了一种基于预配置的全局资源调度策略和一个基于临界回退的局部资源调度策略，基于此架构实现了一个满足跨域环境的实时数据分发服务 Minos，最后的实验结果表明了 Minos 的 TPS 高达 14w。

关键词 服务；数据分发； BrokerSet；易扩展；实时；

# Real-time Data Distribution Service in Cross Domain Environment

Bi Li[1, 2, 3], Peng Zhang[1, 2], Rong Yang[1, 2] JunPeng Liu[1, 2]

[1)] (Institute of Information Engineering, Chinese Academy of Sciences, Beijing 100093)

[2)] (National Engineering Laboratory for Information Security Technologies, Beijing 100093)

[3)]( School of Cyber Security ， University of Chinese Academy of Sciences, Beijing 100049)

**Abstract** With the rapid development of Internet, how to ensure key data’s stable transmission and sharing among different business subsystems in the distributed environment has become an urgent problem. Aiming at data distribution in complex network environment, OMG proposed a data-centric publish / subscribe model named data distribution service. This paper studies the DDS specification, and then introduces the stateless BrokerSet to implement a highly scalable data distribution service supporting horizontal expansion and vertical expansion. For resource scheduling of multiple BrokerSet in the cross domain scenarios, this paper designs a global resource scheduling strategy based on pre-configuration and a local resource scheduling strategy based on critical back. Based on this architecture, the paper implements a real-time data distribution service named Minos in the cross domain environment. Finally, the experimental results show that the Minos’s TPS is up to 14w

**Key words** Service; Data Distribution; BrokerSet; easily scalable; real-time.

## 1 引言

过去几年间，任何电子设备只要连上网络，都可以实时向云端传输数据，因此数据爆炸性增长，。面对海量数据以及不断增长的业务规模，单机服务器逐渐不能支撑海量数据存储、高并发用户访问，

———————————————————

收稿日期：年-月-日；最终修改稿收到日期：年-月-日 *投稿时不填写此项*. 本课题得到… …基金中文完整名称(No.项目号)、… …基金中文完整名称(No.项目号)、… … 基金中文完整名称(No.项目号)资助.作者名1(通信作者)，性别，xxxx年生，学位(或目前学历)，职称，是/否计算机学会(CCF)会员（提供会员号）,主要研究领域为*****、****.E-mail: **********. 作者名2（通信作者），性别，xxxx年生，学位(或目前学历)，职称，是/否计算机学会(CCF)会员（提供会员号）,主要研究领域为*****、****.E-mail: **********. 作者名3（通信作者），性别，xxxx年生，学位(或目前学历)，职称，是/否计算机学会(CCF)会员（提供会员号）,主要研究领域为*****、****.E-mail: **********.(给出的电子邮件地址应不会因出国、毕业、更换工作单位等原因而变动。请给出所有作者的电子邮件)
第1作者手机号码(投稿时必须提供，以便紧急联系，发表时会删除): … …, E-mail: … …*此部分6号宋体*

因此分布式系统应运而生。而分布式复杂环境下，各业务子系统间关键数据的稳定、可靠传输是分布式系统必须要面对并且解决的问题。

在分布式系统中，如果数据分发服务部署于单个数据中心，那么随着数据规模的不断扩大，单个数据中心无论是电力还是机架容量都会受限，因此数据分发服务存在瓶颈，需要跨多个数据中心来支撑流量，例如同城多中心，甚至异地多中心。然而目前业界大部分的中间件[1][2][3][4]不适合需要跨域部署的场景，如果将 Broker 集群部署于不同的数据中心，则 Broker 间同步延迟较大，不能保证消费者的服务质量。而且随着网络规模的增大，如果生产者与消费者在异地 IDC 机房，并且接入不同的运营商网络，那么其数据分发的延迟更加严重，而且数据分发的带宽成本较高。

因此本文引入无状态的 BrokerSet，设计了一个既支持横向扩展又支持纵向扩展的数据分发服务。该服务将数据缓存至各个核心层 BrokerSet，为生产者、消费者选择最近的 BrokerSet 数据中心提供服务，不仅消除了骨干网的带宽瓶颈、而且节约了高成本的骨干服务器资源，避免了异地、甚至跨运营商的数据传输延迟，最终达到跨域环境下的实时数据分发。

而跨域环境下，多个 BrokerSet 会带来路由不一致、数据不一致的挑战，例如假设生产者往 BrokerSet A 发布数据，接下来往 BrokerSet B 发送数据，消费者却因为路由错误从 BrokerSet C 消费数据，最终导致生产者和消费者的数据不一致。因此为了对多个 BrokerSet 集群进行路由与流量控制，本文设计了一种基于预配置的全局资源调度策略；为了对 BrokerSet 集群内的 Broker 节点进行资源调度，本文设计了一种基于临界回退的局部资源调度策略，并基于此实现了一个资源管理模块，保证数据分发服务高效、稳定运行。

由此可知，本文定义一个可扩展，与平台无关、与位置无关的基础服务模型，设计了一个跨域环境下的实时数据分发服务 Minos，来实现发送者、消费者间松散耦合。

本文的性能测试结果表明：单生产者时，与 Kafka 和 RocketMQ 相比，Minos 数据分发的实时性最强，其数据分发速度接近于 Kafka 的 2 倍，RocketMQ 的 3 倍。而且 Minos 严格保证了断电等突发故障环境下，数据分发过程中的时序一致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

我们研究了目前学术界和工业界通用的分布式消息中间件。这一小节从实时性、可靠性、可扩展性等方面对比这几个消息中间件。

HbaseMQ[5]是学术界首个基于 Hbase 云的高级消息队列。HbaseMQ 支持消息时序一致性，“至少一次”或“最多一次”的消息传递语义，而且对消息的大小没有限制。但是 HbaseMQ 不支持精确一次的传输语义，当发生网络故障时，数据会丢失与重复。而且如果生产者、消费者需要异地，甚至跨运营商传输数据，则数据分发延迟较大。

FaBRiQ[6][7]是学术界首个基于 DHT 的分布式消息中间件，由 P2P 组成 Broker 集群，强调易扩展，并且支持“精确一次”的语义保障。但是 FaBRiQ 不支持消息的时序一致性，消费者接收的消息顺序不一定是生产者发送的消息顺序。

HDMQ[8]是学术界首个层次型消息中间件，根据业务需求，HDMQ 跨区域扩展，适合跨域场景下的数据传输，而且支持消息的时序一致性和“精确一次”语义，但是对消息的大小有限制，当数据大小超过 512KB 时，不能提供数据分发服务。

Amazon SQS[4][9]是目前商业界应用广泛的消息中间件，但是 SQS 不保证消息的时序一致性，而且 SQS 只提供至少一次的语义保障，消费者可能收到重复的消息。同时 SQS 对消息的大小有限制，为 256 KB。另外如果将 SQS 部署于跨域环境下，用户需要根据网络拓扑，自实现资源调度模块。

Apache Kafka[3]是开源界高吞吐量的消息中间件。Kafka 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是乱序的。Kafka 支持最多一次和至少一次的数据传输语义，但是不支持精确一次的数据传输语义，当发生网络故障和进程故障时，消费者接收的数据可能丢失或者重复。Kafka 只能通过横向增加 Broker 节点数来满足不断增长的数据需求，而对跨域通信的场景，Kafka 表现出较大的延迟。

RocketMQ[2]是在阿里巴巴特定业务需求的驱动下应运而生的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，不能满足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，难以满足跨域环境下的实时数据分发。

因此无论是学术界、还是工业界，当发生异常时，其大部分消息中间件都会发生数据乱序，数据丢失或者重复，只适合于小规模网络下日志处理的应用场景。而且大部分中间件不能满足跨域场景下的实时数据分发。因此面向一些国际化的企业，设计一个跨域环境下的实时数据分发服务[10][11][12]迫在眉睫。

# 3 系统设计与实现

## 3.1 系统架构设计

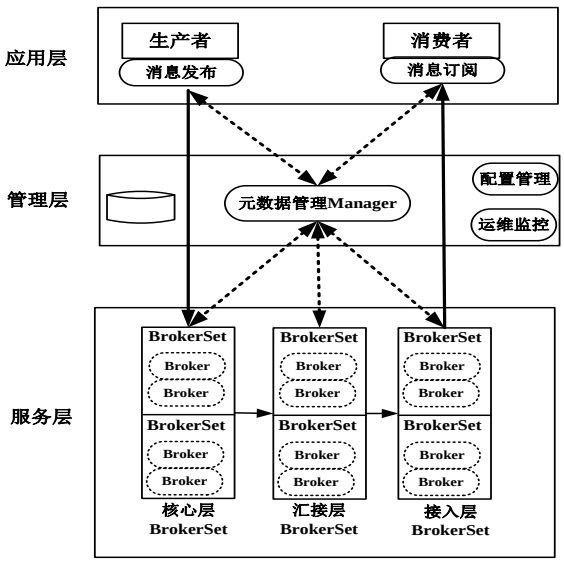


图 1 系统框架图

表 1 名词解释

| 术语 | 业务 | 备注 |
|---|---|---|
| Producer | 生产者 | 生产和发送数据 |
| Consumer | 消费者 | 接收和消费数据 |
| Broker | 存储节点 | 数据的接收、存储和转发 |
| BrokerSet | Broker 集群 | 由 Broker 组成的集群，通常部署在一个数据中心 |

如图 1 所示，本服务结构分为 3 个层次，下面对每个层次进行简单介绍。

**应用层**

本服务提供一种通用的应用层接口，规范、清楚地定义各业务子系统之间的数据通信；提供一个轻量级的客户端，包括发布、订阅接口。

**管理层**

生产者、消费者、BrokerSet 与管理者建立长连接，定时发送心跳，以便管理者监控系统运行时的状态，评估系统的健康；管理层支持对元数据的查询，包括消息的生产情况、消费情况等；管理层支持异常报警，间接保证系统正确、稳定的运行。

**服务层**

服务层是数据分发服务的核心。本文对应的名词解释如表 1。本文引入 BrokerSet[13][14]，设计一个数据分发服务架构，该架构既支持 BrokerSet 横向扩展，也支持 BrokerSet 纵向扩展。当数据量增加时，该架构增加 BrokerSet 内的 Broker 节点来支撑流量；当单个数据中心存在容量瓶颈，不能满足企业规模需求时，该架构横向增加同城或者异地的 BrokerSet 集群，实现横向扩展；当网络规模增大、消费者规模增大，数据需要异地，甚至跨运营商传输时，该架构可以纵向增加 BrokerSet 集群，实现纵向扩展。

因此本服务设计了 3 层 BrokerSet：核心层 BrokerSet、汇接层 BrokerSet、接入层 BrokerSet，其定义如下。

1) 核心层 BrokerSet：连接生产者。

2) 汇接层 BrokerSet：为了区分与生产者建立连接的核心层 BrokerSet 和与消费者建立连接的接入层 BrokerSet，本服务将中间可以任意扩展多层的 BrokerSet 命名为汇接层 BrokerSet。如果消费者规模、网络规模较小，汇接层 BrokerSet 可以不部署。

3) 接入层 BrokerSet：连接消费者。

## 3.2 业务逻辑设计

假设北京总公司需要将数据分发到各个子公司，各个子公司分别位于不同的城市、例如天津，杭州，深圳。假设由于环境因素，各个子公司分别接入不同的运营商网络，例如天津子公司接入电信网，杭州子公司接入移动网，深圳子公司接入联通网。假设根据数据规模，每一个 BrokerSet 数据中心由 4 个 Broker 节点组成。则根据企业网络规模，该服务需要部署 3 层 BrokerSet 集群。第一层包括 3 个 BrokerSet 集群。如图 2 所示，从左到右，第一层 brokerSet 分别托管于北京电信，北京移动，北京联通的 IDC 机房；第二层包括 3 个 BrokerSet 集群，分别托管于北京电信，上海移动，广东联通的 IDC 机房；第三层包括 6 个 BrokerSet 集群，分别托管于天津电信，天津电信，杭州移动，杭州移动，深

圳联通，深圳联通的IDC机房。其中第3层BrokerSet同城部署 2 个 BrokerSet 数据中心只是示例，具体视消费者规模而定。

基于层次型的架构，本服务会为发送者、消费者选择最近的 IDC 机房提供服务，避免了跨运营商、跨多个城市通信，实时性强。而且通过此服务架构，随着企业规模的不断扩展，即使子公司遍布全国，本服务也可以通过不修改原有的架构，增加第 2 层和第 3 层的 BrokerSet 集群，实现各个城市的实时数据分发。

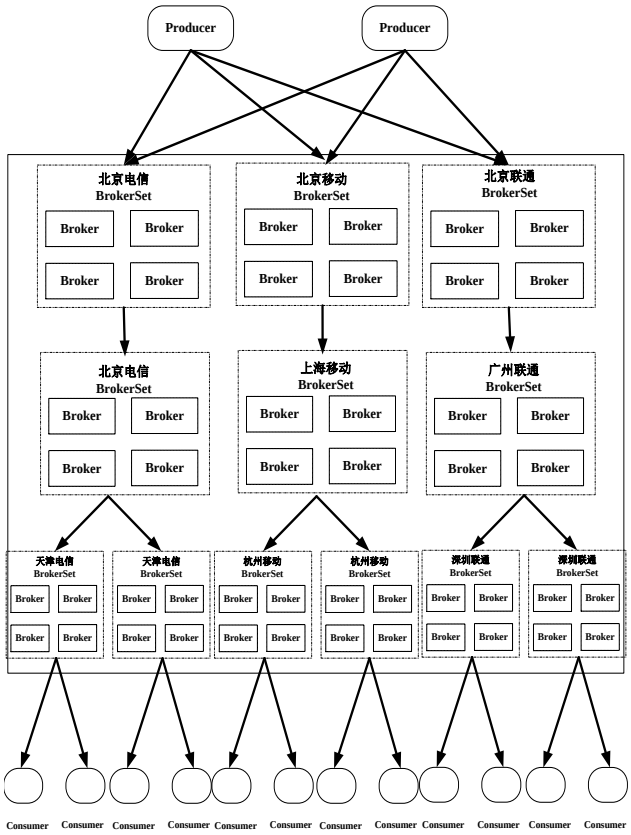


图 2 系统架构图

### 3.3 模块设计

本小节详细介绍 Minos 的主要模块设计。

#### 3.3.1 BrokerSet 模块设计

如图 3 所示，本小节简要介绍 3 种 BrokerSet 的模块设计。

**核心层 BrokerSet**

数据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，数据接收模块便开始接收数据，将数据缓存至数据缓存模块。数据接收模块会对收到的数据进行哈希计算，并与传来的哈希值进行校验，如果两个哈希值不一样，则请求数据重发。本服务通过数据校验保证数据在网络传输过程中的完整性。数据存储模块从数据缓存模块中取出消息，持久化数据至 KV 数据库，并通过 KV 数据库去重，避免数据重复。数据转发模块监听指定端口，等待下层 BrokerSet 建立连接。数据回收模块通过数据转发模块的下发状态回收内存中的数据。

**汇接层 BrokerSet**

数据拉取模块通过管理层查询资源合适、网段合适的上层 BrokerSet，然后向合适的 BrokerSet 请求建立连接，一旦建立连接，便开始与上层 BrokerSet 传输数据。其中上层 BrokerSet 可能是核心层 BrokerSet，也可能是汇接层 BrokerSet，这取决于消费者规模和网络规模的需求。数据过滤模块根据消息标签过滤，例如北京电信 BrokerSet 只存储天津电信需要的数据。数据缓存模块、数据存储模块，数据转发模块，数据回收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 BrokerSet**

数据拉取模块、数据缓存模块、数据存储模块、数据回收模块与汇接层的模块设计大同小异，因此不做具体介绍。数据过滤模块根据消息标签过滤，例如天津电信 BrokerSet 只存储天津电信需要的数据。数据转发接口监听指定端口，等待消费者建立连接。

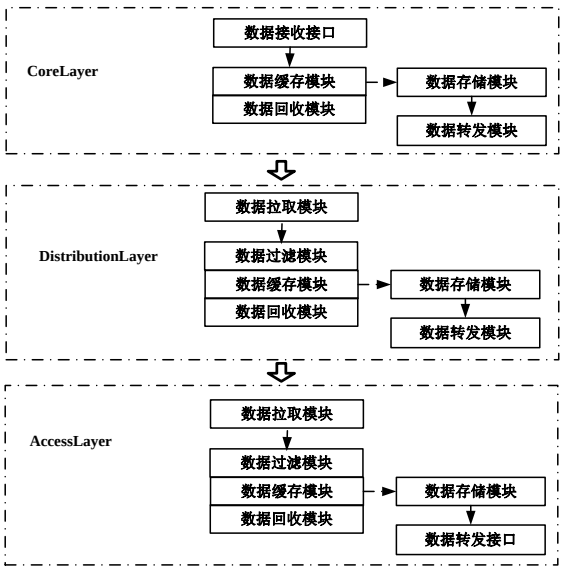


图 3 模块设计图

#### 3.3.2 资源管理模块设计

如图 4 所示，管理层的资源管理模块[15]基于 zookeeper 开发。资源管理模块根据存储的元数据信

息，将 BrokerSet、发送者、消费者的请求智能重定向到网段合适、资源合适的 BrokerSet 进行数据传输。基于预配置的全局资源调度策略根据网络拓扑对 BrokerSet 进行统一流量控制，智能切换数据分发路径。基于临界回退的局部资源调度策略根据 Broker 服务器资源进行资源调度，保证各 Broker 服务器负载均衡。资源管理模块通过文件接口统一配置层次等信息，对元数据进行管理。当发生异常时，资源管理模块在 zookeeper Web 界面告警，并支持在 Web 界面中查询各 Broker 服务器的数据分发情况以及 CPU、内存、磁盘等资源的负载情况。

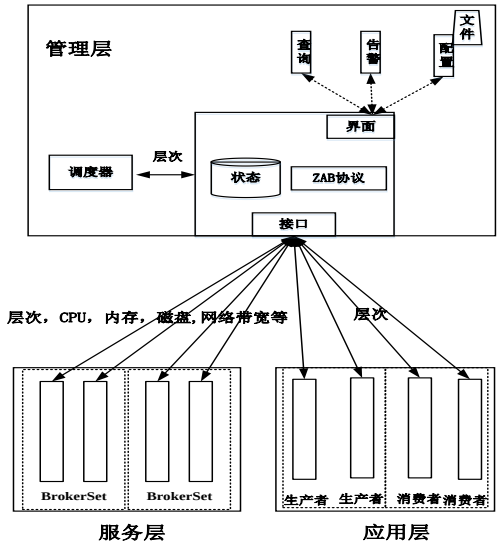


图 4 资源管理模块图

### 3.4 方法设计

3.4.1 基于预配置的全局资源调度策略

本小节基下假设对全局资源调度策略进行详细说明。

1) 本服务定义核心层 BrokerSet 的层次为 L，L 默认为 0，核心层 BrokerSet 存储于 zookeeper 的一个特定目录下。
2) 本服务定义汇接层 BrokerSet 的层次为 L+1。随着网络规模扩大，汇接层 BrokerSet 将越来越多，每增加一层，层次加 1，同理汇接层 BrokerSet 存储于 zookeeper 的一个特定目录下。
3) 本服务定义接入层 BrokerSet 的层次为最后一层汇接层 BrokerSet 层次加 1，假设为 H，同理存储于 zookeeper 中。

其基于预配置的全局资源调度策略简述如下：

1) 如果资源请求者是生产者，则从 level=L 的核心层 BrokerSet 中选择最近的数据中心建立连接。一般来说，就近选择可以通过预配置实现，例如天津分公司优先选择北京的数据中心建立连接；杭州分公司优先选择上海的数据中心建立连接。
2) 如果资源请求者是中间的汇接层 BrokerSet，假设 BrokerSet 自身的 level 为 S，则从 level=S-1 的 BrokerSet 中同理选择最近的数据中心建立连接。
3) 如果资源请求者是消费者，则从 level=H 的接入层 BrokerSet 选择合适的数据中心提供服务。

本服务的核心层 BrokerSet 会缓存所有业务的数据。因此如果任一层的 BrokerSet 发生机房断电等突发故障，甚至地震、火灾等不可抗故障，全局资源调度策略会为生产者选择下一个 BrokerSet 继续提供服务，保证服务高可用。

3.4.2 基于临界回退的局部资源调度算法

假设 BrokerSet 集群总共有 n 个 Broker 节点；本算法将 Broker 服务器资源抽象为四种资源：内存(M)，CPU(C)，磁盘(D)，网络带宽(N)。则第 i 个服务器的固有负载能力 LoadCapacity(i)如公式（一）所示，第 i 个服务器的当前负载 Load（i）如公式（二）所示，其中 $\alpha_1,\alpha_2,\alpha_3,\alpha_4$ 为加权因子。本系统线上测试，主要是磁盘资源不足，其次是网络带宽，其次是内存和 CPU。因此线上的系统 $\alpha_1,\alpha_2,\alpha_3,\alpha_4$ 依次取 10%，10%，50%，30%。

则第 i 个服务器的负载率 LoadRate(i)如公式（三）所示，其中 ρ 为临界因子，通常情况下 ρ=80%，此临界值取自学术界的标准；其中 γ 为回退因子，当服务器达到临界负载时，我们增加增加因子 β 来减轻服务器的负载，通过 γ 来调整递减的速度，避免服务器进入超载状态。通常情况下 $r$ =2。集群的平均负载率 AveraRate 如公式（四）所示。

$$LoadCapacity(i)=\alpha_1 M_{init}+\alpha_2 C_{init}+\alpha_3 D_{init}+\alpha_4 N_{init}$$

$$\alpha_1,\alpha_2,\alpha_3,\alpha_4\in[0,1]$$
$$\alpha_1+\alpha_2+\alpha_3+\alpha_4=1$$

------公式（一）

$$Load(i)=\alpha_1 M_{ret}+\alpha_2 C_{ret}+\alpha_3 D_{ret}+\alpha_4 N_{ret}$$

$$\alpha_1,\alpha_2,\alpha_3,\alpha_4\in[0,1]$$
$$\alpha_1+\alpha_2+\alpha_3+\alpha_4=1$$

------公式（二）

$$LoadRate(i)=\frac{Load(i)}{LoadCapacity(i)}+\beta$$

$$\beta=\begin{cases}0 & LoadRate(i)<=\rho\\ \frac{1}{r}(1-LoadRate(i)) & LoadRate(i)>\rho\end{cases}\quad p\in[0,1]$$

------公式（三）

$$AveraRate=\sum_{k=1}^{n}\frac{LoadRate(k)}{n}\qquad k\in[1,n]$$

------公式（四）

面对业务请求，本服务首先通过基于预配置的全局资源调度策略为生产者选择 BrokerSet 数据中心，然后其局部的动态资源调度策略为：首先通过一致性哈希算法为每一个业务分配 BrokerSet 集群中的一台 Broker 服务器，然后判断服务器的负载率是否超过了集群的平均负载率。如果没有超过，则将业务请求分配给该 Broker 节点；如果超过，则选择当前负载最小的 Broker 节点提供服务。

基于临界回退的局部资源调度算法，不仅避免了一致性 hash 没有考虑业务本身的资源需求不平衡，而且合理的为临界服务器分配合适的业务，避免 BrokerSet 集群中的 Broker 节点进入超载状态，系统出现抖动，影响数据分发服务性能。经实验验证，本服务负载较均衡。

# 4 性能评估

表 2 服务器配置

| 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 |

基于本文的架构，本文实现了一个数据分发服务的原型系统，从消息及时性、消息吞吐量、以及消息的可靠性[16][17]三个方面与 Kafka、RocketMQ 进行性能测试，最后与 Kafka 进行负载均衡测试。本实验中，Broker 服务端均为单机部署，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。Kafka 生产者等所有的副本同步完毕后返回，三个数据分发服务都只有一个消费者。

## 4.1 消息及时性

本小节对比 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。

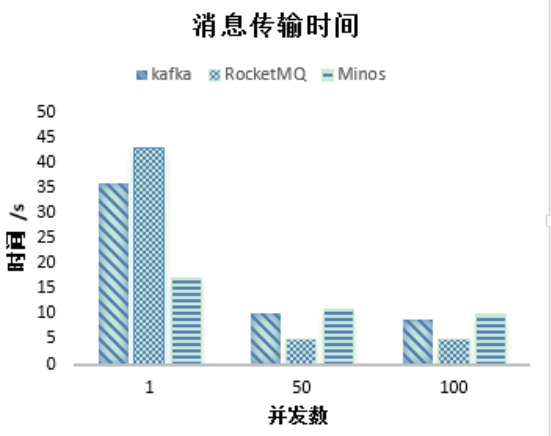


图 5 消息及时性

如图 5 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 字节的数据传输所对应的时间。当发送端的并发数较小时，Minos 的数据传输时间最短，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 的数据传输速度有所提高，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 为了追求数据持久化以及去重工作，损失了部分性能，而且对于并发场景，Minos 为了保证数据的全局顺序性，当生产者较多时，生产者同步的代价较大，因此性能下降较快，因此后期还需要一些并发的优化工作。

## 4.2 消息吞吐量

本小节对比 Kafka、RocketMQ、Minos 的最佳吞吐量。发送者不断发送大小为 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，本实验分别记录 Kafka、RocketMQ、Minos 的最大 TPS 条数。如图 6 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。

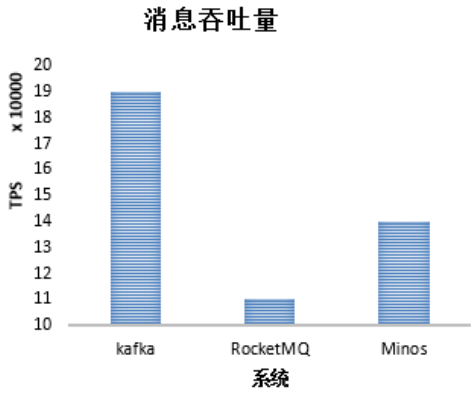


图 6 消息吞吐量

### 4.3 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。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。而且 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 Broker 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，保证了消息幂等。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。

### 4.4 消息负载均衡

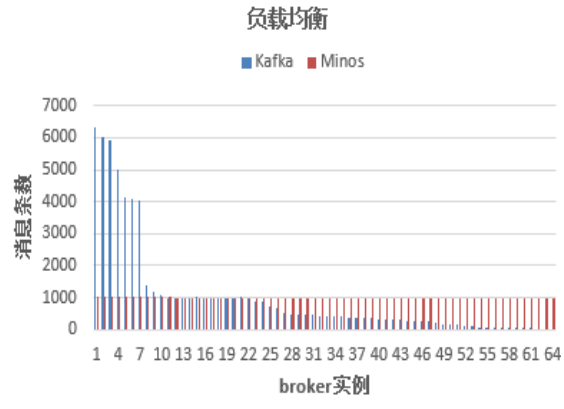


图 7 负载均衡测试

本小节测试 Kafka 和 Minos Broker 实例的负载状况。本实验部署了 64 个 Broker 实例，64 个生产者，每一个生产者生产 1000 条大小为 100 字节的消息，等生产者生产完毕后，实验统计了每一个实例上的消息数。

如图 7 所示，Kafka 最高负载的 Broker 实例为 6352 条，最低负载的 Broker 实例为 0 条，可见 Kafka 的 Broker 实例负载非常不均衡，这和 Kafka 使用轮询的负载均衡策略有关，而 Minos 每一个 Broker 实例上负载较均衡。Kafka 最高负载的 Broker 实例丢失了 350 条消息，64 个实例共丢失了 882 条消息，消息丢失的比例是 1.3%，而 Minos 负载均衡，无消息丢失。

## 5 结论与展望

面对跨域场景，本文设计了一个支持横向扩展、纵向扩展的易扩展架构。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数据分发服务实时性最强，而且受益于基于临界回退的动态负载均衡策略，Broker 实例的负载较均衡，系统的稳定性较高。

由于受到实验环境的限制，本论文只测试了单机 Broker 的性能，并没有仿真跨城、跨运营商的大规模网络环境进行性能测试。因此未来的工作包括本服务和业界常用的消息中间件在大规模网络下的性能对比。本服务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等优化策略，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。

**致 谢** 本课题得到国家重点研发计划（2016YFB0801304，2016YFB0800302），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] Kafka official website [online] 2013,https://Kafka.apache.org/

[2] RocketMQ[online]2016,https://www.oschina.net/p/rocketmq

[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. HBase消息中间件: A distributed message queuing system on clouds with HBase[C]//INFOCOM, 2013 Proceedings IEEE. IEEE, 2013: 40-44.

[6]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.

[7] Li T, Zhou X, Brandstatter K, et al. ZHT: A light-weight reliable persistent dynamic scalable zero-hop distributed hash table[C]//Parallel & distributed processing (IPDPS), 2013 IEEE 27th international symposium on. IEEE, 2013: 775-787

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

[9] 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.

[10]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.

[11]Lu Z Y, Guo Z B, Du X F, et al. A Method of Data Synchronization Based on Message Oriented MIDdleware and Xml in Distributed Heterogeneous Environments[C]//Proceedings of International Conf. on Artificial Intelligence and Industrial Eng. no. Aiie. 2015: 210-212.

[12]Kang W, Kapitanova K, Son S H. RDDS: A real-time data distribution service for cyber-physical systems [J]. IEEE Transactions on Industrial Informatics, 2012, 8(2): 393-405

[13]Cidon A, Rumble S M, Stutsman R, et al. Copysets: Reducing the Frequency of Data Loss in Cloud Storage[C]//USENIX Annual Technical Conference. 2013: 37-48

[14] Calder B, Wang J, Ogus A, et al. Windows Azure Storage: a highly available cloud storage service with strong consistency[C]//Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles. ACM, 2011: 143-157.

[15]Eisenbud D E, Yi C, Contavalli C, et al. Maglev: A fast and reliable software network load balancer[C]//13th USENIX Symposium on Networked Systems Design and Implementation (NSDI 16). USENIX Association, 2016: 523-535.

[16]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.....

[17]Calder B, Wang J, Ogus A, et al. Windows Azure Storage: a highly available cloud storage service with strong consistency[C]//Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles. ACM, 2011: 143-157.

## 附录 1

**李碧**，女，1994 年出生于湖南省娄底市 中国科学院大学硕士生，主要从事消息中间件，网络安全等领域的关键技术研究

**张鹏**，男，中国科学院副研究员，硕士生导师
主要从事大规模数据实时处理与网络安全等领域的关键技术研究

**杨嵘**，男，中国科学院高级工程师，硕士生导师
主要从事网络数据分析与处理，网络测量与行为分析等领域的关键技术研究

**刘俊朋**，男，中国科学院初级工程师
主要从事大规模数据实时处理与分布式等领域的关键技术研究

**Background**

This paper mainly studies data distribution service in the distributed environment. Most of academic current message middleware cannot satisfy message order consistence, "exactly once" semantic and no limit to the message size. While most commercial and open source message middleware in the industrial circles cannot adapt to cross-domain environment where data distribution’s delay is higher. Moreover, industrial message middleware is suitable for logging scenario which tolerates disorder and repeat. Therefore, this paper designs and realizes a scalable, platform-independence and position-independence data distribution service. And it provides a generic application layer interface to realize multi-type data distribution from multiple sources. The final test shows that the message distribution of Minos is the timeliest compared to Kafka and RocketMQ on behalf of open source distributed message middleware. Besides, Minos strictly guarantees the message order and "exactly-once" semantic when an outage failure occurs.

The research work is supported by the National Key R&D Program with No.2016YFB0800302, Key Research and

Development Plan with No.2016YFB081304, National Natural Science Foundation of China (No. 61402464), and Youth Foundation of National Computer Network Emergency Response Technical Coordination Center (No.2016QN-19.)