通过流控机制看rabbimq的性能（持久化）瓶颈

我们在对rabbimq进行性能测试时发现在单个队列的情况下，无论怎么压qps都上不去，而此时服务器的cpu、网络、磁盘都没什么压力，但通过web管理界面看到发送通道不断的处于流控状态，显然流控机制被启动是性能不佳的最表象原因，那这背后到底是什么原因使得rabbimq启动流控机制呢？

首先总结下前篇博客讲述的rabbitmq流控机制原理---实质上就是通过监控没各进程的mailbox，当某个进程负载过高来不及接收消息时，这个进程的mailbox就会开始堆积消息，当堆积到一定量时，就会阻塞住上游进程让其不得接收新消息，从而慢慢上游进程的mailbox也会开始积压消息，到了一定的量也会阻塞上游的上游的进程接收消息，最后就会使得负责网络数据包接收的进程阻塞掉，暂停接收数据。这就有点像一个多级的水库，当下游水库压力过大时，上游水库就得关闭闸门，使得自己的压力也越来越大这就需要关闭更上游的水库闸门直到关闭最最上游的闸门。

从这套流控机制可以看出，对于处于整个流控链中的任意进程，只要该进程被阻塞，上游进程必定全部被阻塞，若某个进程是性能瓶颈，必然会导致起上游所有进程被阻塞。所以我们可以利用流控机制的这个特点找出瓶颈所在的进程，即通过trace找出流控链中被阻塞进程中最下游的那个进程，这个进程的下游进程就是瓶颈所在。例如有四个进程A、B、C、D，消息在进程间传递的顺序是A->B->C->D， 若此时观察到 A，B不断的被阻塞，则瓶颈一定在C ，因为若A是瓶颈就不会有很多的消息往后面传，若B是瓶颈，则A一定被阻塞，下游的消息就会变少，B就不会成为瓶颈，若D是瓶颈则 C也会被阻塞。

1.首先找出消息传递链条的所有进程以及顺序关系，rabbimq的流控机制实现在模块credit\_flow中，当处于流控机制的进程往下游发前会调用credit\_flow:send/1在接收消息时会调用crediat\_flow:ack/1参数都为pid，在send中该pid为下游进程pid，ack为上游进程pid，为此我们可以trace这两个函数找出所有处于流控机制中的进程以及顺序关系，在通过i(Pid)提供的进程信息结合代码分析各进程扮演的角色。

|  |
| --- |
| {ok, Log} = file:open("flow\_trace", [write, append]),  Tfun = fun(Msg, \_) ->  io:format(Log,"~p ~n", [Msg])  end,  dbg:tracer(process, {Tfun, null}),  dbg:tp({credit\_flow, send, '\_'}, []),  dbg:tp({credit\_flow, ack, '\_'}, []),  dbg:p(all, c). |

最后分析出这些进程的关系如下：



rabbit\_reader:负责接收网络数据包，解析数据。

rabbit\_channel:负责处理amqp协议的各种方法，进行路由解析等。

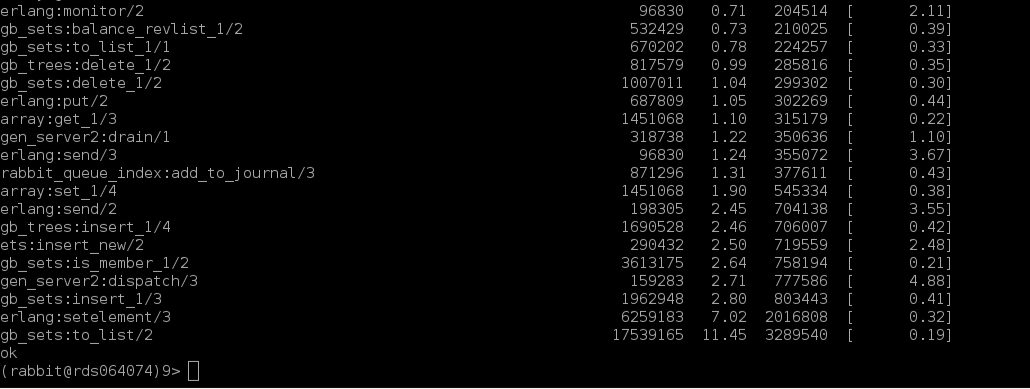
rabbit\_amqqueue\_process:负责实现queue的所有逻辑。

rabbit\_msg\_store:负责实现消息的持久化。

2.有了以上信息后，我们就可以开始给rabbitmq加上load，查看哪些进程被阻塞。由于阻塞这个动作是在credit\_flow:send/1这个函数内完成的，并不是每次调用都为阻塞绝大多数时候不会，而且阻塞的状态信息保存在进程字典中，所有直接trace阻塞这个动作不太方便。但由于阻塞之后必然有反阻塞，而这个动作是调用credit\_flow:unblock/1完成的，所以我们可以trace这个函数来查看哪些进程被阻塞。

|  |
| --- |
| {ok, Log} = file:open("flow\_trace", [write, append]),  Tfun = fun(Msg, \_) ->  io:format(Log,"~p ~n", [Msg])  end,  dbg:tracer(process, {Tfun, null}),  dbg:tpl({credit\_flow, unblock, '\_'}, []),  dbg:p(all, c). |

通过trace结果发现，瓶颈在rabbit\_amqueue\_process这个进程。为此eprof看下该进程有没有热点可以优化,profile的结果如下（部分截图）



从上可以看出并没有热点。

对于rabbit\_amqqueue\_process进程是每个队列都会对应一个这样的进程，那多申明几个队列，多几个这样的进程性能是否能提升呢？通过实验发现答案是肯定的，增加队列能够显著提升qps，但并不是越多越好，太多反而性能会下降，在增加队列数量时发现慢慢瓶颈会转移到rabbit\_channel进程上，当然channel也可以声明多个解决该问题，但无论怎么设置多少channel多少queue，瓶颈始终都不会出现在msg\_store\_persistent进程。