RocketMQ 最佳实践

v3.0.0

2013/9/23

文档变更历史

序号	主要更改内容	更改人	更改时间
1	建立初始版本	誓嘉 vintage.wang@gmail.com	2013/9/23
2			
3			
4			
5			
6			
7			

目录

1 前言

本文档旨在描述 RocketMQ 如何使用，以及服务器集群的部署方式，面向应用方和运维人员。

2 Producer 最佳实践

2.1 发送消息注意事项

1. 一个应用尽可能用一个 Topic，消息子类型用 tags 来标识，tags 可以由应用自由设置。

   message.setTags("TagA");

2. 每个消息在业务层面的唯一标识码，要设置到 keys 字段，方便将来定位消息丢失问题。服务器会为每个消息创建索引（哈希索引），应用可以通过 topic，key 来查询这条消息内容，已经消息被谁消费。由于是哈希索引，请务必保证 key 尽可能唯一，这样可以避免潜在的哈希冲突。

   ```
   // 订单 Id
   String orderId = "20034568923546";
   message.setKeys(orderId);
   ```

3. 消息发送成功或者失败，要打印消息日志，务必要打印 sendresult 和 key 字段。

4. send 消息方法，只要不抛异常，就代表发送成功。但是发送成功会有多个状态，在 sendResult 里定义。

   - SEND_OK

     消息发送成功

   - FLUSH_DISK_TIMEOUT

     消息发送成功，但是服务器刷盘超时，消息已经进入服务器队列，只有此时服务器宕机，消息才会丢失

   - FLUSH_SLAVE_TIMEOUT

     消息发送成功，但是服务器同步到 Slave 时超时，消息已经进入服务器队列，只有此时服务器宕机，消息才会丢失

   - SLAVE_NOT_AVAILABLE

     消息发送成功，但是此时 slave 不可用，消息已经进入服务器队列，只有此时服务器宕机，消息才会丢失

5. 对于消息不可丢失应用，务必要有消息重发机制

例如如果消息发送失败，存储到数据库，能有定时程序尝试重发，或者人工触发重发。

2.2 消息发送失败如何处理

2.3 选择 oneway 形式发送

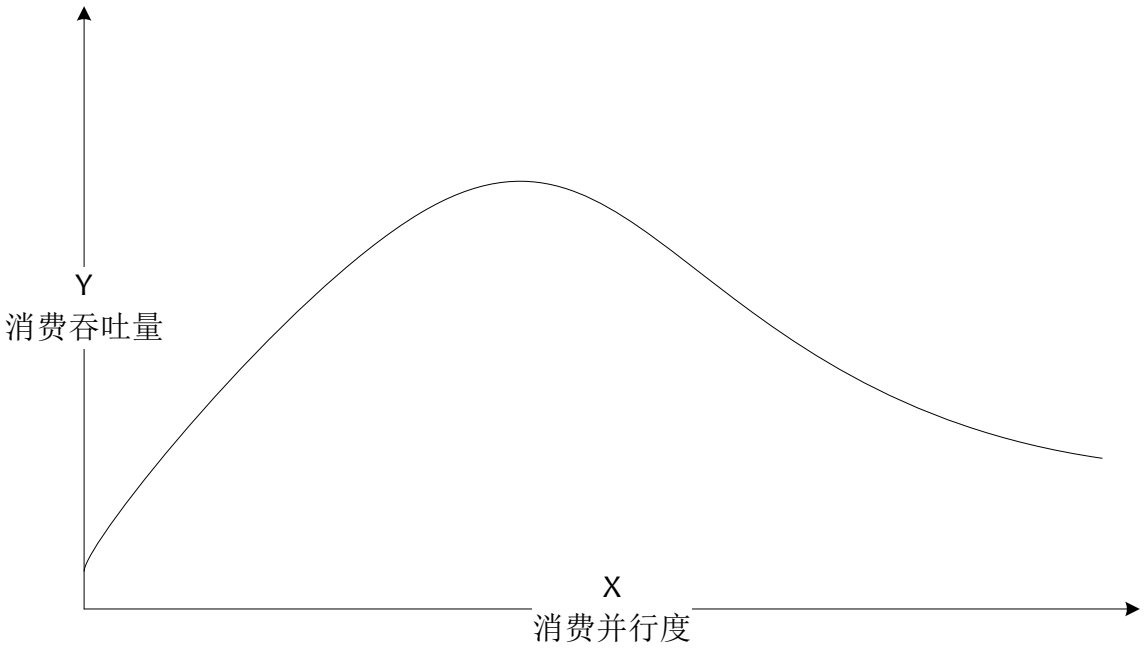
2.4 发送顺序消息注意事项

3 Consumer 最佳实践

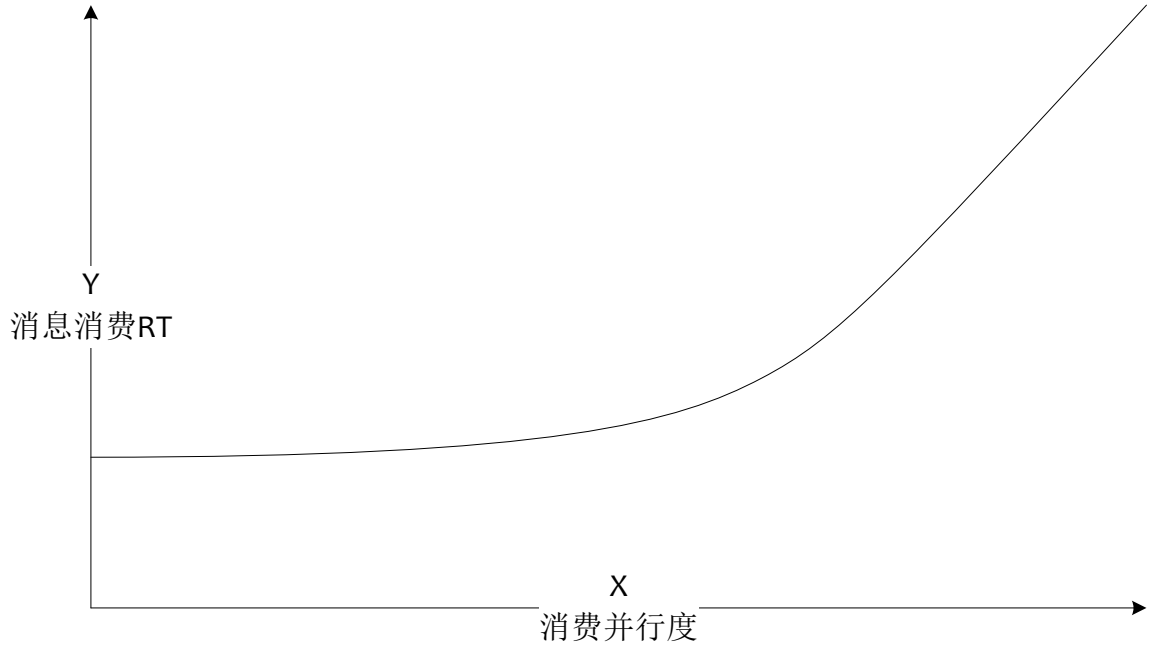
3.1 消费失败处理方式

3.2 消费速度慢处理方式

3.2.1 提高消费速度



3-1 消费并行度与消费吞吐量关系



3-2 消费并行度与消费 RT 关系

绝大部分消息消费行为属于 IO 密集型，及可能是操作数据库，或者调用 RPC，这类消费行为的消费速度在于后端数据库或者外系统的吞吐量，通过增加消费并行度，可以提供总的消费吞吐量，但是并行度增加到一定程度，反而会下降，如图所示，呈现抛物线形式。

所以应用必须要设置合理的并行度。

CPU 密集型除外。

3.2.2 跳过非重要消息

发生消息堆积时，如果消费速度一直追不上发送速度，可以选择丢弃不重要的消息

3.3 消费打印日志

如果消息量较少，建议在消费入口方法打印消息，方便后面排查问题。

3.4 利用服务器消息过滤，避免多余的消息传输

4 新上线一个应用需要注意什么