计算机系统应用 ISSN 1003-3254, CODEN CSAOBN
Computer Systems & Applications,2018,27(3):252−257 [doi: 10.15888/j.cnki.csa.006258]

E-mail: csa@iscas.ac.cn
http://www.c-s-a.org.cn
Tel: +86-10-62661041

RabbitMQ 小消息确认机制优化①

徐　震[1,2], 焦文彬[1]

[1](中国科学院 计算机网络信息中心, 北京 100190)
[2](中国科学院大学, 北京 100049)
通讯作者: 焦文彬, E-mail: wbjiao@cnic.cn

摘　要: RabbitMQ 的消息确认机制包括生产者确认 (confirm) 与消费者确认 (ack). 若消息不需要进行持久化, 生产者收到确认, 消息仍有可能丢失, 且生产者不知道消息已丢失, 消费者可能无法收到该消息. 若消息需要进行持久化, 生产者收到确认时, 消费者可能还没有收到消息. 本文对 RabbitMQ 的消息确认机制进行优化, 在收到消费者确认后向生产者发送确认消息, 消息丢失时由生产者重发消息, 减少消息确认过程记录的信息. 生产者收到确认即可保证消息被消费者成功接收, 提高了非持久化小消息投递的可靠性. 实验结果表明, 在客户端数量较少时, 该方法能够明显提高持久化小消息的发送速率.

关键词: 生产者确认; 消费者确认; RabbitMQ; 小消息确认; 性能优化

引用格式: 徐震,焦文彬.RabbitMQ 小消息确认机制优化.计算机系统应用,2018,27(3):252–257. http://www.c-s-a.org.cn/1003-3254/6258.html

Optimization of Tiny Message Acknowledgement in RabbitMQ

XU Zhen[1,2], JIAO Wen-Bin[1]

[1](Computer Network Information Center, Chinese Academy of Sciences, Beijing 100190, China)
[2](University of Chinese Academy of Sciences, Beijing 100049, China)

Abstract: Message acknowledgement mechanism in RabbitMQ includes confirm and ack. If the message doesnot need to be persisted, when the producer receives confirmation, the message may still be lost and the producer canot know that, so consumers may not receive the message. If the message needs to be persisted, when the producer receives confirmation, the message may be on the way to consumers. This study optimizes the message acknowledgement mechanism in RabbitMQ. After optimization, confirmation is sent to the producer after receiving the ack. The producer resends the message if the message is lost. Information that needs to be recorded is reduced during the message acknowledgement process. The producer receives confirmation after consumers receive the message successfully. The reliability of transient tiny messages delivery is improved. The experimental results reveal that the method can improve the sending rate of persistent tiny messages obviously when the number of clients is small.

Key words: confirm; ack; RabbitMQ; tiny message acknowledgement; performance optimization

RabbitMQ 是开源的基于 Erlang 的高效部署分布式消息队列, 实现了 AMQP 协议, 具有良好的可靠性、稳定性, 可运行在多种操作系统, 便于集群运行[1−6]. RabbitMQ 支持多种编程语言的客户端, 可以通过安装插件扩展功能. RabbitMQ 可以解耦应用程序, 将不同语言开发的程序粘合在一起, 在完全不同的应用之间共享数据.

Erlang 采用轻量级并发模型, 用于高并发、分布式“软实时系统”编程, 支持运行系统中的软件升级[7]. Erlang 程序简短紧凑, 采用函数式编程, 自动存储管理.

① 基金项目: 中国科学院“十三五”信息化专项 (Y72922802401)
收稿时间: 2017-06-27; 修改时间: 2017-07-10; 采用时间: 2017-07-20; csa 在线出版时间: 2018-02-09

Erlang 进程不共享内存, 进程间通信通过消息传递进行.

RabbitMQ 投递消息的速度受软硬件配置影响. 硬件方面有: 处理器、内存、磁盘、网络配置等; 软件方面有: 消息持久化机制、消息确认机制、交换器类型等. 要以高速度向消费者投递消息, 应尽可能保持队列为空.

对 RabbitMQ 进行优化有较多方法. 如对生产者确认机制的优化: 直接建立 channel 与消息存储之间的联系, 减少插入、删除、消息传递等操作, 可以大幅降低处理时间[8]. 还可以优化 topic 路由匹配算法; 批量发送消息; 优化消息持久化机制; 开启 Erlang HiPE 编译选项; 使用位运算等.

生产者发送一条消息到 broker, 消息可能被 n 个消费者接收. 同时启用生产者确认与消费者确认, 生产者仍无法获知 n 个消费者是否全部接收到消息. 本文对小消息情况下 RabbitMQ 的确认机制进行优化, 在 broker 收到 n 个消费者的确认消息后, 向生产者发送确认消息. 生产者收到确认消息则表明消费者已成功接收到消息. 若消息丢失, 由生产者负责重发消息. 对不同生产者、消费者、队列数量的情况进行测试, 分析比较优化前与优化后的持久化小消息发送速率.

1 RabbitMQ 架构与相关模块简介

1.1 RabbitMQ 架构

如图 1 所示, 生产者发送消息到交换器, 队列通过路由键绑定到交换器, 根据交换器类型与路由键将消息路由到队列, 消费者从队列接收消息. 常用的交换器类型有 direct、fanout 和 topic. 对于 direct 交换器, 如果路由键匹配, 消息就被投入相应的队列; fanout 交换器将收到的消息广播到绑定的队列; topic 交换器对路由键进行模式匹配, 消息被路由到匹配的队列. 消费者通过 basic.consume 命令自动从队列获取下一条消息; 通过 basic.get 命令获取单条消息. 队列具有多个消费者时, 采用 round-robin 方式向消费者发送消息. broker 是消息队列服务器实体, 一个 broker 中可以有多个虚拟主机.

1.2 相关模块功能

1) channel 接收 reader 解析的来自客户端的协议帧; 使用 writer 向客户端发送帧; 路由消息给队列进程; 处理 AMQP 方法; 发出 AMQP 命令. 一条 TCP 连接中可以有多个 channel.

2) 支持队列 (backing queue, BQ), 一般情况下默认为 rabbit_variable_queue. 队列进程使用 BQ 实现队列功能. 队列中消息具有 4 种状态: alpha、beta、gamma、delta. 持久化消息只可能处于 alpha、gamma、delta 三种状态之一. BQ 具有 5 个内部队列: q1、q2、q3、q4、delta. q1 和 q4 中只有 alpha 状态的消息; q2 和 q3 包含 beta 和 gamma 状态的消息; delta 队列不在内存中, 只有 delta 状态的消息.

3) 队列索引 (queue index) 用于在磁盘上记录队列中消息的顺序. 每个队列有一个队列索引. 消息依次被发布、投递、确认. 发布记录包括消息 ID、消息在队列中的序列号等内容. 发布记录也可能包括完整的消息. 投递和确认记录只包括消息在队列中的序列号. 队列索引使用日志文件 (journal) 避免过多磁盘寻址. 日志文件具有固定的长度, 默认为 32 768, 由 queue_index_max_journal_entries 参数配置.

4) 消息存储 (message store) 用于将消息写入磁盘或将消息从磁盘加载到内存. 存储的消息是引用计数的, ID 相同的消息多次写入时只会存储一次.

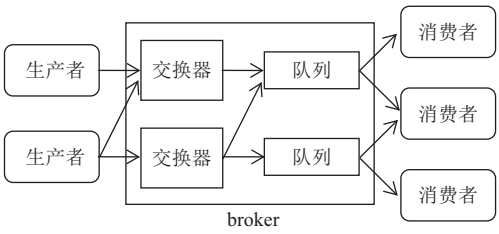

图 1 RabbitMQ 架构图

1.3 小消息嵌入队列索引

RabbitMQ 3.5.0 版本引入小消息嵌入队列索引. 小于 queue_index_embed_msgs_below 参数值的消息属于小消息, 该参数默认值为 4096 bytes. 小消息的持久化操作直接在队列进程中进行, 不使用消息存储, 只需要写一次磁盘, 可以减少 I/O 和内存消耗, 提高 10% 左右的性能[9].

如果小消息被一个交换器路由到多个队列, 这条消息需要被写入多个队列索引; 若使用消息存储, 则只需要写一次. 从磁盘读取消息时, 每个队列索引需要在内存中保持至少 1 个段文件. 段文件包含 16 384 条消息记录. 因此 queue_index_embed_msgs_below 参数的少量增加会导致大量的内存使用[10].

2 RabbitMQ 消息确认过程分析

如图 2 所示, 生产者确认是异步的, 生产者发送消息到 broker, 可以在等待确认的同时发送下一条. 为了在 broker 重启或崩溃时不丢失消息, 消息投递给消费者前需要进行持久化, 消息写入磁盘后向生产者发送确认消息. 消费者收到消息后必须进行确认, 可以发送 basic.ack 命令进行显示确认, 也可以使用自动确认. 若使用自动确认, 消费者接收到消息, 即视其确认了消息. broker 收到消费者发送的确认消息, 将确认记录追加到队列索引的日志文件.

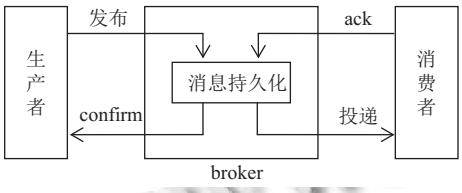

图 2 消息确认过程与消息持久化关系图

开启生产者确认与消费者确认, 持久化小消息在生产者、消费者、RabbitMQ 相关模块间的传递过程如图 3 所示. 1–6: 生产者发送消息到消费者; 7–10: 消费者确认相关过程; 11–14: 生产者确认相关过程.

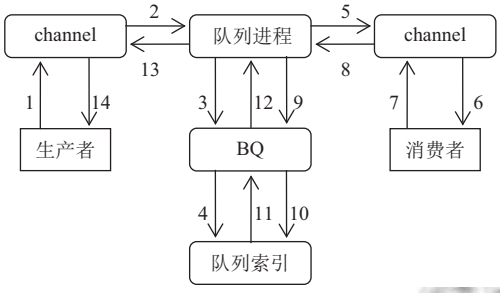

图 3 小消息确认过程图

2.1 生产者确认过程分析

生产者确认过程会依次在 channel、队列进程、队列索引、BQ 处记录生产者确认相关信息. 消息写入磁盘后, 已确认的消息 ID 从队列索引依次传递给 BQ、队列进程、channel, 各处均会将已确认记录删除.

channel 收到生产者发送的消息, 为消息分配一个唯一的序列号, 组装#delivery, 获取需要投递的队列记录列表 Qs, 将#delivery 投递到 Qs 中的队列. channel 使用 dtree 记录消息被投递到哪些队列, 格式为: {消息在 channel 中的序列号, 队列进程 pid 列表, 交换器名称}. 若 channel 将消息投递到 *m* 个队列, channel 收到相应的 *m* 个队列发送的确认消息才会向生产者发送确认消息.

队列进程收到消息, 判断队列的消费者是否满足消息投递条件. 若有消费者满足投递条件且消息队列为空, 则消息不会进入队列, 而是直接投递给消费者端 channel. 需要组装消息状态 (message status). 将包含小消息的发布记录与只包括消息在队列中序列号的投递记录追加到队列索引的日志文件.

若没有消费者满足投递条件或消息队列非空, 则将消息进队. 需要组装消息状态. 将发布记录追加到队列索引的日志文件. 将消息状态加入消息队列. 从消息队列中取消息时, 若消息队列非空且有消费者满足消息投递条件, 则将消息从消息队列中移除. 将投递记录追加到队列索引的日志文件. 将取出的消息投递给消费者端 channel.

队列进程收到生产者端 channel 投递的消息, 使用 gb_trees 记录未确认的消息 ID、发送消息的 channel 和该消息在 channel 中的序列号, 格式为: {消息 ID, {channel pid, 消息在 channel 中的序列号}}.

在发布记录写入日志文件前, 队列索引使用 gb_sets 记录未确认的消息 ID. 队列索引的日志文件可能在两种情况下写入磁盘.

1) 队列进程设置同步定时器, 每 200 毫秒向自身发送 sync_timeout 消息. 队列进程收到消息后, 对队列索引的日志文件执行 sync 操作.

2) 当日志文件中记录数目达到一定数量时, 将内存中预分割的日志文件写入段文件.

消息持久化操作完成后, BQ 使用 gb_sets 记录未确认的消息 ID.

2.2 消费者确认过程分析

消费者确认过程会依次在 BQ、队列进程、channel 处记录消费者确认相关信息. 收到消费者发送的确认消息后, 会按照相反的顺序从未确认记录中删除已确认记录, 最终将包括消息在队列中序列号的确认记录追加到队列索引的日志文件.

消息到达队列进程直接投递给消费者端 channel 时或从消息队列中取消息时, 会在 BQ 相应的 gb_trees 中添加未确认记录, 格式为: {消息在队列中的序列号, 消息状态}.

队列进程将消息投递给 channel, 在 Erlang queue 中添加未确认记录, 格式为: {消息在队列中的序列号, 消费者标签}.

channel 使用 writer 将消息投递给消费者, 在 Erlang queue 中添加未确认记录, 格式为: {投递标签, 消费者标签, {队列进程 pid, 消息在队列中的序列号}}. 收到消费者发送的确认消息后, channel 根据确认消息中的投递标签与 multiple 字段从未确认记录中获取已确认记录, 将已确认的消息序列号发送给相应的队列进程.

2.3 对性能的影响

生产者确认与消费者确认过程涉及较多 dtree、gb_trees、gb_sets 和 Erlang queue 操作, 包括插入、删除、查找、集合运算等. 队列索引的日志文件会定时地或在记录达到一定数量时写入磁盘, 对性能影响较大.

使用 RabbitMQ 2.8.1 进行简单测试, CPU 为双 Xeon E5530, RAM 为 40GB, Erlang R15B, 开启 HiPE, 1 个生产者, 1 个消费者[11]. 不使用生产者确认与消费者确认, 不进行消息持久化, 消息发送速率为: 44824 msg/s; 开启消费者确认后: 32005 msg/s; 接着开启生产者确认: 26103 msg/s; 在此基础上对消息进行持久化: 4725 msg/s[12]. 可见消息确认机制对消息发送速率有一定影响, 消息持久化机制对消息发送速率有较大影响.

3 优化方法

3.1 小消息确认机制的优化

如图 4 所示, 优化后持久化与非持久化小消息的确认过程是相同的. 需要将生产者确认过程与消费者确认过程衔接起来. 生产者发送消息到 broker, 消息投递给消费者前, 不进行消息持久化操作. 不会向日志文件追加记录, 不写段文件; 不会设置同步定时器, 不执行代价较大的 sync 操作. 队列索引与 BQ 不记录未确认的消息 ID. 消费者收到消息后, 向 broker 发送确认消息. broker 收到消费者确认消息, 向生产者发送确认消息. 若消费者没有收到消息, 生产者不会收到确认消息, 此时由生产者重发该消息. 该方法保证了在生产者收到确认消息时消费者已成功接收到消息.

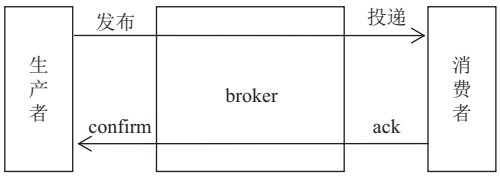

图 4 优化后小消息确认过程图

1) 小消息到达队列进程, 队列进程需要记录生产者确认相关信息. 需要修改队列进程模块的 send_or_record_confirms/2 函数. SenderPid 是发送消息给队列进程的 channel pid, MsgSeqNo 是消息在 channel 中的序列号, MTC 用于记录未确认消息 ID 对应的 {SenderPid, MsgSeqNo}. 添加未确认记录后, 更新队列进程状态.

```
send_or_record_confirms(#delivery{confirm = true,
sender = SenderPid, msg_seq_no = MsgSeqNo,
message = #basic_message{id = MsgId}},
State = #q{msg_id_to_channel = MTC}) ->
  MTC1 = gb_trees:insert(MsgId, {SenderPid,
  MsgSeqNo}, MTC),
  {eventually, State#q{msg_id_to_channel =
  MTC1}};
```

2) 队列进程在消费者确认过程结束后, 向生产者端 channel 发送确认消息. 需要修改队列进程模块的 ack/3 函数. MsgIds 是已确认的消息 ID 列表. 需要获取 MsgIds 对应的 channel pid 和消息在 channel 中的序列号, 将包含消息在 channel 中序列号的确认消息发送给相应的 channel. 更新队列进程状态.

```
fun (State1 = #q{backing_queue = BQ,
backing_queue_state = BQS,
msg_id_to_channel = MTC}) ->
  {MsgIds, BQS1} = BQ:ack(AckTags, BQS),
  MTC1 = confirm_messages(MsgIds, MTC),
  State1#q{backing_queue_state = BQS1,
  msg_id_to_channel = MTC1}
end
```

3.2 继续优化消费者确认过程

可以在上述优化的基础上减少消费者确认过程中的插入、删除等操作, 提高性能, 减少内存使用. BQ 和队列进程不记录消费者确认相关消息, 消费者端 channel 记录: {消息投递标签, 消费者标签, {队列进程 pid, 消息 ID}}.

1) 队列进程向 channel 投递消息, 格式为:{deliver, ConsumerTag, AckRequired, Msg}. ConsumerTag 是消费者标签, AckRequired 取值为 true 或 false. Msg 类型为 rabbit_amqqueue:qmsg(), 格式为: {队列名称, 队列进程 pid, 消息在队列中的序列号, Redelivered, Message}. Redelivered 取值为 true 或 false, Message 类型为 #basic_message. 使用模式匹配从 Message 中提取消息 ID. 需要修改 channel 模块的 record_sent/4 函数.

#basic_message{id = MsgId} = Message

2) 消费者端 channel 向队列进程发送的消息中包括已确认的消息 ID 列表 MsgIds. 队列进程收到确认消息, 向生产者端 channel 发送相应的确认消息. 更新队列进程状态. 需要修改队列进程模块的 handle_cast/2 函数.

```
handle_cast({ack, MsgIds, _ChPid},
State = #q{msg_id_to_channel = MTC}) ->
    MTC1 = confirm_messages(MsgIds, MTC),
    noreply(State#q{msg_id_to_channel = MTC1});
```

4 性能测试

4.1 测试环境与方法

生产者、消费者、RabbitMQ 在同一台机器上. 开启生产者确认与消费者确认. 持久化小消息, 消息大小为 1500 bytes. 性能测试工具为 PerfTest. 测试环境配置如表 1 所示.

表 1 性能测试环境配置表

配置项	参数
CPU	4核 2.66 GHz
内存	4 G
硬盘	280 G
操作系统	Windows 7 32位
Erlang版本	19.3
RabbitMQ版本	3.6.6
PerfTest版本	1.2.0

在 1 个虚拟主机中启动不同数量的持久化队列, 每个队列有 2 个生产者、3 个消费者, 每个生产者连接中有 2 个 channel, 每个消费者连接中有 3 个 channel. 队列数量小于等于 15 时, 绑定到同一个持久化 direct 交换器; 队列数量大于 15 时, 绑定到两个持久化 direct 交换器. 开启 management 插件与 top 插件. 分别记录优化前与优化后的消息发送速率, 每种情况测试 10 分钟, 测试 3 次取平均值.

消息发送速率提高百分比的计算方法为: (优化后消息发送速率–优化前消息发送速率)/优化前消息发送速率*100%.

消息发送速率平均提高百分比为: 不同队列数量时, 消息发送速率提高百分比的算术平均值.

4.2 测试结果与分析

如图 5 所示, 在 1 个虚拟主机中, 随着队列数量增加, 消息发送速率先增加然后缓慢下降. 优化后消息发送速率提高的比例是逐渐下降的. 1 个队列时, 优化后的消息发送速率是优化前的 3.08 倍; 2 个队列时, 优化后的消息发送速率是优化前的 2.01 倍; 15 个队列时, 消息发送速率提高 40.9%; 30 个队列时, 消息发送速率提高 40.3%. 队列数量大于 3 时, 消息发送速率平均提高 42.9%.

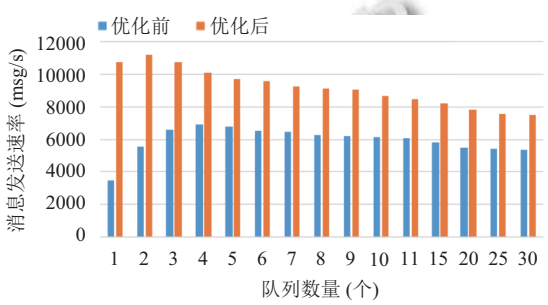

图 5 优化后消息发送速率对比图

如图 6 所示, 对消费者确认过程进一步优化后, 在同样的测试环境中, 随着队列数量增加, 优化后的消息发送速率逐渐下降. 1 个队列时, 优化后的消息发送速率取得最大值, 是优化前的 3.48 倍; 2 个队列时, 优化后的消息发送速率是优化前的 2.16 倍; 15 个队列时, 消息发送速率提高 52.6%; 30 个队列时, 消息发送速率提高 53.4%. 队列数量大于 3 时, 消息发送速率平均提高 56.5%.

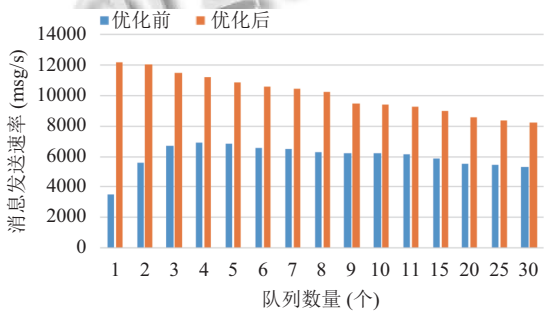

图 6 继续优化后消息发送速率对比图

优化后不执行消息持久化操作, 减少部分内存操作, 使消息发送速率得到提高. 继续优化后, 不会在 BQ 和队列进程处记录消费者确认相关信息, 减少了插入、查找、删除等操作, 进一步提高了消息发送速率, 但是在队列数量较多时可靠性略有下降. 上述两种优化方法需要确保每个队列至少有一个消费者, 适用于

消费速度很快的情形. 在生产者、消费者、队列数量较少时可以获得更大的性能提升. 生产者重发消息的策略, 可以根据实际应用场景确定. 与改进前类似, 在异常情况下, 消费者可能收到重复的消息. 第一种优化方法保留了完整的消费者确认过程, 能够较好地处理 basic.reject 与 basic.nack 等命令, 消费者可以拒绝接收某些消息. 第二种优化方法简化了消费者确认过程, 无法处理消费者拒绝消息的情况, 适用于消费者只对消息进行确认的情况. 可以根据应用程序对性能、可靠性的不同需求使用相应的优化方法.

5 结语

本文详细分析了 RabbitMQ 中持久化小消息的确认过程, 将生产者确认过程与消费者确认过程结合起来进行优化, 使生产者可以获知消费者成功接收到消息, 提高了持久化小消息的发送速率. 要使消息发送速率得到根本提高, 可以重新设计 RabbitMQ 的架构: 使用多个轻量级进程实现逻辑队列与逻辑 channel, 或集群部署使用.

参考文献

1 Rostanski M, Grochla K, Seman A. Evaluation of highly available and fault-tolerant middleware clustered architectures using RabbitMQ. Federated Conference on Computer Science and Information Systems. Warsaw, Poland. 2014. 879–884.

2 Videla A, Williams JJW. RabbitMQ 实战: 高效部署分布式消息队列. 汪佳南, 译. 北京: 电子工业出版社, 2015.

3 Ionescu VM. The analysis of the performance of RabbitMQ and ActiveMQ. 14th RoEduNet International Conference - Networking in Education and Research (RoEduNet NER). Craiova, Romania. 2015. 132–137.

4 Vandikas K, Tsiatsis V. Performance evaluation of an IoT platform. 8th International Conference on Next Generation Mobile Apps, Services and Technologies. Oxford, UK. 2014. 141–146.

5 Dawar S, Fallon E, Bennet T, *et al*. An extensible architecture for mobile network management event distribution and rule processing - a performance evaluation. 1st International Conference on Artificial Intelligence, Modelling and Simulation. Kota Kinabalu, Malaysia. 2013. 451–456.

6 Yang WJ, Liu XG, Zhang L, *et al*. Big data real-time processing based on storm. 12th IEEE International Conference on Trust, Security and Privacy in Computing and Communications. Melbourne, VIC, Australia. 2013. 1784–1787.

7 Cesarini F, Thompson S. Erlang 编程指南. 慕尼黑 Isar 工作组, 杨剑, 译. 北京: 机械工业出版社, 2011.

8 袁佳. 基于主机日志的入侵检测系统的设计与实现[硕士学位论文]. 北京: 北京邮电大学, 2014.

9 李帅. RabbitMQ 进程结构分析与性能调优. https://www.qcloud.com/community/article/164816001481011847. [2016-10-10].

10 Persistence Configuration. http://www.rabbitmq.com/persistence-conf.html. [2017-06-01]

11 RabbitMQ performance measurements, part 1. http://www.rabbitmq.com/blog/2012/04/17/rabbitmq-performance-measurements-part-1/. [2012-04-17]

12 RabbitMQ performance measurements, part 2. http://www.rabbitmq.com/blog/2012/04/25/rabbitmq-performance-measurements-part-2/. [2012-04-25]