RabbitMQ初步优化分析报告

## 调研

* 1. rabbitmq内部消息处理路径分析（进程）

消息的处理会流经以下几个进程。

Reader

Channel

amqqueue

Channel

Writer

Msg\_store

1

2

5

3

4

**Reader**：负责接收tcp数据包，按照amqp协议进行帧解析。

**Channel**：负责处理大部分amqp协议方法以及实现路由功能。

**Amqqueue**：负责记录处理index，pending\_ack，pending\_confirm,对消息进行非持久化存储，根据当前磁盘，内存使用情况决定何时将消息回写磁盘。

**Msg\_store**:负载实现消息的持久化存储。

**Writer**: 安装amqp协议封装数据包，并通过tcp连接发送出去。

**消息的处理的主干流程：**

1. 由reader进程通过gen\_tcp接收tcp数据包，按照amqp协议解析，解析后根据协议头中channel\_id将消息转发给对应channel进程。
2. Channel进程，根据消息体中的routing\_key,exchange以及本地的路由表bindings进行路由，决定该消息应该被放入哪些queue,最后将该消息转发给对应的amqqueue 进程。
3. Amqqueue进程检查消息体中是否设置有持久化标志，若有就该消息交给msg\_store进行持久化（异步），持久化成功后msg\_store会回调通知amqqueue，amqqueue给producer发ack。另外该进程还需要负责记录消息的index，该index是queue中消息seq\_id与msg\_id的对应关系，一个消息有唯一的msg\_id，只存储一份，但它可以存在于多个queue中。
4. Amqqueue进程选择一个订阅了本队列的consumer，将消息发送给该consumer对应的channel进程。
5. Channel进程收到消息后, 记录pending ack，并消息再交给writer，由writer封包发送给consumer。
   1. qps瓶颈定位

1.2.1 定位进程

我们通过trace rabbitmq的流控机制（具体定位过程参见<http://goo.gl/yOSuy>），定位到瓶颈位于amqueue进程。

1.2.2瓶颈进程各部分功能耗时分析

通过对该进程对应的代码分析以及trace，发现该进程的处理时间主要花在以下方面：

Store(57%)

Confirm(30%)

 Ack(5%)

 Other

其中Store又主要分为两部分：一部分是index的处理占store的50%，一部分是msg的处理占22%。

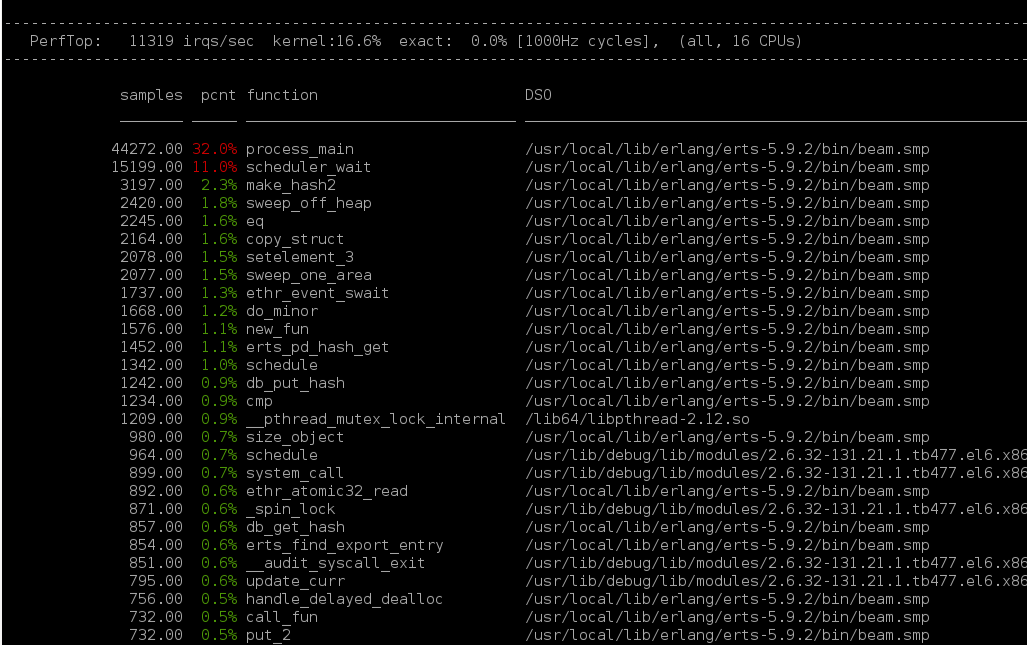
* 1. cpu的使用情况

通过工具erlang：etop发现最耗cpu的进程依次是Reader,Amqque,Channel和Msg\_Store。

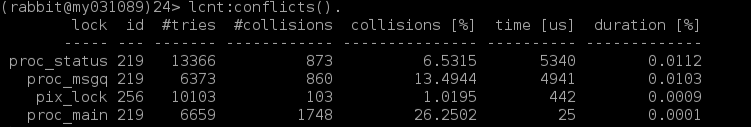
* 1. 网络与磁盘使用情况

通过测试，在最大压力下网络与磁盘都有很大余量。

* 1. perf top



* 1. 锁使用情况



## 优化实验

#### 实验环境

硬件：

三台开发测试机(一生产者，一消费者，一MQ服务器): Intel(R) Xeon(R) CPU E5620 @ 2.40GHz （16核），24G RAM， 普通磁盘。

软件：

Erlang虚拟机: R16B

RabbitMQ:2.8.2

网络：

Ping：0.220ms

Netperf: 941.43 (10^6bits/sec)

#### 实验内容

1. Hipe（JIT）

为测试打开hipe是否能够提升性能，我们做了以下测试：

1. 1 vhost，1 queue，1 channel durable+confirm+ack 包大小1kb

Nohipe: qps 6000 cpu:270% hipe: qps 8000 cpu 260%

1. 2 vhost , 5 queues , 2 channel durable+confirm+ack 包大小1kb

Nohipe:qps 23000 cpu:1120% hipe:qps 23000 cpu %1130

从这两个测试可以看出打开hipe，能在单个queue的情况下提升qps，降低单位qps的cpu消耗量。但在多并发多个queue的情况下对性能没有影响。

1. 进程堆大小调整

增大关键进程堆大小，期望能减少gc对性能的影响，测试结果性能没能提高。

1. gen\_tcp delay\_send

启用发送缓冲区，期望减少阻塞次数，在rabbit\_writer的main\_loop中设置缓冲区大小：

rabbit\_net:setopts(State#wstate.sock, [{delay\_send, true}, {sndbuf, 327680}, {recbuf, 327680}])，

测试在2 vhost , 5 queues , 2 channel durable+confirm+ack 包大小1kb的环境下，qps有小幅提高从23000涨到了24500。

1. scheduler binding

设置erlang虚拟机调度器的cpu绑定，期望降低调度带来的开销，测试结果表明没有效果。

1. 设置进程优先级

提高amqque进程的优先级为high，测试结果显示没有效果。

1. 不启用confirm

通过前面的分析，可以看出在瓶颈所在的进程amqque进程中，处理confirm消息占用了大量cpu资源。confrim的作用在于当消息真正落地写到磁盘时，给生产者发送ack确认，若生产者在收到该ack后才丢弃该消息，就可以保证消息一定不丢，这是一种非常高强度的可靠性保证。但若没有这么高的要求则可以不启用confirm机制，rabbitmq还有heartbeat的机制，在rabbitmq 服务器down掉时，生产者可以及时发现做出相应处理，减少可能丢失消息的风险。

测试， 在2 vhost , 5 queues , 2 channel durable +ack 包大小1kb的环境下，不开启confirm相比开启confirm，qps从23000提升到了29000。

1. 调整各类进程数比例

为提高单台rabbitmq服务器的qps，需要提高并发度，增加相关进程数目，根据前面的调研可以看出主要需要增加并发的进程为reader,channel,amqqueue这几个进程数的调整都可以通过客户端连接参数控制。Reader与writer是每个vhost对应一个，channl和queue在同一各vhost里可以有任意多个。但这些进程并非越多越好，目前主要瓶颈在amqqueue上，所以需要增加其数量，但对于reader与channel则不需要太多，过多的反而会消耗更多cpu降低性能，但当queue的数据增加时瓶颈也会转移到这两个进程，这时又需要增加这两个进程的数量，所以从整体上看，为达到最近的性能，这些进程必须按照某个最佳比例搭配，而这个最佳的比例在不同的硬件上是不同的。在我们的测试机上(16core sas 24ram)上我们做了以下几组测试(不开前面所述的优化选项)：

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 测试ID | Reader | Channel | Queue | Cpu | QPS |
| 1 | 1 | 2 | 4 | 560% | 18k |
| 2 | 1 | 2 | 5 | 690% | 20k |
| 3 | 1 | 3 | 5 | 730% | 20K |
| 4 | 1 | 3 | 6 | 790% | 22k |
| 5 | 1 | 3 | 9 | 1050% | 26k |
| 6 | 1 | 4 | 10 | 1100% | 29k |
| 7 | 2 | 4 | 10 | 1170% | 29K |
| 8 | 2 | 4 | 12 | 1150% | 25k |
| 9 | 4 | 4 | 4 | 770% | 18k |

从测试1和8可以看出，同样的性能 测试1更加省cpu，其余测试可以看出提高并发度可以明显提高qps。cpu的占用率最高不超过1200%。

VIII 调整参数queue\_index\_max\_journal\_entries

根据前面的调研，rabbitmq在记录和处理index的耗时较多，增大该值可以减少写index segment文件的次数，经测试增大该值对性能没有影响，降低则可减少qps“毛刺”的发生。

## 总结与建议

**总结：**

经过前面的实验可以看出使用的优化措施有III,IV,VII，组合应用这些优化措施，实验测得单机（16核 24G内存 sas盘）1k包大小，QPS可以达到32000，cpu占用1150%, 在这样的负载下，我们用

stress --cpu 16 --io 4 --vm 2 --vm-bytes 128M --timeout 600s

压内存cpu和磁盘，得益于rabbitmq良好的流控机制，系统表现稳定。

**建议：**

1. 采用更多核的cpu，16或更多。
2. 内存不必太大，普通sas磁盘。
3. 不启用confirm机制
4. 对不同机器进行测试找出最的进程数比例。
5. 开启writer进程的gen\_tcp 缓存