RabbitMQ 的Confirm机制优化

### 背景：

RabbitMQ的confirm是于生产者与MQ之间的确认机制，确保消息确实投递到了MQ。在许多对可靠性要求比较高的应用场景下都需要使用该机制确保消息不丢。测试显示，该功能开启与否对性能影响很大，在单台服务器上开8个queue，消息持久化，1kb大小的消息，不开confirm 时QPS 28k，开启confirm后QPS降到21k。

### 源码分析：

开启confirm后，生产者与RabbitMQ之间通过发送确认序号来对消息进行确认，该序号是per channel的。对消息进行确认就是简单的将该消息对应的序号发回给生产者，但RabbitMQ收到消息后并不是立即回ack，在不同配置下，回ack的时机是不同的。Confirm的过程伴随着消息在MQ中整个处理流程，为此接下来我们从整个消息的生命周期来分析confirm的处理机制。以下是confirm机制的主干流程。

1. Rabbit\_channel 接收到新消息后

根据路由规则确定该消息应该被投递给哪些queue, 为每个消息生成一个全局唯一标识msgid,当然每个消息也有per channel的确认序号，后面还会有别的序号，为了避免冲突我们将该确认序号命名为ch\_seq\_no。在将该消息以及对应的msgid，ch\_seq\_no投递给相应queue进程后，记录下unconfirmed消息。

|  |
| --- |
| dtree:insert(MsgSeqNo, QPids, XName,State#ch.unconfirmed)}. |

1. Rabbit\_amqqueue\_process收到信息以后
2. 记录为未确认的消息

|  |
| --- |
| maybe\_record\_confirm\_message({eventually, SenderPid, MsgSeqNo, MsgId},  State = #q{msg\_id\_to\_channel = MTC}) ->  State#q{msg\_id\_to\_channel =  gb\_trees:insert(MsgId, {SenderPid, MsgSeqNo}, MTC)}; |

其中MsgId是消息的全局唯一表示，SenderPid是接收该消息的channel进程，MsgSeqNo是该消息的确认序号。

（2）交给backing\_queue做持久化

3. backing\_queue收到新消息后

（1）记录为未确认的消息

|  |
| --- |
| UC1 = gb\_sets\_maybe\_insert(NeedsConfirming, MsgId, UC), |

（2）将消息投递给rabbit\_msg\_store

4. rabbit\_msg\_store收到新消息后

（1）记录为未确认的消息

|  |
| --- |
| record\_pending\_confirm(CRef, MsgId, State) ->  update\_pending\_confirms(  fun (\_MsgOnDiskFun, CTM) ->  dict:update(CRef, fun (MsgIds) -> gb\_sets:add(MsgId, MsgIds) end,  gb\_sets:singleton(MsgId), CTM)  end, CRef, State). |

（2）定期或者在切换消息存储文件时，对消息进行确认

|  |
| --- |
| internal\_sync(State = #msstate { current\_file\_handle = CurHdl,  cref\_to\_msg\_ids = CTM }) ->  State1 = stop\_sync\_timer(State),  CGs = dict:fold(fun (CRef, MsgIds, NS) ->  case gb\_sets:is\_empty(MsgIds) of  true -> NS;  false -> [{CRef, MsgIds} | NS]  end  end, [], CTM),  ok = case CGs of  [] -> ok;  \_ -> file\_handle\_cache:sync(CurHdl)  end,  lists:foldl(fun ({CRef, MsgIds}, StateN) ->  client\_confirm(CRef, MsgIds, written, StateN)  end, State1, CGs).  client\_confirm(CRef, MsgIds, ActionTaken, State) ->  update\_pending\_confirms(  fun (MsgOnDiskFun, CTM) ->  case dict:find(CRef, CTM) of  {ok, Gs} -> MsgOnDiskFun(gb\_sets:intersection(Gs, MsgIds),  ActionTaken),  MsgIds1 = rabbit\_misc:gb\_sets\_difference(  Gs, MsgIds),  case gb\_sets:is\_empty(MsgIds1) of  true -> dict:erase(CRef, CTM);  false -> dict:store(CRef, MsgIds1, CTM)  end;  error -> CTM  end  end, CRef, State)  update\_pending\_confirms(Fun, CRef,  State = #msstate { clients = Clients,  cref\_to\_msg\_ids = CTM }) ->  case dict:fetch(CRef, Clients) of  {\_CPid, undefined, \_CloseFDsFun} -> State;  {\_CPid, MsgOnDiskFun, \_CloseFDsFun} -> CTM1 = Fun(MsgOnDiskFun, CTM),  State #msstate {  cref\_to\_msg\_ids = CTM1 }  end. |

这里是将所有需要被confirm的消息按所属的queue进行分组，然后分别调用对应queue发送callback函数MsgOnDiskFun以及需要confirm的MsgIds。

5. 当rabbit\_amqqueue\_process收到确认消息后

（1）执行传过来的callback函数MsgOnDiskFun，该Callback函数最终调用的是backing\_queue中的

|  |
| --- |
| fun (?MODULE, State = #vqstate { msgs\_on\_disk = MOD,  msg\_indices\_on\_disk = MIOD,  unconfirmed = UC }) ->  Confirmed = gb\_sets:intersection(UC, MsgIdSet),  record\_confirms(gb\_sets:intersection(MsgIdSet, MIOD),  State #vqstate {  msgs\_on\_disk =  gb\_sets:union(MOD, Confirmed) })  end). |

（2）在每次reply和norely将需要确认的消息发送给对应的Channel进程

|  |
| --- |
| reply(Reply, NewState) ->  assert\_invariant(NewState),  {NewState1, Timeout} = next\_state(NewState),  {reply, Reply, NewState1, Timeout}.  noreply(NewState) ->  assert\_invariant(NewState),  {NewState1, Timeout} = next\_state(NewState),  {noreply, NewState1, Timeout}.  next\_state(State = #q{backing\_queue = BQ, backing\_queue\_state = BQS}) ->  {MsgIds, BQS1} = BQ:drain\_confirmed(BQS),  State1 = ensure\_stats\_timer(  ensure\_rate\_timer(  confirm\_messages(MsgIds, State#q{  backing\_queue\_state = BQS1}))),  case BQ:needs\_timeout(BQS1) of  false -> {stop\_sync\_timer(State1), hibernate };  idle -> {stop\_sync\_timer(State1), ?SYNC\_INTERVAL};  timed -> {ensure\_sync\_timer(State1), 0 }  end.  confirm\_messages(MsgIds, State = #q{msg\_id\_to\_channel = MTC}) ->  {CMs, MTC1} =  lists:foldl(  fun(MsgId, {CMs, MTC0}) ->  case gb\_trees:lookup(MsgId, MTC0) of  {value, {SenderPid, MsgSeqNo}} ->  {rabbit\_misc:gb\_trees\_cons(SenderPid,  MsgSeqNo, CMs),  gb\_trees:delete(MsgId, MTC0)};  none ->  {CMs, MTC0}  end  end, {gb\_trees:empty(), MTC}, MsgIds),  rabbit\_misc:gb\_trees\_foreach(fun rabbit\_misc:confirm\_to\_sender/2, CMs),  State#q{msg\_id\_to\_channel = MTC1}. |

6. 当channel收到confirm消息后

(1)记录需要confirm的消息

|  |
| --- |
| record\_confirm(undefined, \_, State) ->  State;  record\_confirm(MsgSeqNo, XName, State) ->  record\_confirms([{MsgSeqNo, XName}], State).  record\_confirms([], State) ->  State;  record\_confirms(MXs, State = #ch{confirmed = C}) ->  State#ch{confirmed = [MXs | C]}.  confirm([], \_QPid, State) ->  State;  confirm(MsgSeqNos, QPid, State = #ch{unconfirmed = UC}) ->  {MXs, UC1} = dtree:take(MsgSeqNos, QPid, UC),  record\_confirms(MXs, State#ch{unconfirmed = UC1}). |

1. 合并可以合并的confirm消息，并发送给发送者

|  |
| --- |
| send\_confirms(State = #ch{tx\_status = none, confirmed = C}) ->  MsgSeqNos =  lists:foldl(fun ({MsgSeqNo, XName}, MSNs) ->  maybe\_incr\_stats([{XName, 1}], confirm, State),  [MsgSeqNo | MSNs]  end, [], lists:append(C)),  send\_confirms(MsgSeqNos, State#ch{confirmed = []});  send\_confirms(State) ->  maybe\_complete\_tx(State).  send\_confirms([], State) ->  State;  send\_confirms([MsgSeqNo], State = #ch{writer\_pid = WriterPid}) ->  ok = rabbit\_writer:send\_command(WriterPid,  #'basic.ack'{delivery\_tag = MsgSeqNo}),  State;  send\_confirms(Cs, State) ->  coalesce\_and\_send(Cs, fun(MsgSeqNo, Multiple) ->  #'basic.ack'{delivery\_tag = MsgSeqNo,  multiple = Multiple}  end, State).  coalesce\_and\_send(MsgSeqNos, MkMsgFun,  State = #ch{writer\_pid = WriterPid, unconfirmed = UC}) ->  SMsgSeqNos = lists:usort(MsgSeqNos),  CutOff = case dtree:is\_empty(UC) of  true -> lists:last(SMsgSeqNos) + 1;  false -> {SeqNo, \_XName} = dtree:smallest(UC), SeqNo  end,  {Ms, Ss} = lists:splitwith(fun(X) -> X < CutOff end, SMsgSeqNos),  case Ms of  [] -> ok;  \_ -> ok = rabbit\_writer:send\_command(  WriterPid, MkMsgFun(lists:last(Ms), true))  end,  [ok = rabbit\_writer:send\_command(  WriterPid, MkMsgFun(SeqNo, false)) || SeqNo <- Ss],  State. |

总结：

以上过程可以总结归纳为下图所示的流程，



### 优化：

从上面的分析可以看出，整个confirm的处理流程在不同的模块间多次传递确认序号，msgid，ch\_seq\_no等信息，期间有大量的gb\_sets，gb\_trees，dtree，dict的操作，包括插入、删除、求交集差集，遍历等。考虑到一个消息只可能从一个channel被接收，所有上述流程可以简化，只建立channel与msg\_store之间的联系，缩短处理路径。

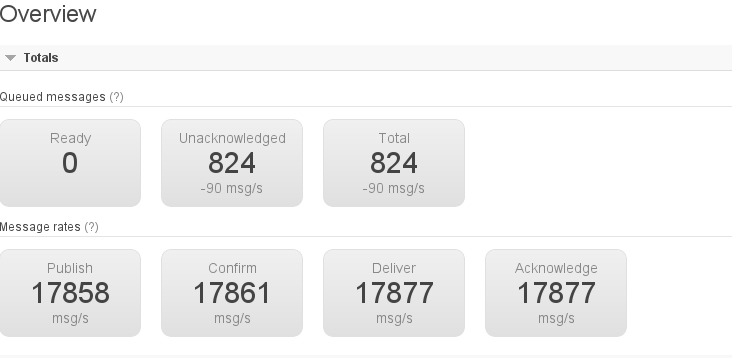


### 测试结果

RabbitMQ版本 2.8.4 erlang：R16B

测试参数：ack = true ，confirm=true，vhost=2，queue=4, channel=2

优化前



优化后

