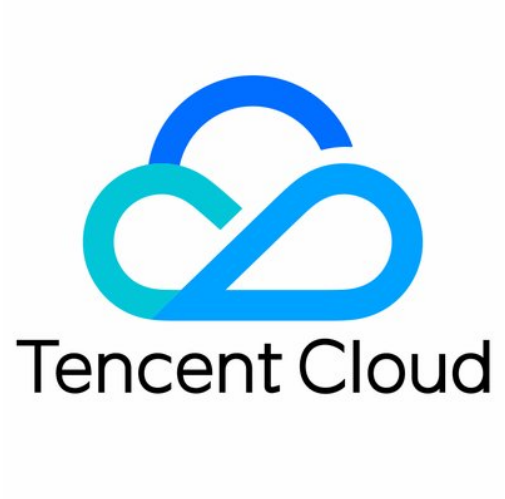
TDMQ for CMQ

Best Practices

Product Documentation



©2013-2019 Tencent Cloud. All rights reserved.

Copyright Notice

©2013-2019 Tencent Cloud. All rights reserved.

Copyright in this document is exclusively owned by Tencent Cloud. You must not reproduce, modify, copy or distribute in any way, in whole or in part, the contents of this document without Tencent Cloud's the prior written consent.

Trademark Notice

Tencent Cloud

All trademarks associated with Tencent Cloud and its services are owned by Tencent Cloud Computing (Beijing) Company Limited and its affiliated companies. Trademarks of third parties referred to in this document are owned by their respective proprietors.

Service Statement

This document is intended to provide users with general information about Tencent Cloud's products and services only and does not form part of Tencent Cloud's terms and conditions. Tencent Cloud's products or services are subject to change. Specific products and services and the standards applicable to them are exclusively provided for in Tencent Cloud's applicable terms and conditions.

Contents

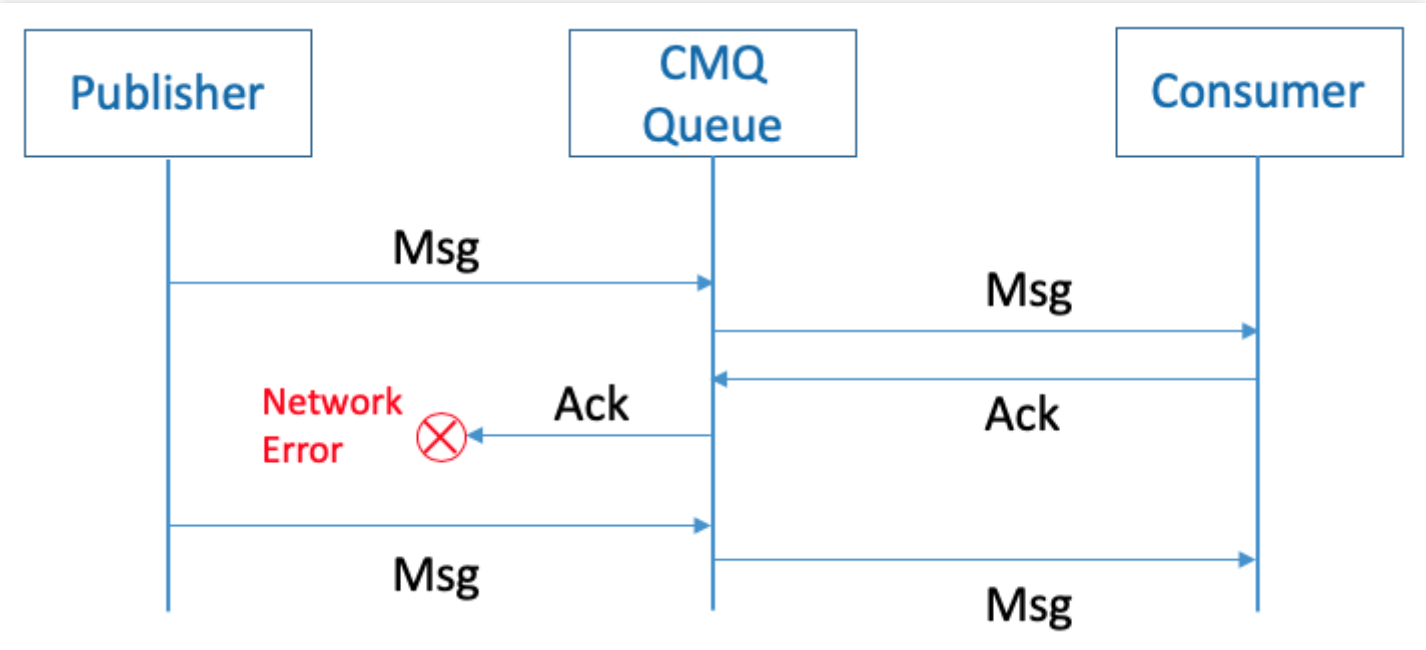
©2013-2019 Tencent Cloud. All rights reserved.

Best Practices
Message Deduplication

Last updated : 2022-02-11 11:39:33

For duplicate messages, the best solution is to make them reentrant (repeated message consumption has no impact on the business). If this is not possible, deduplication needs to be performed on the consumer side.

Causes of Message Duplication



Messages may be lost due to network exceptions, server failures, or other issues. To prevent message loss and ensure reliable delivery, TDMQ for CMQ adopts a mechanism of dual acknowledgment mechanism for message production and consumption.

Message production acknowledgment: normally, the producer sends a message to TDMQ for CMQ and waits for acknowledgment from it, and it stores the message in the disk and returns a success to the producer. However, if the producer request times out or TDMQ for CMQ returns an error, the producer needs to send the message to it again.

Consumer acknowledgment: TDMQ for CMQ delivers the message to the consumer and sets it to invisible, and the consumer uses a handle to delete the message during the invisibility period. If the message is not deleted and the invisibility period elapses, it will become visible again.

©2013-2019 Tencent Cloud. All rights reserved.

As the message acknowledgment mechanism of TDMQ for CMQ guarantees at-least-once delivery of every message, repeated production/consumption may occur due to network jitters or producer/consumer exceptions.

Deduplication Scheme

The first step of deduplication is to identify duplicate messages. A common approach is to insert a deduplication key in the message body when producing a message, which is used to identify duplicate messages at the time of consumption. The deduplication key can be a unique value composed of <producer ip="" +="" thread="" id="" timestamp="" incremental="" value="" within="" a="" time="" period="">.

If there is only one consumer, you can cache the consumed deduplication keys (e.g., KV) and check whether they have already been consumed during each consumption. The key cache can be set to expire based on the maximum validity period of messages. TDMQ for CMQ provides a parameter for the current minimum unconsumed message time (min_msg_time) in the queue. You can use this parameter and the maximum retry period for message production to determine the cache expiration time.
If there are multiple consumers, distributed caching of deduplication keys should be used.

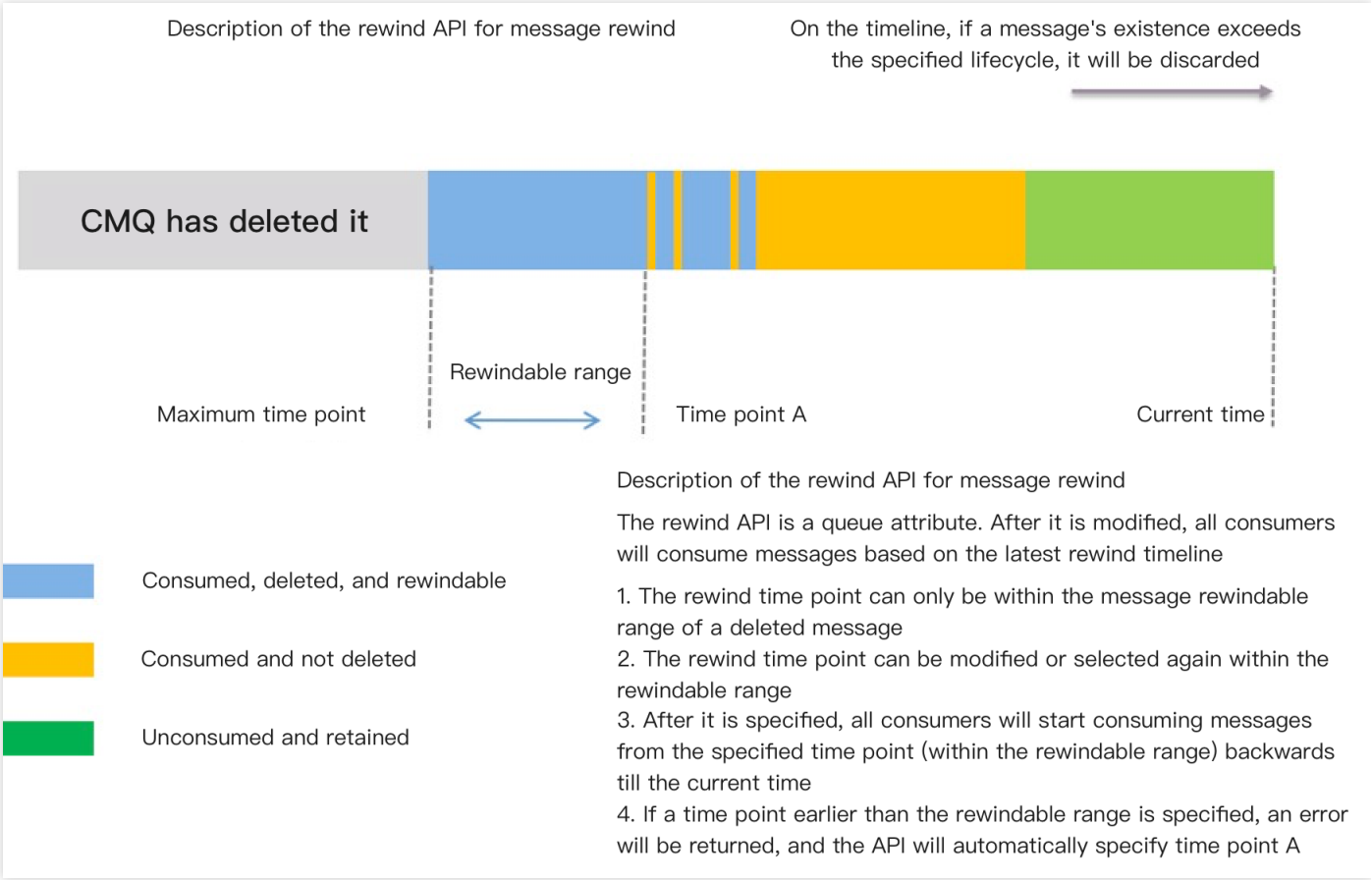
- Calculate key expiration time based on the maximum message validity period:
  current_time + max_retention_time + max_retry_time + max_network_time
  (Current time) + (maximum validity period) + (maximum retry period) + (maximum network time)

- Calculate key expiration time based on the minimum unconsumed time provided by TDMQ for CMQ:
  min_msg_time + max_retry_time + max_network_time
  (Minimum unconsumed time) + (maximum retry period) + (maximum network time)

  TDMQ for CMQ supports a maximum message validity period of 15 days. You can adjust the validity period within the value range as needed.
  The earliest time of a consumed and acknowledged message in the current TDMQ for CMQ queue, which is the earliest time point as shown in the figure below. All the messages before this time

©2013-2019 Tencent Cloud. All rights reserved.

point have already been deleted, while those after it may have not.



Examples

How to avoid duplicate commits:

- Scenario: A is a producer, B is a consumer, and TDMQ for CMQ is between them. A transferred 10 USD to B and sent a message to TDMQ for CMQ. TDMQ for CMQ successfully received the message but failed to respond to A due to network jitters or A's server failures. A assumed that the sending failed and therefore produced a new message, resulting in duplicate commits.

- Solution: when producing the message, A can add information such as timestamp to generate a unique deduplication key. If A assumes that the delivery failed due to network exceptions and retries, the same deduplication key will be used. At this point, consumer B can deduplicate the message based on the deduplication key.
  As implied in this example, message ID in TDMQ for CMQ cannot be used for deduplication, because the two messages have different IDs but the same body.

©2013-2019 Tencent Cloud. All rights reserved.

- Note: before sending a message, producer A should make the deduplication key persistent (such as writing it to the disk to avoid loss due to power failure).

How to prevent multiple messages with the same body from being filtered out:

- Scenario: A transferred 10 USD to B with five requests containing the same body committed. If B arbitrarily uses the message body as the decisive criterion for deduplication, those five requests will be mistaken for one.

- Solution: A can add information such as timestamp when producing the messages. In this way, even if the messages have the same body, the generated deduplication keys will be different, which makes it possible to send multiple messages with the same body.

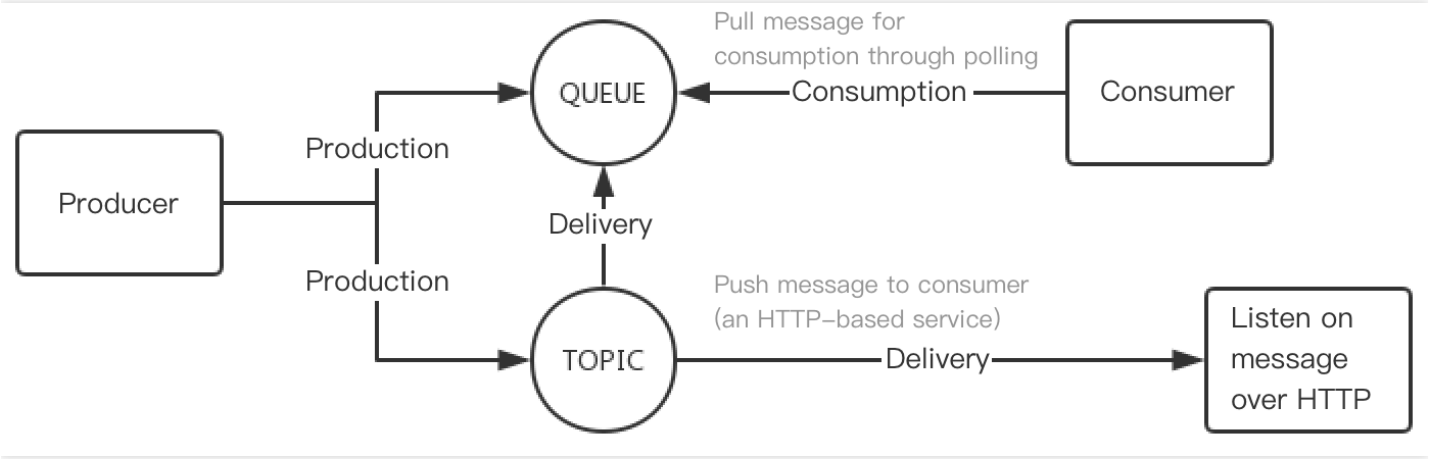
©2013-2019 Tencent Cloud. All rights reserved.

Push/Pull Selection

Last updated : 2022-02-11 11:39:45

TDMQ for CMQ supports two models: **push (topic)** and **pull (queue)**.

- Push model: when a message sent by the producer arrives at the server, the server will immediately deliver it to the consumer.
- Pull model: the server does not process the received messages; instead, it only waits for the consumer to actively read them from it, that is, the consumer needs to "pull" messages.



This document analyzes the pros and cons of the push and pull models in different scenarios.

Scenario 1. Fast producer and slow consumer

If the producer is faster than the consumer, there will be two possibilities: the efficiency of the producer itself is higher than that of the consumer (for example, the business logic of message processing on the consumer side may be very complicated or involve disk, network, and other I/O operations); message consumption fails or is hindered in a short time due to failure of the consumer.

In the push model, the server cannot know the current status of the consumer, so it will continuously push the generated data. The consumer may have a heavier load and even crash under the two aforementioned circumstances (for example, the producer is Flume which collects a massive number of logs, and the consumer is HDFS + Hadoop which lags behind the producer in processing efficiency), unless it has an appropriate mechanism that can inform the server of its status.

In the pull model, this problem becomes much simpler. As the consumer actively pulls data from the server, it only needs to reduce its access frequency. For example, if the log collection business such

©2013-2019 Tencent Cloud. All rights reserved.

as Flume on the frontend continuously generates messages to the TDMQ for CMQ instance that delivers messages to the backend, the efficiency of backend businesses such as data analysis may be lower than that of the producer.

Scenario 2. Real-timeness of messages

In the push model, once a message arrives at the server, the server can immediately push it to the consumer, which achieves a remarkable real-timeness. In the pull model, in order not to put pressure on the server (especially, when the data volume is insufficient, continuous polling will make no sense), it is important to control the consumer's polling interval, which will definitely compromise the real-timeness.

Scenario 3. Long polling in the pull model

The pull model has a problem where the consumer pulls messages proactively and thus cannot determine when exactly to pull the latest message. If the consumer successfully pulls a message, it can continue pulling the next one; otherwise, it needs to wait for a period of time to pull the message again.

As it is difficult to determine the waiting time period, you may use many dynamic pull interval adjustment algorithms. However, you still may encounter problems, as whether a message arrives is not determined by the consumer; for example, there may be 3,000 messages continuously arriving in 1 minute but no new message generated in the next several minutes or even hours.

TDMQ for CMQ provides an optimized mechanism called **long polling** to balance the advantages and disadvantages of the pull and push models. The basic principle is that when a consumer fails to pull a message, the failure will not be returned; instead, the connection will be in pending state to wait for the next message, and when the server receives a new message, it will activate the connection to return the latest message.

Scenario 4. Part or all of consumers in offline status

In a message queue system, the producer and consumer are completely decoupled from each other. When the producer sends messages, the consumer does not have to be online, and vice versa, which is exactly the main difference between message-based communication and RPC communication. There are many circumstances where a consumer is offline.

©2013-2019 Tencent Cloud. All rights reserved.

In case of accidental downtime or offline status of the consumer, the production by the producer will not be affected, and the consumer will continue to consume the last message when it goes back online; therefore, no message data will be lost. However, if the consumer is down for a long time or cannot be started again due to failures, there will be various issues to be addressed, such as whether the server needs to retain data for the consumer and how long the data should be retained.

In the push model, as it is impossible to predict whether the consumer will be down or offline temporarily or persistently, if all the historical messages since the crash are retained for the consumer, the data cannot be cleaned up even after all other consumers finish the consumption, and the queue will get increasingly longer over time. In this case, no matter whether the messages are stored temporarily in the memory or persistently on the disk (in systems using the push model, message queues are generally maintained in the memory for higher performance and real-timeness of push, which will be discussed in detail later), they will put huge pressure on the TDMQ for CMQ server and even affect the normal consumption of other consumers, especially when messages are produced at a high pace. However, if the data is not retained, it may be lost when the consumer is restarted.

A compromise is to set a timeout period for data in TDMQ for CMQ which will clean up the data when the consumer's downtime exceeds this threshold; however, the threshold is not easy to determine.

In the pull model, the situation is improved, as the server no longer cares about the consumer status; instead, it provides services only when the consumer actively communicates with it. The server does not make any guarantees about whether the consumer can consume the data in a timely manner (there is also a timeout period for cleanup).

©2013-2019 Tencent Cloud. All rights reserved.