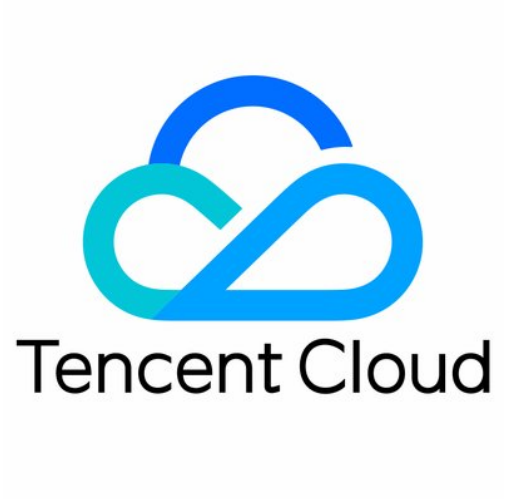
TDMQ for RocketMQ

Product Introduction

Product Documentation



Contents

Product Introduction

Overview

Last updated：2023-03-01 10:33:06

TDMQ for RocketMQ is distributed message middleware developed by Tencent Cloud based on Apache RocketMQ. It is fully compatible with all the components and principles of Apache RocketMQ and supports connection to open-source RocketMQ clients without any modifications.
Featuring low latency, high performance, reliability, and scalability, and trillions of QPS, TDMQ for RocketMQ can add async decoupling and peak shifting capabilities to distributed application systems. It also provides the capabilities necessary to internet applications, such as massive message retention, high throughput, and reliable retry mechanism.

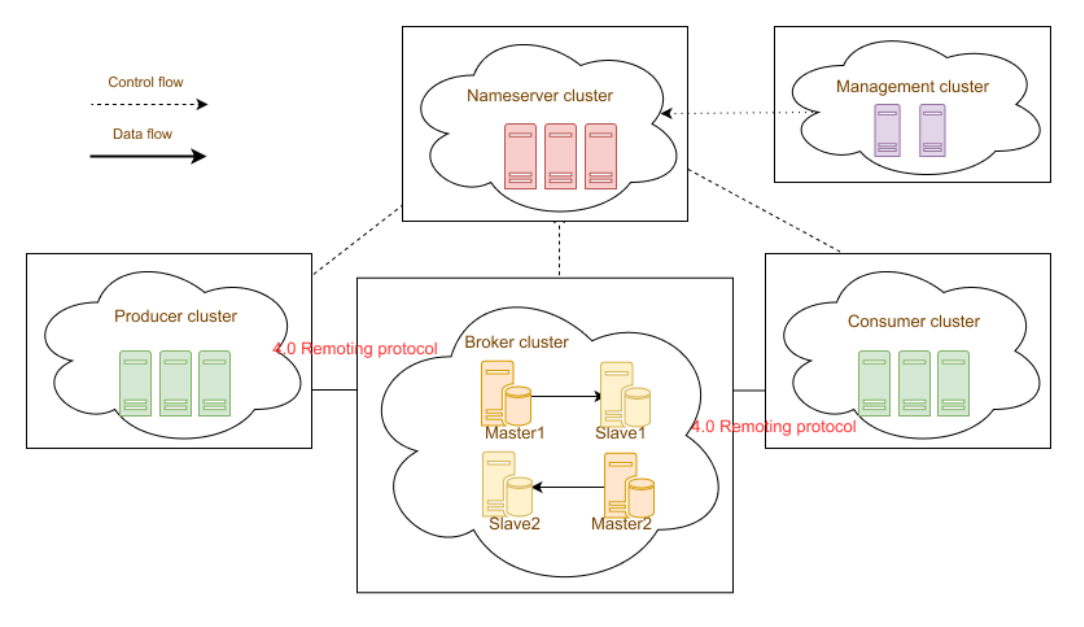
Architecture

Last updated：2023-04-14 16:50:11

This document describes the deployment architecture of TDMQ for RocketMQ to help you better understand its architectural principles.

Deployment Architecture

The system deployment architecture of TDMQ for RocketMQ is shown in the following diagram:



The core concepts are as follows:
Producer cluster: Client-side application, which is responsible for producing and sending messages.
Consumer cluster: Client-side application, which is responsible for subscribing to and consuming messages.
Nameserver cluster: server-side application, which is responsible for address routing and broker heartbeat registration.
Heartbeat registration: Nameserver acts as the registration center. Machines in each role must regularly report their status to Nameserver. If a machine fails to report beyond the timeout period, Nameserver will consider it faulty and unavailable and remove it from the available list.
Address routing: Each Nameserver stores the entire routing information of the Broker cluster and the queue information used for client queries. Producers and consumers obtain the routing information of the entire Broker cluster through Nameserver to deliver and consume messages.

Broker cluster: Server application, which is responsible for receiving, storing, and delivering messages. It supports primary-secondary multi-copy mode where the deployment of secondary nodes is optional. The actual high reliability of data in the production environment on the public cloud directly depends on the three copies of the cloud disk.
Management cluster: Server application that is a visual management and control console. It is responsible for operating the entire cluster, such as source data sending/receiving and management.

For the advantages of TDMQ for RocketMQ over self-built open-source Apache RocketMQ, see Comparison with Apache RocketMQ.

Concepts

Last updated：2023-10-19 10:38:05

This document lists the common concepts and their definitions in TDMQ for RocketMQ.

Message (`Message`)

A message is the physical carrier of information transmitted by the messaging system. It is the smallest unit of the produced or consumed data. A producer encapsulates the load and extended attributes of business data into messages and sends the messages to a TDMQ for RocketMQ broker. Then, the broker delivers the messages to the consumer based on the relevant semantics.

Topic (`Topic`)

A topic is the collection of a type of messages. It is the basic unit for message subscription in TDMQ for RocketMQ. Each topic contains several messages.

Message Tag (`MessageTag`)

Tags are used to categorize different types of messages in the same topic. Topic and tag are basically the first-level and second-level classifications of messages, respectively.

Message Queue (`MessageQueue`)

A message queue (also known as a message partition) is a physical entity for message storage, and a topic can contain multiple queues. Messages in a queue can only be consumed by one consumer rather than multiple consumers in one consumer group.

Message Offset (`MessageQueueOffset`)

Messages are stored in multiple queues of a specified topic based on the order in which they arrive at the TDMQ for RocketMQ broker. Each message has a unique coordinate of type `Long` in the queue, which is defined as the message offset.

Consumption Offset (`ConsumerOffset`)

A message is not removed from the queue immediately after it has been consumed by a consumer. TDMQ for RocketMQ will record the offset of the last consumed message based on each consumer group. Such an offset is defined as the consumption offset.

Message Index (`MessageKey`)

A message index is a message-oriented index property in TDMQ for RocketMQ. By setting the message index, you can quickly find the corresponding message content.

Producer (`Producer`)

A producer in TDMQ for RocketMQ is a functional messaging entity that creates messages and sends them to the broker. It is typically integrated into the business system to encapsulate data as messages and send them to the broker.

Consumer (`Consumer`)

A consumer is an entity that receives and processes messages in TDMQ for RocketMQ. It is usually integrated into the business system to obtain messages from TDMQ for RocketMQ brokers and convert the messages into information that can be perceived and processed by business logic.

Group (`Group`)

Groups include producer groups and consumer groups.
Producer group: It is the collection of the same type of producers that send the same type of messages with the same sending logic. If a producer sends transactional messages and crashes afterward, the broker will contact other producer instances in the producer group to commit or cancel the transaction.
Consumer group: It is the collection of the same type of consumers that consume the same type of messages with the same consumption logic. It can ensure load balancing and fault tolerance in the message consumption process. Consumer instances in a consumer group must subscribe to the same topics.

Message Type (`MessageType`)

Messages are classified by message transmission characteristic for message type management and security verification. TDMQ for RocketMQ has four message types: general message, sequential message, transactional message, and scheduled/delayed message.

General Message

The general message is a basic message type. After the produced general messages are delivered to a specified topic, they will be consumed by consumers that subscribe to this topic. A topic with general messages is sequence-insensitive. Therefore, you can use multiple topic partitions to improve message production and consumption efficiency. This approach performs best when dealing with high throughput.

Sequential Message

In TDMQ for RocketMQ, the sequential message is an advanced message type. Sequential messages in a specified topic are published and consumed in a First In First Out (FIFO) manner, that is, the first produced messages are first consumed.

Retry Letter Queue

A retry letter queue is designed to ensure that messages are consumed normally. When a message is consumed for the first time by a consumer but is not acknowledged, it will be placed in the retry letter queue and will be retried there until the maximum number of retries is reached. It will then be delivered to the dead letter queue.
In actual scenarios, messages may not be processed promptly due to temporary issues such as network jitter and service restart. The retry mechanism of the retry letter queue can be a good solution in this case.

Dead Letter Queue

A dead letter queue is a special type of message queue used to centrally process messages that cannot be consumed normally. If a message cannot be consumed after a specified number of retries in the retry letter queue, TDMQ will determine that the message cannot be consumed under the current situation and deliver it to the dead letter queue.
In actual scenarios, messages may not be consumed due to service downtime or network disconnection. In this case, they will not be discarded immediately; instead, they will be persistently stored in the dead letter queue. After fixing the problem, you can create a consumer to subscribe to the dead letter queue to process such messages.

Clustering Consumption

Clustering consumption: If the clustering consumption mode is used, each message only needs to be processed by any of the consumers in the cluster. This mode is suitable for scenarios where each message only needs to be processed once.

Broadcasting Consumption

Broadcasting consumption: If the broadcasting consumption mode is used, each message will be pushed to all registered consumers in the cluster to ensure that the message is consumed by each consumer at least once. This mode is suitable for scenarios where each message needs to be processed by each consumer in the cluster.

Message Filtering

A consumer can filter messages by subscribing to specified message tags to ensure that it only receives the filtered messages. The whole filtering process is completed in the TDMQ for RocketMQ broker.

Consumption Offset Reset

Resetting the consumption offset means resetting a consumer group's consumption offset for subscribed topics within the persistent message storage period based on the time axis. After the offset is reset, the consumers will receive the messages that the producer sends to the TDMQ for RocketMQ broker after the set time point.

Message Trace

The message trace records the entire lifecycle of a message from the time it is sent by the producer to the time it is received and processed by the consumer. With this feature, you can track the entire trace of a message, starting from its production by a producer, its storage and distribution within the TDMQ for RocketMQ broker, and finally its consumption by one or more consumers. This helps you troubleshoot any problems that may occur during message processing.

Message Heap

Message heap occurs in scenarios where the producer has sent messages to the TDMQ for RocketMQ broker but the consumer fails to normally consume all these messages promptly due to its consumption capability limit. In this case, the unconsumed messages will be heaped in the broker. The message heap data is collected once every minute. After the message retention period (3 days by default) elapses, the unconsumed messages will no longer be heaped in the broker because they have been deleted by the broker.

Strengths

Last updated：2024-01-18 09:45:29

Open-Source Version Compatibility

TDMQ for RocketMQ is compatible with open-source RocketMQ 4.3.0 and later. It supports access from open-source clients in Java, C, C++, Go, and other programming languages.

Resource Isolation

TDMQ for RocketMQ offers a multi-level resource structure that allows for both namespace-based virtual isolation and cluster-level physical isolation, making it simple for you to enable namespace-level permission verification to distinguish clients in different environments.

Rich Message Types

TDMQ for RocketMQ supports multiple message types such as general, sequential, delayed, and transactional messages. It also supports message retry and the dead letter mechanism, fully meeting the requirements in various business scenarios.

High Performance

A single TDMQ for RocketMQ server can sustain a production/consumption throughput of up to 10,000 messages. With the distributed architecture and stateless services, the cluster can be scaled horizontally to increase the cluster throughput.

Observability

TDMQ for RocketMQ supports various monitoring metrics in the console where message traces can be displayed. It also provides alarming capabilities and all the TencentCloud APIs you may need to integrate with your self-service Ops systems.

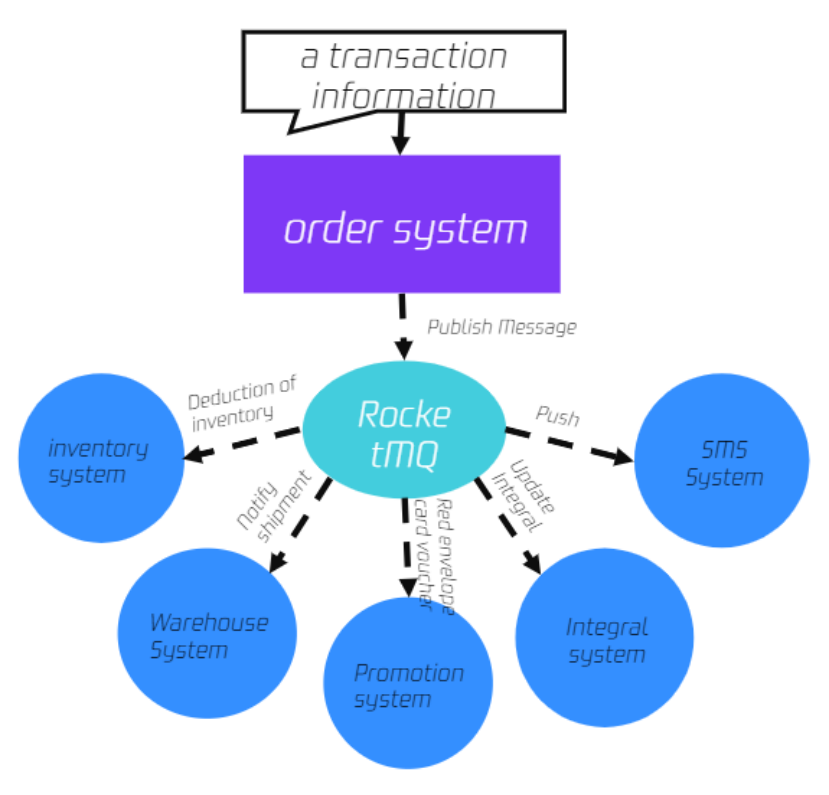
Use Cases

Last updated：2023-09-12 16:02:09

TDMQ for RocketMQ is a distributed message middleware based on Apache RocketMQ. It is applied to message communication between distributed systems or components. It has the characteristics of massive message heap, low latency, high throughput, high reliability, and strong transaction consistency, which meets the requirements of async decoupling, peak shifting, sequential sending and receiving, distributed transaction consistency, and log sync.

Async Decoupling

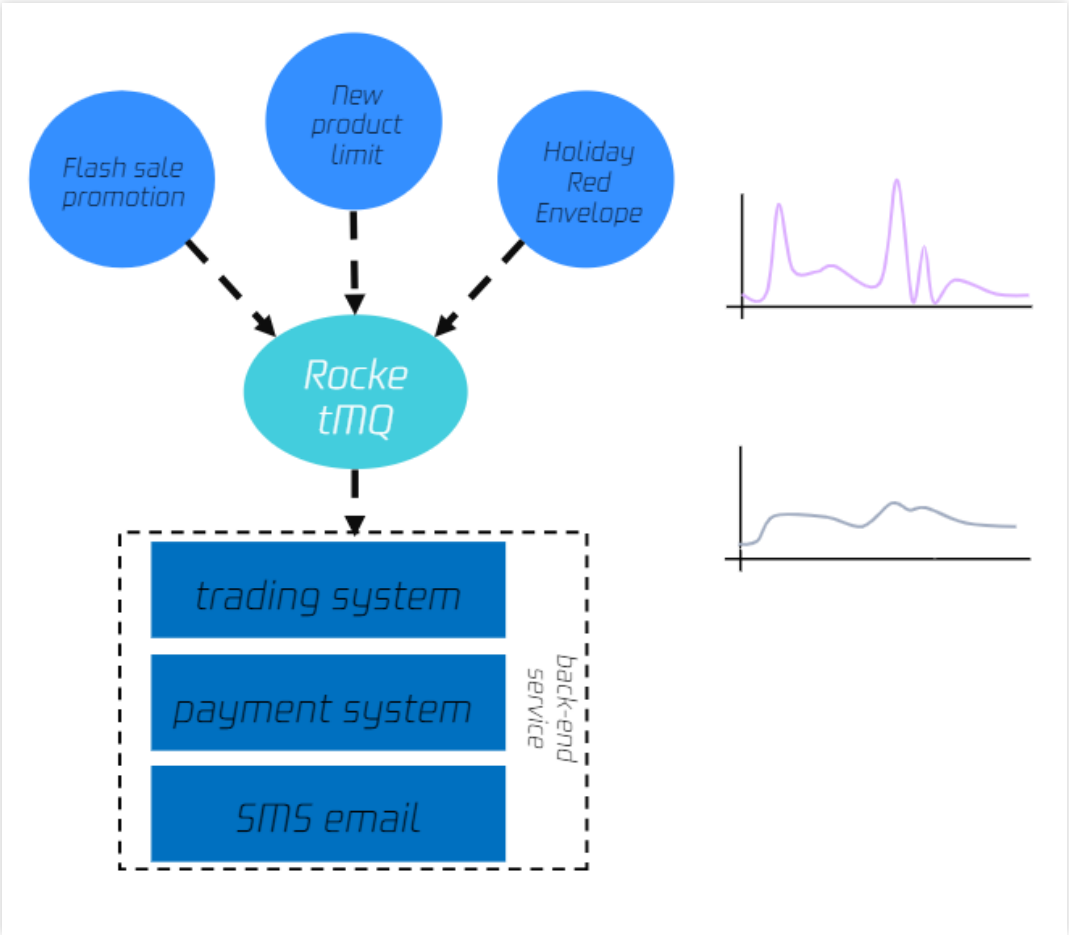
The transaction engine is the core system of Tencent billing. The data of each transaction order needs to be monitored by dozens of downstream business systems, including inventory system, warehousing system, promotion system, and points system. Such systems use different message processing logic, making it impossible for a single system to adapt to all associated business. In this case, TDMQ for RocketMQ can decouple the coupling between multiple business systems to reduce the impact between systems and improve the response speed and robustness of core business.



Peak Shifting

Companies hold promotional campaigns such as new product launch and festival red packet grabbing from time to time, which often cause temporary traffic spikes and pose huge challenges to each backend application system. In this case, TDMQ for RocketMQ can withstand spikes in traffic. It heaps up messages during peak periods and consumes

them in the downstream during off-peak periods, which balances the processing capacities of upstream and downstream systems and improve system availability.



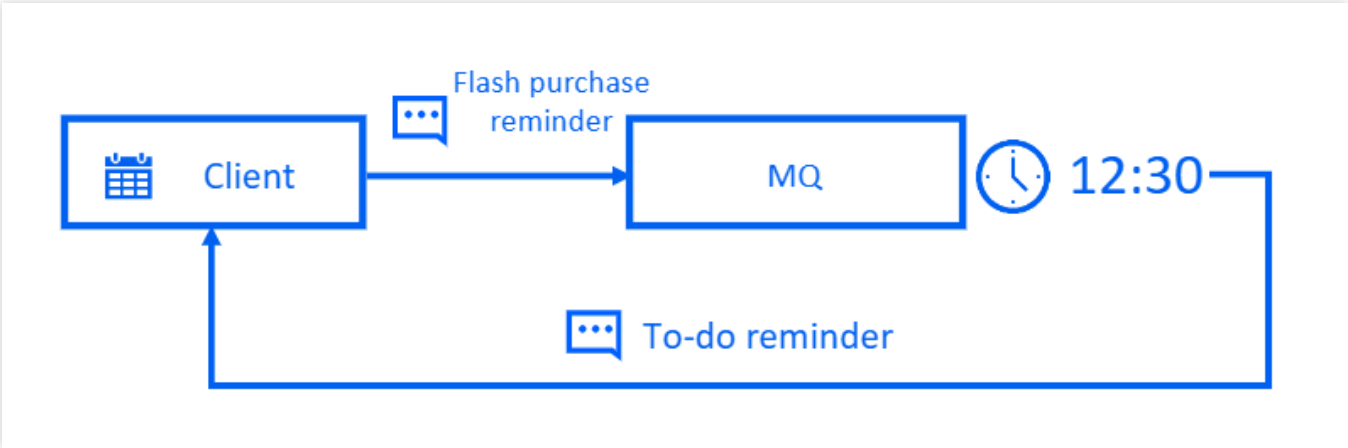
Subscription Notifications

TDMQ for RocketMQ provides scheduled and delayed messages to can meet the ecommerce subscription notification scenarios.

Scheduled message: After a message is sent to the server, the business may want the consumer to receive it at a later time point rather than immediately. This type of message is called "scheduled message".

Delayed message: After a message is sent to the server, the business may want the consumer to receive it after a period of time rather than immediately. This type of message is called "delayed message".

For details about scheduled and delayed messages, see Scheduled Message and Delayed Message.



Consistency of Distributed Transactions

TDMQ for RocketMQ provides distributed transactional messages to loosely couple applications. Reliable transmission and multi-replica technology can ensure that messages are not lost, and the At-Least-Once feature ensures eventual data consistency.

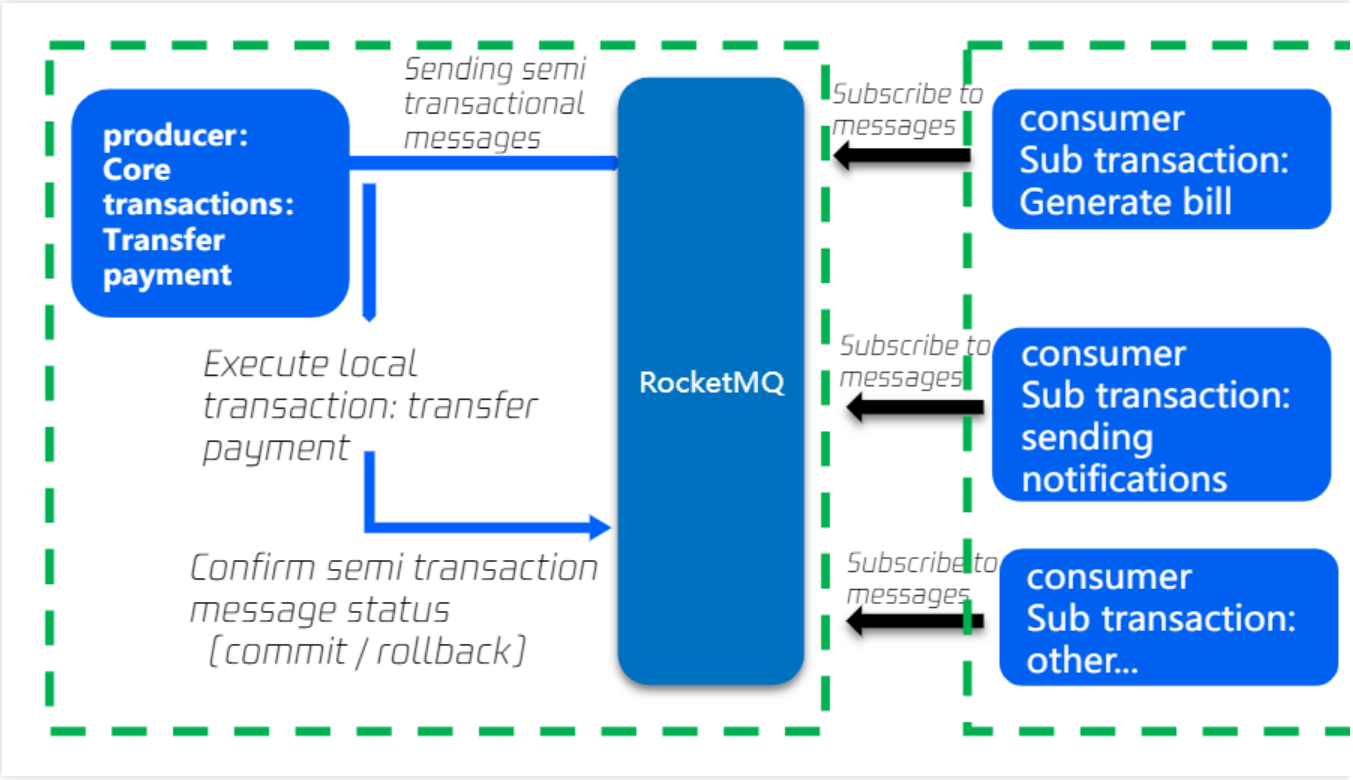
As a producer, the payment system forms a transaction with the message queue to ensure the consistency of local transactions and message sending.

Downstream business systems (bills, notifications, others) work as consumers to process in parallel.

Messages support reliable retries to ensure eventual data consistency.

The transaction messages of TDMQ for RocketMQ can be used to process transactions, which can greatly improve processing efficiency and performance. A billing system often has a long transaction linkage with a significant chance of error or timeout. TDMQ's automated repush and abundant message retention features can be used to provide transaction compensation, and the eventual consistency of payment tips notifications and transaction pushes can also be achieved through TDMQ for RocketMQ.

For details about transactional messages, see Transactional Message.



Sequential Message Sending/Receiving

Sequential message is an advanced message type provided by TDMQ for RocketMQ. For a specified topic, messages are published and consumed in strict accordance with the principle of First-In-First-Out (FIFO), that is, messages sent first are consumed first, and messages sent later are consumed later. Sequential messages are often used in the following business scenarios:

Order creation: In some ecommerce systems, an order's creation, payment, refund, and logistics messages must be produced or consumed in strict sequence, otherwise the order status will be messed up during consumption, which will affect the normal operation of the business. Therefore, the messages of this order must be produced and consumed in a certain sequence in the client and message queue. At the same time, the messages are sequentially dependent, and the processing of the next message must be dependent on the processing result of the preceding message.

Log sync: In the scenario of sequential event processing or real-time incremental data sync, sequential messages can also play a greater role. For example, it is necessary to ensure that database operations are in sequence when MySQL binlogs are synced.

Financial scenarios: In some matchmaking transaction scenarios like certain securities transactions, the first bidder is given priority in the case of the same biding price, so it is necessary to produce and consume sequential messages in a FIFO manner.

For details about sequential messages, see Sequential Message.

Distributed Cache Sync

During sales and promotions, there are a wide variety of products with frequent price changes. When users query item prices multiple times, the cache server's network interface may be fully loaded, which makes page opening slower. After the broadcast consumption mode of TDMQ for RocketMQ is adopted, a message will be consumed by all nodes once, which is equivalent to syncing the price information to each server as needed in place of the cache.

Use Limits

Last updated：2023-09-12 17:53:17

This document lists the limits of certain metrics and performance in TDMQ for RocketMQ. Be careful not to exceed the limits during use so as to avoid exceptions.

Cluster

Limit	Virtual Cluster	Exclusive Cluster
Maximum number of clusters per region	10	Unlimited
Cluster name length	3-64 characters	3-64 characters
Maximum TPS	4000	Above 4,000, subject to the node specification
Maximum bandwidth (production + consumption) per cluster	40 Mbps	Above 80 Mbps, subject to the node specification

Namespace

Limit	Virtual Cluster	Exclusive Cluster
Maximum number of namespaces per cluster	10	10
Namespace name length	3-32 characters	3-32 characters

Topic

Limit	Virtual Cluster	Exclusive Cluster
Maximum number of topics per cluster	150	200-500, subject to the node specification
Topic name length	3-64 characters	3-64 characters
Maximum number of producers per topic	1000	1000
Maximum number of consumers per topic	500	500

Group

Limit	Virtual Cluster	Exclusive Cluster
Maximum number of groups per cluster	1500	2000-5000, subject to the node specification
Group name length	3-64 characters	3-64 characters

Message

Limit	Virtual Cluster	Exclusive Cluster
Maximum message retention period	3 days	3 days
Maximum message delay	40 days	40 days
Message size	4 MB	4 MB
Consumption offset reset	3 days	3 days

Comparison with Apache RocketMQ

Last updated：2024-01-18 09:53:49

The performance comparison between TDMQ for RocketMQ and Apache RocketMQ is detailed below:

Feature Type	Feature	TDMQ for RocketMQ	Apache RocketMQ
Basic features	Scheduled message	The scheduled time is accurate down to the second and can be customized.	You can only specify the delay level.
	Visual management	Visual management for clusters, topics, and groups is supported. You can view the details of subscriptions and consumer status.	Visual management is supported but is less user-friendly. The console doesn't distinguish between topic types.
Availability	Elastic scaling	You don't need to manually deploy, configure, or scale up underlying computing resources because operations such as node registration are automatically performed in a visual manner. You can expand the number of nodes horizontally, increase the disk capacity, and upgrade the configurations of a single node vertically as needed at any time.	A self-built Ops team is required, and operations are performed in a less automatic or visualized manner.
	High reliability	With three data replicas, the server can be automatically restarted in seconds after the downtime, without affecting the message capacity and data.	Data can be replicated in sync or async mode. You need to design the deployment scheme and related parameters. The primary sync schemes won't be automatically used after the failover.
	Cross-AZ high-availability deployment	This feature is supported to avoid losses caused by data center-level failures.	This feature is supported, but it is time-consuming for you to design the deployment schemes and parameters.
Observability	Resource dashboard	You can monitor core metrics at a fine granularity and view production and consumption details.	This feature is supported but with fewer monitoring metrics.

	Alarming	With the capabilities provided by Cloud Monitor, alarms will be triggered in case of message heap or delayed message sending/receiving.	Not supported
Security management and control	Tenant namespace isolation	You can implement this feature in the console in a visual manner.	This feature is not supported. Namespaces cannot be truly isolated due to bugs.
	Root account and sub-account management	Supports authorization between Tencent Cloud CAM root accounts and sub-accounts and between enterprise accounts.	Not supported
Migration tool	Tool for migrating from Apache RocketMQ	You can easily migrate from Apache RocketMQ to TDMQ for RocketMQ by using scripts.	-