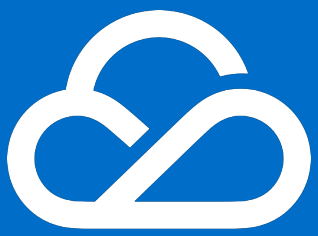
腾讯云消息队列 CMQ

案例分享

产品文档



腾讯云

【版权声明】

【商标声明】

【服务声明】

本文档意在向客户介绍腾讯云全部或部分产品、服务的当时的整体概况，部分产品、服务的内容可能有所调整。您所购买的腾讯云产品、服务的种类、服务标准等应由您与腾讯云之间的商业合同约定，除非双方另有约定，否则，腾讯云对本文档内容不做任何明示或模式的承诺或保证。

文档目录

案例分享

腾讯云 CMQ 与 RabbitMQ 的对比分析

RabbitMQ 是具有代表性的开源消息中间件，当前较多地应用于企业系统内，用于对数据一致性、稳定性和可靠性要求较高的场景中。

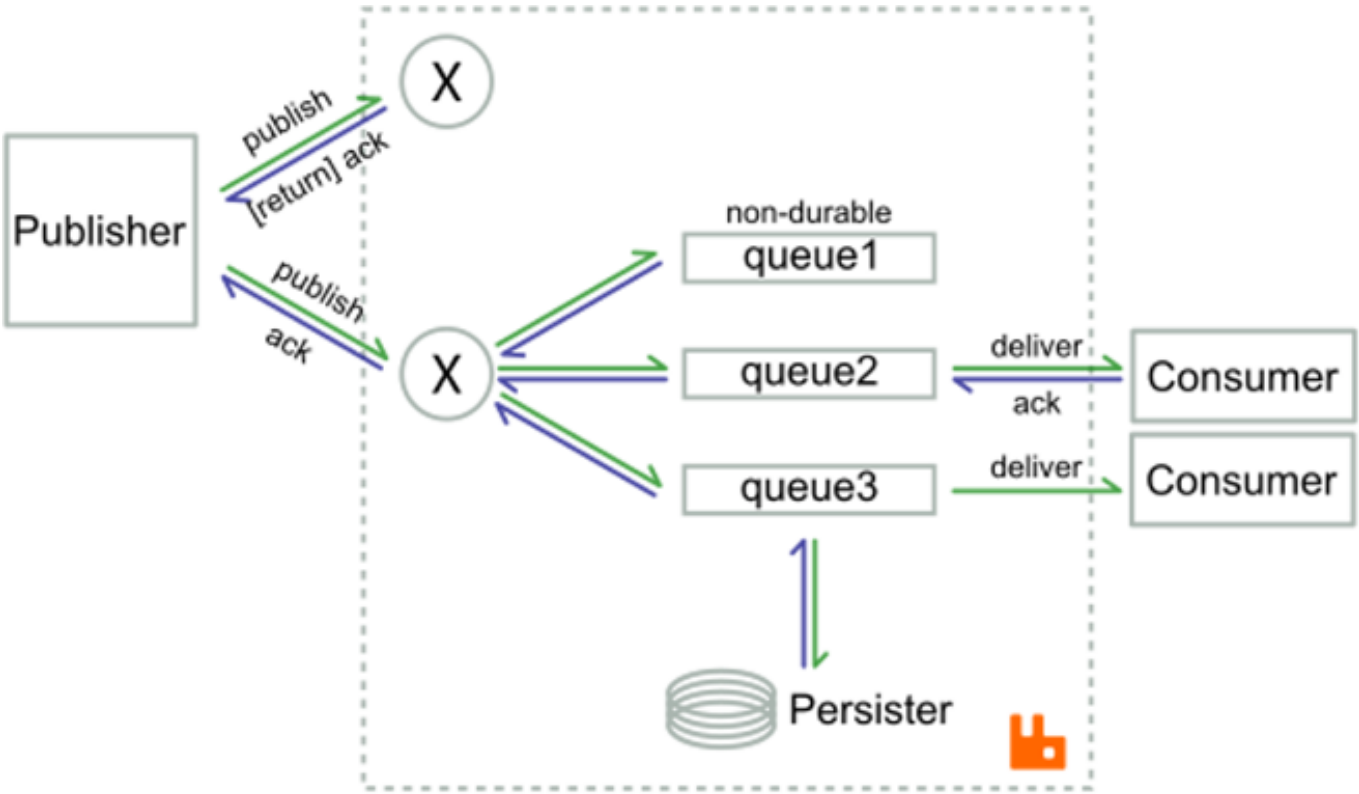
腾讯云 CMQ 基于 RabbitMQ/AMQP
高可靠的原理，并使用Raft协议重新设计实现，在可靠性、吞吐量和性能上有了大幅提升。

本文将重点介绍 RabbitMQ 的可靠性原理、腾讯云 CMQ 进行的改进以及性能对比。

1. RabbitMQ的消息可靠交付

1.1. 确认机制

网络异常、机器异常、程序异常等多种情况都可能导致业务丢失消息。对消息进行确认可以解决消息的丢失问题，确认成功意味着消息已被验证并正确处理。



RabbitMQ 使用生产消息确认、消费者确认机制来提供可靠交付功能。

- 生产消息确认：生产者向MQ发送消息后，等待MQ回复确认成功；否则生产者向MQ重发该消息。此过程可以异步进行，生产者持续发送消息，MQ将消息批量处理后再回复确认；生产者通过识别确认返回中的ID来确定哪些消息被成功处理。

- 消费者确认：MQ向消费者投递消息后，等待消费者回复确认成功；否则MQ重新向消费者投递该消息。该过程同样可以异步处理，MQ持续投递消息，消费者批量处理完后回复确认。

可以看出 RabbitMQ/AMQP 提供的是“至少一次交付”（at-least-once delivery），异常情况下，消息会被重复投递或消费。

1.2. 消息存储

为提高消息的可靠性，保证在 RabbitMQ 重启服务不可用时，要对收到的消息持久化写入磁盘。在收到消息时 RabbitMQ 将消息写入文件中，当写入达到一定数量或一定时间周期后 RabbitMQ 将文件落盘存储。

生产消息确认就是在消息落盘存储后，MQ向生产者回复已落盘存储的消息ID。

2. 腾讯云 CMQ 与 RabbitMQ 的对比

CMQ 与 RabbitMQ 的底层原理、实现方式有很多相似之处，但在更大程度上进行了升级改造：

2.1 功能升级

除了生产、消费确认机制，CMQ 还提供了消费回溯功能。

用户可以指定腾讯云 CMQ 保存生产消息一定天数，随后将消费回溯到该时间段内某一时间点，从该点开始重新消费。在用户业务逻辑异常时，以时间为起点的消息重放功能对业务恢复非常有帮助。

2.2 性能优化

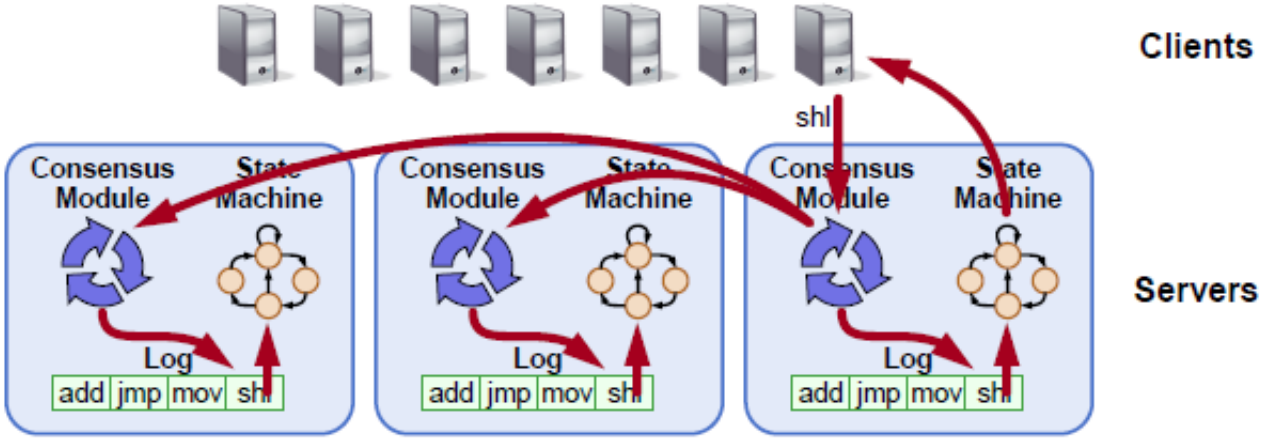
性能指标	说明
网络IO	CMQ 能够批量生产/消费消息，RabbitMQ 则不支持批量生产。在大量小消息场景中，CMQ

性能指标	说明
	具有更少的请求数和更低的平均延迟。
文件IO	CMQ生产/消费消息顺序写单个文件，并周期性落盘存储，能最大限度地充分利用文件系统缓存。RabbitMQ持久化消息机制为先入内存队列进行状态转换，然后写日志缓存，最后写消息文件和索引文件（索引文件为顺序写、消息文件为随机写），涉及三次IO操作，性能较差。
CPU性能	RabbitMQ的日志缓存和状态转换运算较为复杂，需大量耗用CPU 资源，性能较差。

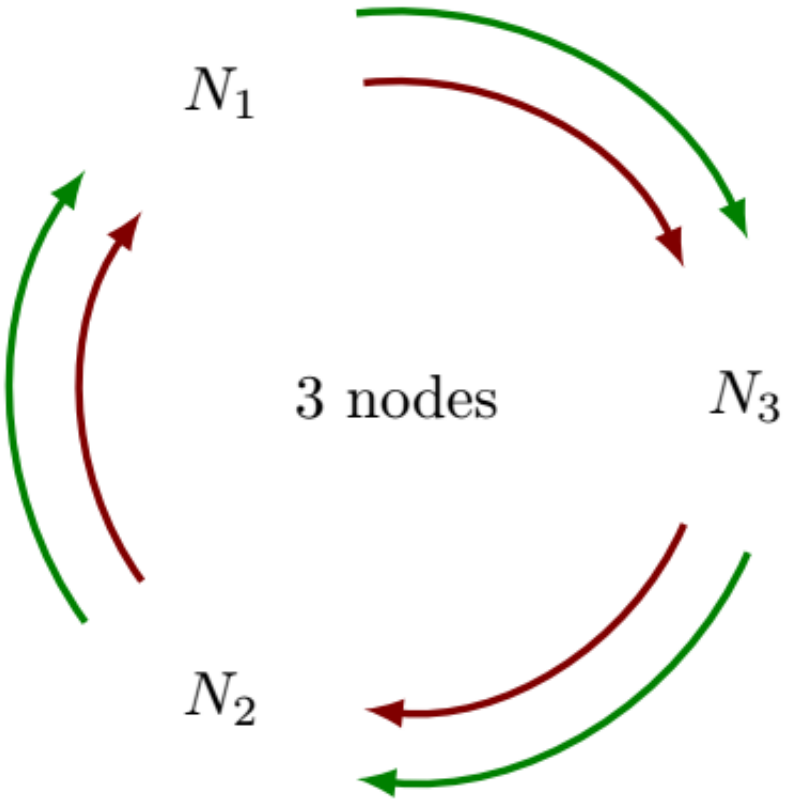
2.3. 可用性提升

CMQ 和 RabbitMQ 都能够使用多台机器进行热备份，提高可用性。CMQ 基于 Raft 算法实现，简单易维护。RabbitMQ 使用自创的 GM算法（Guaranteed Multicast），学习难度较高。

Raft协议中，Log复制只要大多数节点向Leader返回成功，Leader就可以应用该请求，向客户端返回成功：



GM可靠多播将集群中所有节点组成一个环。Log 复制依次从 Leader 向后继节点传播，当 Leader 再次收到该请求时，发出确认消息在环中传播，直至 Leader 再次收到该确认消息，表明Log 在环中所有节点同步完成。



GM算法要求Log在集群所有节点同步之后才能向客户端返回成功；Raft算法则只要求大多数节点同步完成。Raft算法在同步路径上比GM算法减少了近一半的等待时间。

2.4. CMQ 对比 RabbitMQ 的性能测试

测试场景如下：三台同样配置的机器组成一个集群，腾讯云CMQ、RabbitMQ均配置为镜像队列，数据均在三台机器上同步。腾讯云 CMQ 和 RabbitMQ都开启生产、消费消息确认机制。测试中的生产消息大小为1KB。

测试环境	环境说明
CPU	24核 Intel(R) Xeon(R) CPU E5-2620 v3 @ 2.40GHz
内存	64G
磁盘	12*2T SATA 12块SATA组成Raid0
网卡	10G
Linux版本	2.6.32.43
RabbitMQ版本	3.6.2
Erlang版本	18.3

测试数据如下：

QPS对比	仅生产	仅消费	同时生产/消费
CMQ	生产：6.8w/s	消费：9w/s	生产：3.6w/s 消费：3.6w/s
RabbitMQ	生产：1.25w/s	消费：2.6w/s	生产：0.85w/s 消费：0.85w/s

在高可靠场景中，CMQ 吞吐量优于 RabbitMQ的四倍以上。

第三方支付案例

与微信支付紧密合作的第三方移动金融支付解决方案提供商，如深圳威富通等，促进了全国各行各业的线下商铺，通过微信支付，提高效率，免除现金结算的低效率。支付系统主要架构如下：

1. 老百姓在便利店购物（如7-11）提交的支付请求，发送给微信支付，微信支付确认后会返回ACK

2. 返回的ACK确认后，微信支付系统会下发一条『订单支付成功的消息』，详细说明了消费的账户信息、时间、金额，终端信息等。该消息会发给威富通

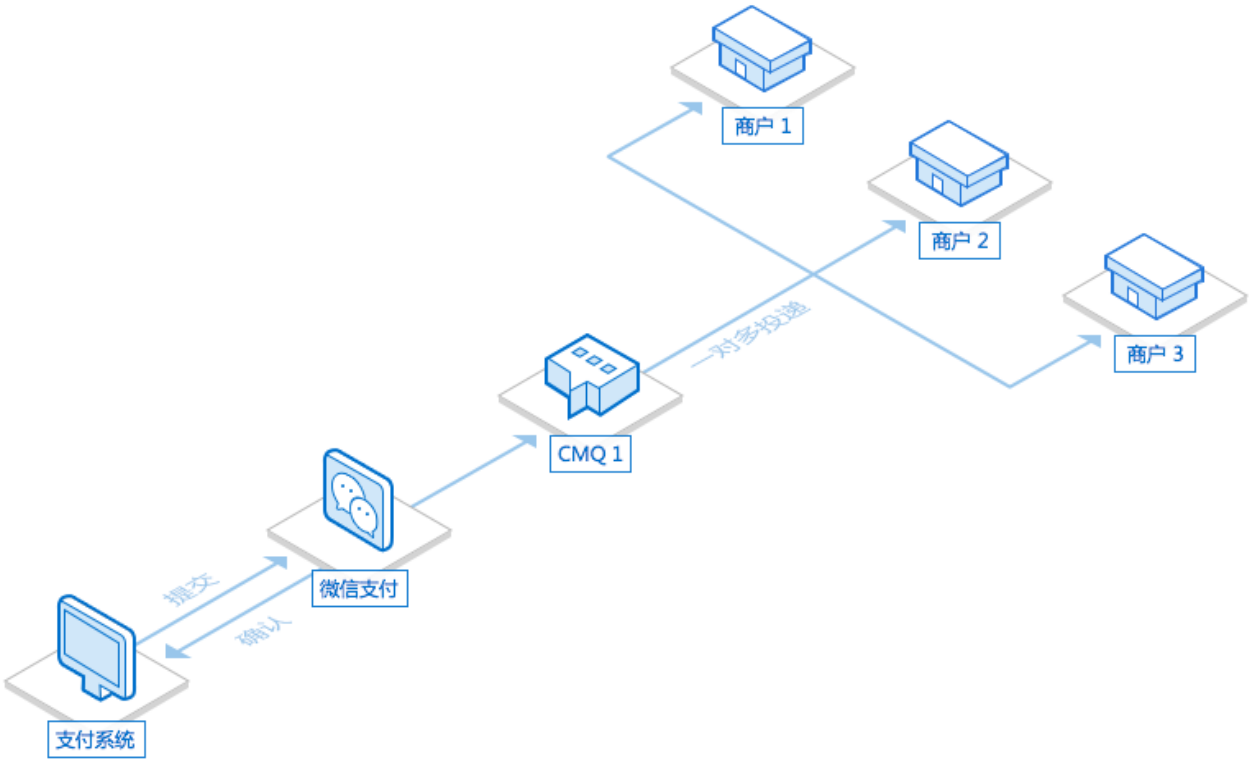
3. 威富通将该明细，写入CMQ，用作暂存。『订单支付成功的消息』作为威富通与商家（便利店）之间结算的重要凭证，必须可靠传递，保证不丢

4. 异步的，将CMQ内的『订单支付消息』，返回给多商家的服务器（便利店），这个返回的过程不需要及时，可以异步处理。具体怎么做呢？是将该消息写到Queue里，然后一个http代理，来拉消息。取出后，http发给商家。

5. 在未接入cmq之前，假如威富通通知商户失败，威富通会重新向微信支付发起请求，微信随后会再次将同样的『订单支付消息』投递给威富通。接入CMQ之后，从微信的角度来看，威富通系统的成功率提升不少，微信对其评级会提升（可靠性、信誉）

6. 最后，每笔『订单支付消息』，由另一个topic，不断向风控管理、活动管理、促销活动等系统投递。例如风控管理会持续分析topic投递的每一笔订单支付情况，当商家A在短时间内交易额大幅上涨时（刷单嫌疑），会用回调接口，禁止商家A的后续交易。

参考图示如下：


商户 1
商户 2
商户 3
一对多投递
CMQ 1
微信支付
提交
确认
支付系统

起点文学网案例

阅文集团旗下的起点文学网，使用CMQ满足了3个核心需求：

1、『仗义书财』的运营系统，里面抢红包月票的功能，消费者入账的时候是异步的。入账信息会先写到MQ里。 消费者过来拉，且消费者确认已成功消费后，回调接口把MQ里的信息删掉。

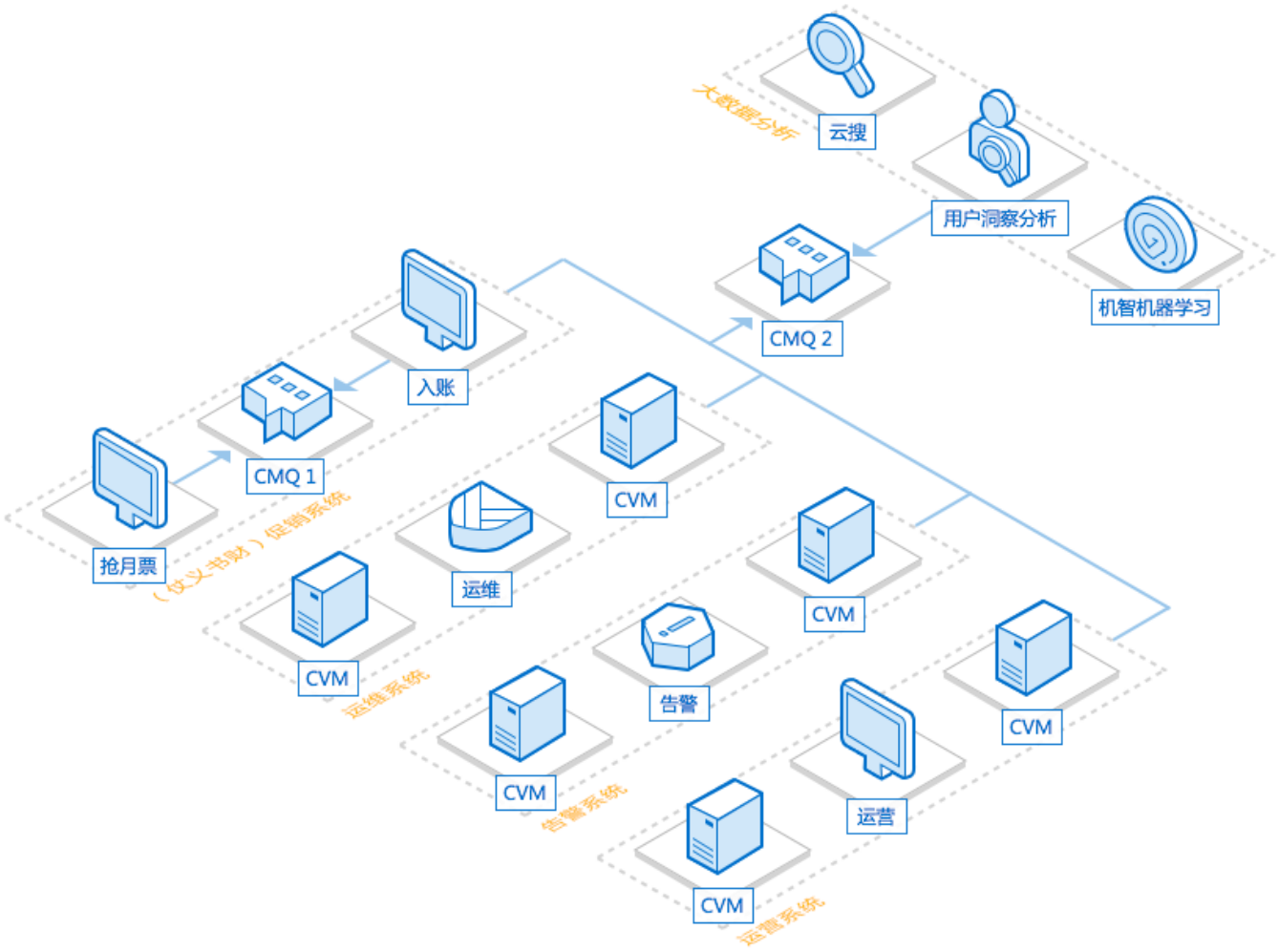
2、另一个场景是，起点文学网的各大系统，包括运维、告警、运营系统的日志流水，会先聚合到CMQ中，后端的大数据分析集群，会按处理能力，不断到CMQ中拉去，分析。CMQ理论上支持的消息堆积数量无上限，使用无后顾之忧。

3、提供类似于kafka的消息回溯能力。当业务成功消费，并删除消息后，使用消息回溯，可重新消费已删除的消息。可指定offset的位置进行调整。这便于起点文学网，做账单的对账、业务系统重试等。

起点文学的整体业务对CMQ的压力，API请求的QPS超过10万，全天请求量超10亿次，客户担忧如此大的业务压力，CMQ是否能稳定支持？

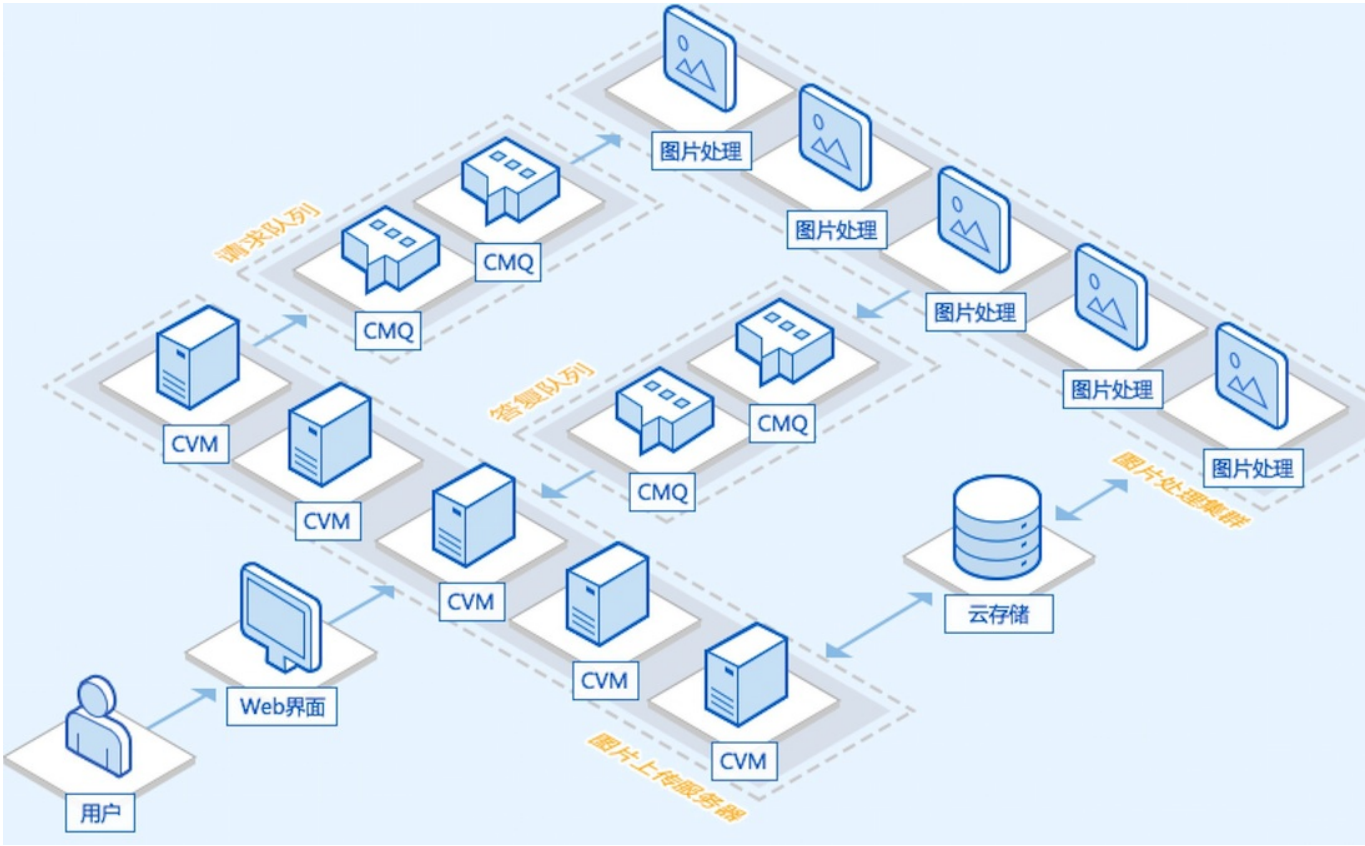
CMQ后端的集群对用户来说是透明无感知的，CMQ controller server 可根据集群的负载情况实时对queue进行调度搬迁。如果某个queue的请求量超过当前集群的服务阈值，controller server 可以将queue
路由分布到多个集群上来提高并发量，理论上可以达到无限的消息堆积以及超高的QPS。

参考图示如下：


大数据分析
云搜
用户洞察分析
机智机器学习
CMQ 2
入账
CMQ 1
抢月票
（仅义书财）促销系统
CVM
运维
CVM
运维系统
CVM
告警
CVM
告警系统
CVM
运营
CVM
运营系统

在线图片处理案例

某美图公司在腾讯云搭建了在线图片处理服务，该服务可以让用户上传照片，并指定需要对这些照片执行的操作，比如裁剪、红眼处理、牙齿美白、重新着色、对比度调节、生成缩略图等。用户上传图片后，提交任务，然后等待图片处理完，下载处理后的图片。不同的操作会耗费不同的处理时间，从几秒到几分钟不等，而且用户可能一次上传几张也可能是几十张甚至几百张图片，所以总的处理时间就跟上传的图片个数、图片的大小、用户选择的操作有关。



使用腾讯云 CMQ 实现了上述需求，用户的图片存储在腾讯云存储中（CBS/COS等），用户的每一个操作请求都会作为一个消息存入请求队列（Request
Queue）中，消息内容为：图片索引，由图片名称+用户请求的操作类型+图片存储的位置索引key等组成。

运行在CVM的图片处理服务从Request Queue中获取消息（图片索引），图片处理服务器从云端下载数据，并进行图片编辑，完毕后把处理结果发送到结果队列（Response Queue）中，结果图片存储到云存储中。流程结束，客户已将原图片、编辑处理后的图片，都存储在云端存储，可随时下载使用。

可扩展的、高可靠的进一步思考：

- 如果因为bug或其他原因导致图片处理服务暂时不可用。但是系统利用 CMQ

使得错误对用户透明，一方面用户可以继续上传照片，web server可以继续发消息到Request Queue，消息会被保存在队列中直到图片处理服务可用后取走；另一方面。图片处理服务在实现时不用记住崩溃前在处理的消息，而且其崩溃时处理的消息还可以被重新处理。因为CMQ提供的接收消息（包括接收顺序队列消息和接收并发队列消息）特性保证消息在接收后仍然在队列中，直到消息的接收者显式来删除它。本特性保证了图片处理服务与图片上传服务的解耦。

- 如果单个图片处理服务不能满足用户需求（用户虽然能够上传照片，但是却长时间拿不到处理的结果），利用CMQ并启动多个图片处理服务便可以满足不断增长的用户访问需求。CMQ的两个特性让这个需求成为可能：
  1) 单个CMQ队列是可以让多个server同时共享访问的（即发送消息、接收并发队列消息、删除并发队列消息功能）；
  2) 一个消息不会同时被多个服务接收，这是通过针对消息的短暂锁来保证的，消息的接收者可以指定消息被锁定的时间，接收者处理完消息需要主动删除消息，如果接收者处理消息失败，那么另一个服务可以在这个消息的锁失效后重新获得这个消息。

这两个特性保证了处理服务器的数量可以随着负载的变化而动态加减。

海量数据处理案例

大数据入门的第一步,就是将海量的数据进行挖掘分析,提炼有价值的结果,引导未来的商业模型, 如大众点评、滴滴打车在腾讯云已有深度的实践。例如：将微信、手机QQ上用户的热门分享的餐馆，实时反映到大众点评app的手机客户端上，推荐给消费者。

抽象出数据分析系统的特性，主要由以下模块组成：数据采集、数据接入、流式计算、离线计算、数据持久化。

1) 数据采集

负责从各节点上实时采集数据，选用开源的flume来实现。所有的业务服务器的日志等数据以漏斗形式的数据流入到CMQ管道。

2) 数据接入

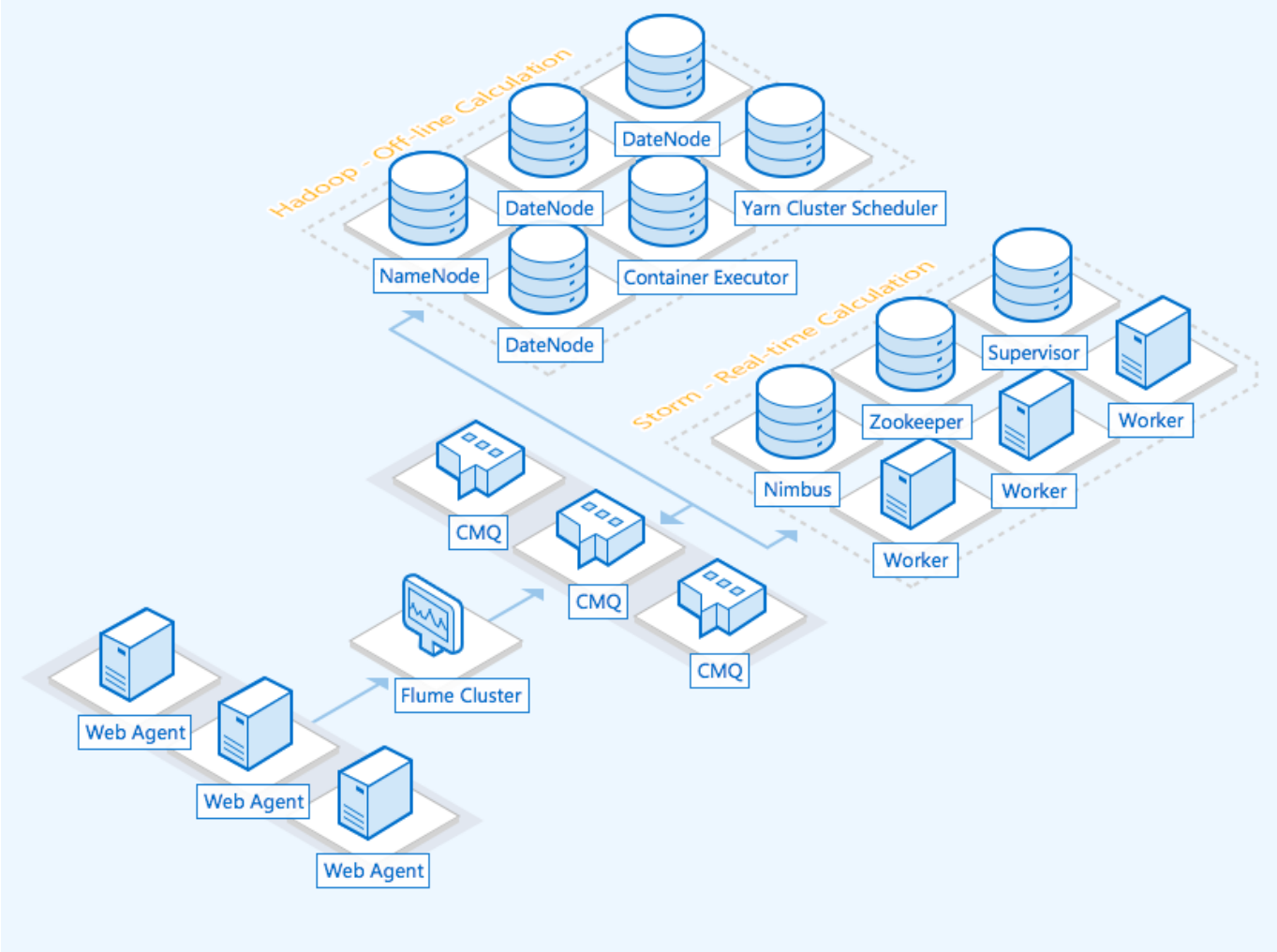
由于采集数据的速度和数据处理的速度不一定同步，为了保证日志写入、分析的稳定可靠，添加一个 CMQ 消息中间件来作为缓冲。

3) 流式计算、离线数据分析

对采集到的数据进行实时分析，选用apache的storm；离线数据分析基于Spark做长久的数据挖掘。

4) 数据输出

对分析后的结果持久化，可用腾讯云CDB、Mysql等方案。



在数据处理场景中，消息生产者是海量的日志数据的输入，在线分析的Storm集群是消息消费者。从经验来看，消息消费端Storm处理消息的业务逻辑可能很复杂（涉及到实时计算、数据流式处理、Topology的数据处理）。另一个问题则是消息消费端Storm出现故障的概率较高，导致短暂时间内无法消费或消费不畅。总而言之，消息生产者的效率是远远高于消息消费者的。

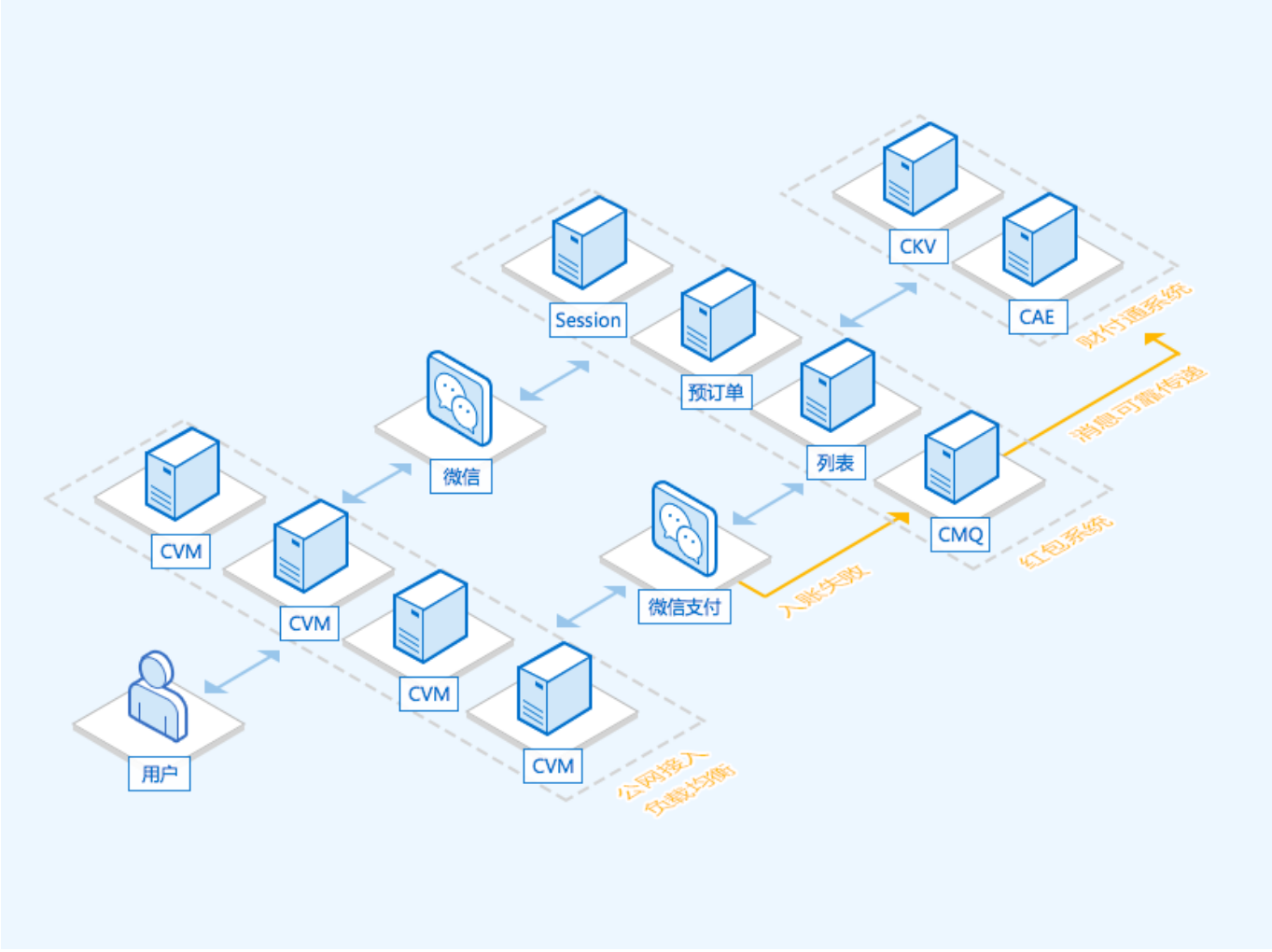
Push方式由于无法得知当前消息消费者的状态，所以只要有数据产生，便会不断地进行推送。在Storm集群处于高负载时，使用Push的方式可能会导致消费者的负载进一步加重，甚至崩溃。除非消息消费者有合适的反馈机制能够让服务端知道自己的状况。而采取Pull的方式问题就简单了许多，由于消息消费者是主动到服务端拉取数据，此时只需要降低访问频率。

腾讯云 CMQ 后续将推出topic主题模式，提供PULL/PUSH两种数据获取方式，CMQ作为生产者数据和消费者数据之间的缓冲区。它允许消费者可用并且准备好的时候才读取数据。缓解消息生产者与消息消费者之间不同步问题，从而在两者之间加了一层缓冲区。

春晚微信红包案例

春晚红包活动涉及四个大型系统的联动，包括微信、微信支付、红包系统和财付通系统。以下简单介绍各个系统：

- 红包系统：个人红包的发、抢、拆和列表查看；
- 财付通系统：包括支付订单、异步入账流水的高性能存储，用户余额和账单的实时展示；
- 微信接入：确保微信用户公网接入的质量；
- 微信支付：在线交易的入口。



类似红包系统的分布式事务是关注的热点。举一个典型的例子，『用户A给用户B发了10元的红包』，有以下步骤：

1) 从A帐号中把余额读出来

2) 对A帐号做减法操作（减10元）

3) 把结果写回A帐号中（一次确认）

4) 从B帐号中把余额读出来

5) 拆开A发送给B的红包，读出数值

6) 对B帐号做加法操作（加10元）

7) 把结果写到B帐号中

为了保证数据的一致性，上述步骤只有两种结果：都成功完成或者都不成功执行回滚。而且这个操作的过程中，对A、B帐号还需引入分布式锁机制来避免脏数据的问题。在微信红包这个庞大的分布式集群内，事情将变的异常复杂。

微信红包系统引入了腾讯云 CMQ
以避免分布式事务增加对系统的开销。同样A用户给B用户发10元红包的场景，下面介绍引入 CMQ 后的新策略。

- 在上述案例中的第七步，B 用户拆开了红包，红包里有 10 块钱。在做最后的入账操作时由于当天并发压力大，常出现入账失败的情况。
- 红包团队把入帐失败的请求，全部转入CMQ。当B用户更新账户余额失败时，手机客户端显示等待状态。随后账户系统将不断从 CMQ 重新拉取重试此更新操作。CMQ 保证了这 10 元的入账消息永远不丢，直至它被取出。
- 在除夕当天，用户红包的发、拆、入账等动作，转化为了十亿级别的海量请求。若使用传统的事务方式，会放大并发压力使系统崩溃。
- CMQ消息队列保证了红包消息的可靠存储、传递，实时写三份保证数据不丢。资金入账失败时可多次重试，避免失败回滚和频繁轮询数据库等传统方式的弊端。