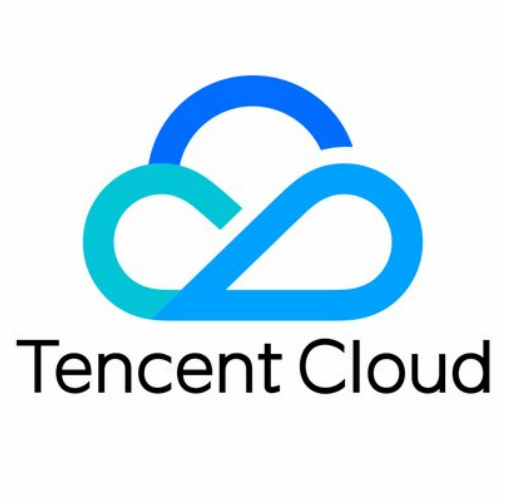
Cloud Message Queue

Solutions and Cases

Product Documentation



Contents

Solutions and Cases
Comparative Analysis of Cloud CMQ and RabbitMQ

Last updated : 2020-07-20 16:23:52

RabbitMQ is a typical open-source messaging middleware program. It is popular among enterprise systems and suitable for scenarios with high requirements for data consistency, stability, and reliability.

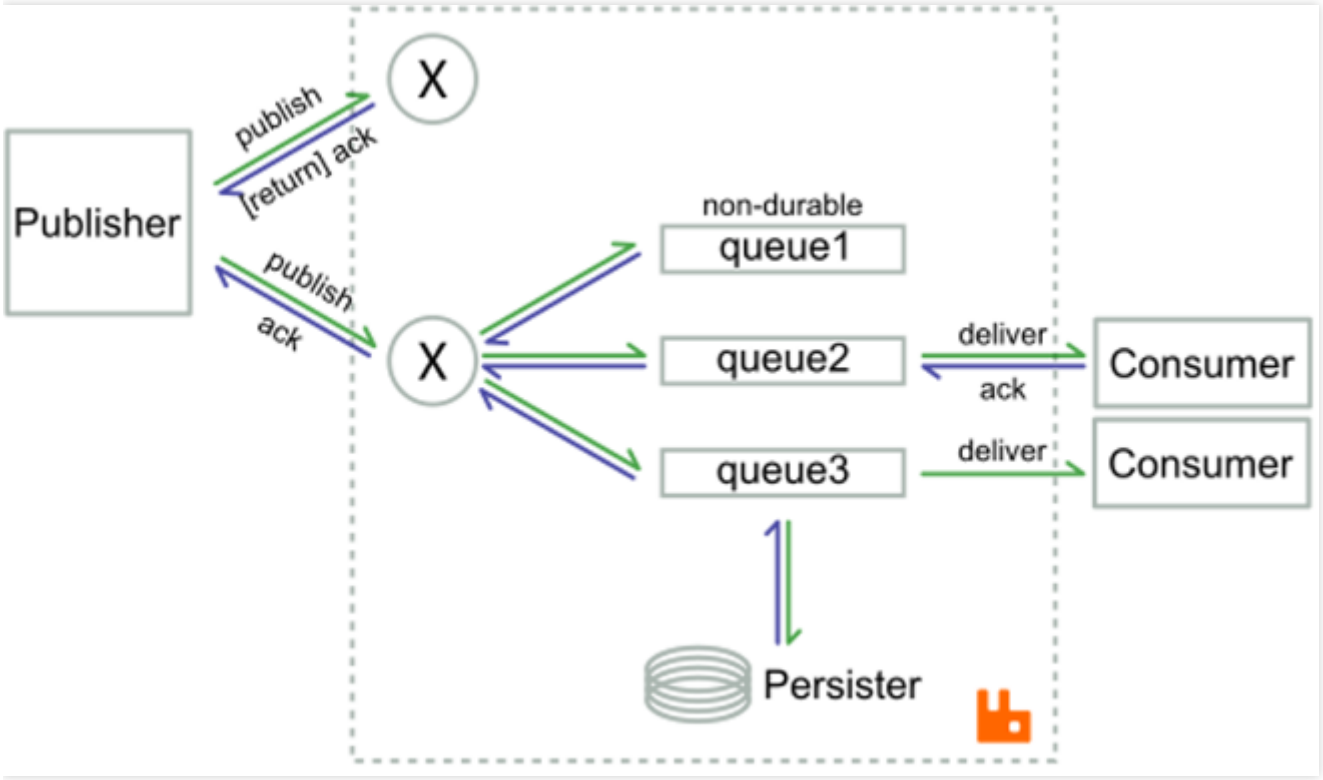
Based on the high reliability of RabbitMQ/AMQP and by leveraging the Raft protocol in redesign and reimplementation, Tencent Cloud CMQ greatly improves the reliability, throughput, and performance.

This document describes RabbitMQ's reliability principle, improvement made by CMQ, and performance comparison between them.

Reliable Message Delivery of RabbitMQ

ACK mechanism

A business may lose messages due to various issues such as network, server, or program exceptions. The message ACK mechanism can solve the problem of message loss. If a message is successfully acknowledged, it means that the message has been verified and correctly processed.



RabbitMQ uses the mechanism of produced and consumed message ACK to ensure reliable delivery.

- Produced message ACK: after the producer sends a message to the message queue, it will wait for the return of an ACK of success; otherwise, the producer will resend the message to the message queue. This process can be async, that is, the producer continuously sends messages, and the message queue can return ACKs after processing them in batches. The producer can identify the message IDs in the returned ACKs to determine which messages have been successfully processed.

- Consumed message ACK: after the message queue delivers a message to the consumer, it will wait for the return of an ACK of success; otherwise, the message queue will resend the message to the consumer. This process can also be async, that is, the message queue continuously delivers messages, and the consumer can return ACKs after processing them in batches.

You can see that RabbitMQ/AMQP provides "at-least-once delivery". When exceptions occur, messages will be repeatedly delivered or consumed.

Message storage

To improve message reliability and ensure that received messages can be persistently written into the disk when the service is unavailable during RabbitMQ restart, RabbitMQ will write received

messages to a file and store it in the disk when the number of written messages reaches a certain value or after a certain period of time.

Produced message ACK is performed after message storage, and the message queue will return the ID of the stored message to the producer.

Comparison Between CMQ and RabbitMQ

CMQ has a lot of similarities with RabbitMQ in underlying principles and implementation methods; however, it is greatly upgraded and improved compared with RabbitMQ:

Feature upgrade

In addition to produced and consumed message ACK mechanisms, CMQ provides the message rewind feature.

You can specify the number of days during which CMQ stores produced messages. Then, the messages can be rewound back to a time point in this period for consumption again starting from the time point. The message rewind feature is very useful in business restoration when the business logic is exceptional.

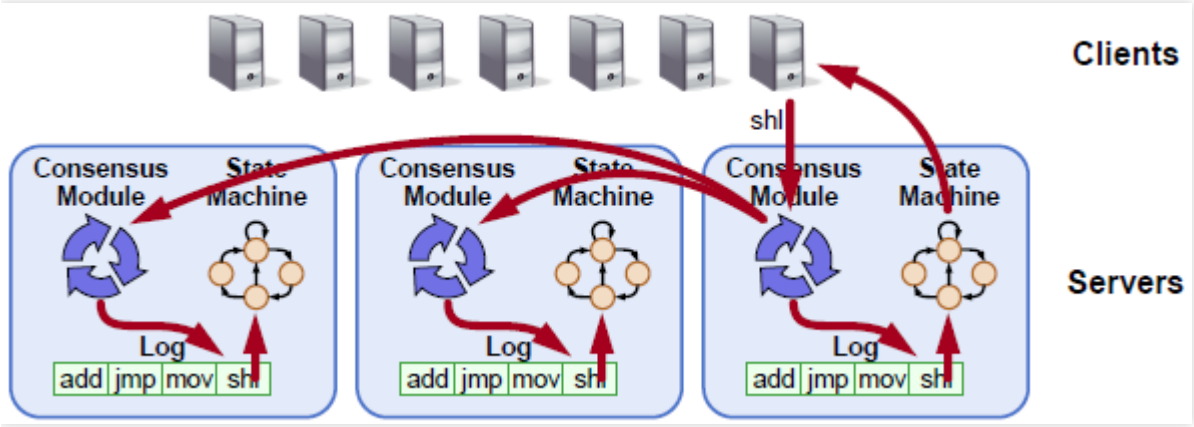
Performance optimization

Performance Metric	Description
Network IO	CMQ can produce/consume messages in batches, while RabbitMQ does not support batch production. In scenarios where there are high numbers of small-sized messages, CMQ can process them with fewer requests at a smaller average latency.
File IO	CMQ writes single messages in sequence during message production/consumption and regularly stores them in the disk, making full use of the file system for caching. In the persistent message mechanism of RabbitMQ, messages are sent to the in-memory queue for status conversion, then written into log cache, and finally written into message and index files (the index file is written sequentially, while the message file is written randomly), which involves three IO operations and has relatively low performance.
CPU performance	Computing for RabbitMQ log caching and status conversion is complicated and needs a lot of CPU resources, therefore compromising the performance.

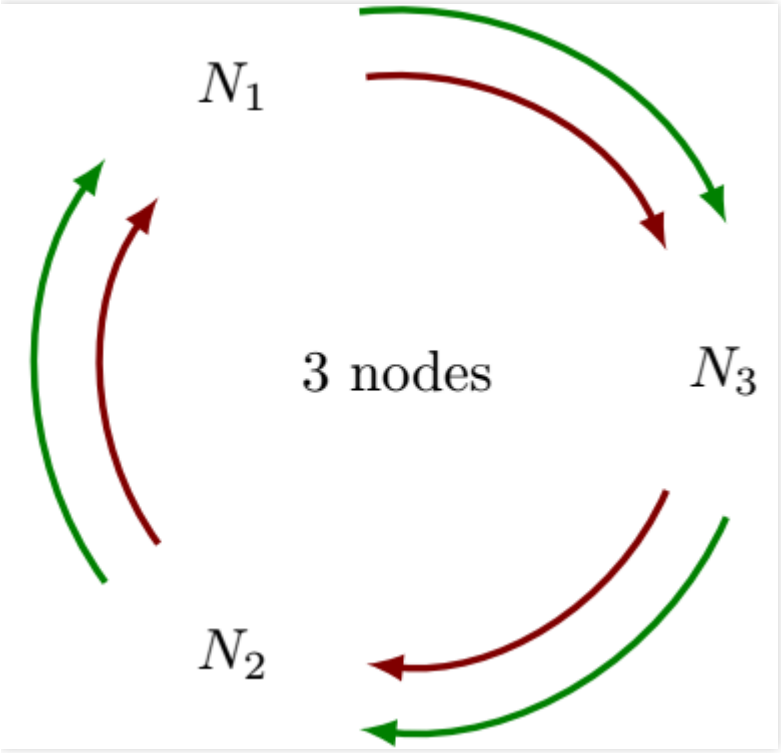
Improved availability

Both CMQ and RabbitMQ support hot backup with multiple servers to improve availability. CMQ implements this feature based on the Raft algorithm which is simple and easy to maintain, while RabbitMQ uses its proprietary Guaranteed Multicast (GM) algorithm which is difficult to learn.

In the Raft protocol, logs can be replicated as long as most nodes return a success to the leader, and the leader can implement the request and return a success to the client.



In GM, all nodes in the cluster are organized as a ring. A log replication request will be sent to subsequent nodes one by one after the leader. The leader will send an ACK message to the ring after it receives the request again. Only after the ACK is received by the leader again can the log be fully synced among all nodes in the ring.



The GM algorithm requires that success be returned only after logs have been synced among all nodes in the cluster, while the Raft algorithm requires sync only among most nodes, which reduces the waiting time in sync route by almost a half.

Performance comparison between CMQ and RabbitMQ

The testing scenario is as follows: three servers with the same configuration form a cluster, CMQ and RabbitMQ are both configured as image queues, and all data is synced on the three servers. CMQ and RabbitMQ both have the produced and consumed message ACK mechanisms enabled, and the size of produced messages is about 1 KB each.

Test Environment	Description
CPU	24-core Intel(R) Xeon(R) CPU E5-2620 v3 @ 2.40 GHz
Memory	64 GB
Disk	12 * 2 TB SATAs RAID 0 of 12 SATAs
ENI	10-Gigabit
Linux version	2.6.32.43
RabbitMQ version	3.6.2
Erlang version	18.3

The testing data is as follows:

QPS Comparison	Production Only	Consumption Only	Simultaneous Production/Consumption
CMQ	Production: 68,000 messages/sec	Consumption: 90,000 messages/sec	Production: 36,000 messages/sec Consumption: 36,000 messages/sec
RabbitMQ	Production: 12,500 messages/sec	Consumption: 26,000 messages/sec	Production: 8,500 messages/sec Consumption: 8,500 messages/sec

In high-reliability scenarios, the throughput of CMQ is four times higher than that of RabbitMQ.

Third-Party Payment

Last updated : 2020-12-18 12:53:52

The close collaboration between leading third-party mobile finance payment solution providers such as SwiftPass and WeChat Pay promotes the use of WeChat Pay among brick-and-mortar stores in various industries, which helps reduce the need of cash payments and improve the payment efficiency. The main architecture of the payment system is as follows:

1. When a user submits a payment request at a convenience store (such as 7-Eleven) to WeChat Pay, WeChat Pay will return an ACK after confirming the request.

2. After the returned ACK is confirmed, WeChat Pay will deliver a **successful order payment message** to SwiftPass, describing the account information, time, amount, and device information of that transaction.

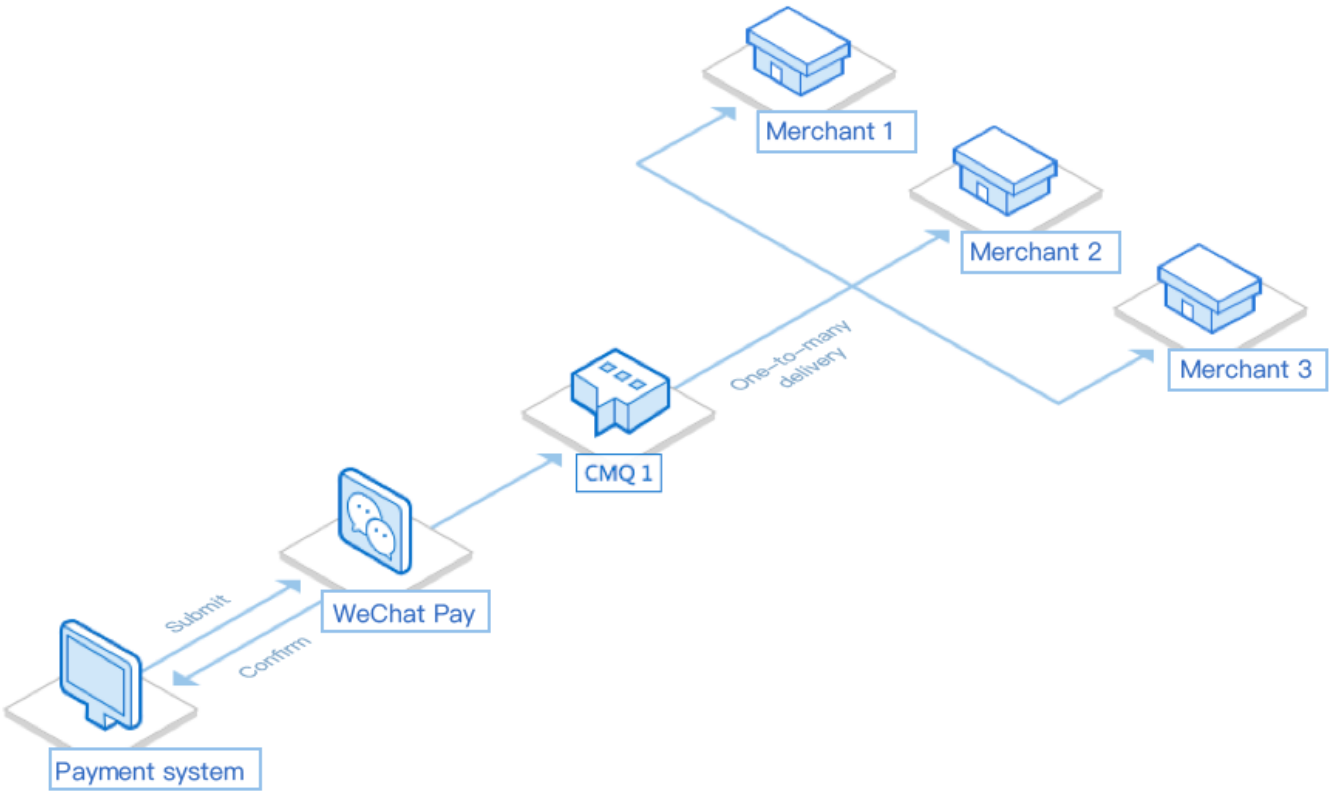
3. SwiftPass writes the message details to CMQ for temporary storage. The **successful order payment message** will be used as an important credential for settlement between SwiftPass and the merchant (i.e., the convenience store); therefore, it must be reliably delivered with guaranteed arrival.

4. The **order payment message** in CMQ will be returned to the server of the merchant (i.e., the convenience store), which can be async since it does not need to be in real time. Specifically, it will be written to a queue, pulled by an HTTP proxy, and then sent to the merchant after being fetched out.

5. Before CMQ is connected, if SwiftPass fails to notify the merchant, it will initiate a new request to WeChat Pay which then will deliver the same **order payment message** to SwiftPass. After CMQ is connected, from WeChat's perspective, the success rate of SwiftPass' system is greatly improved, and its rating (reliability and credibility) in WeChat system will be increased.

6. Finally, all **order payment messages** will be continuously delivered by another topic to the risk management, campaign management, and promotion campaign systems. For example, the risk management system will continuously analyze the order payment conditions in each message delivered by the topic. If the transaction amount of merchant A soars in a short period of time (suspiciously fake orders), the callback API will be used to prohibit subsequent transactions for merchant A.

Please see the following figure:



Starting point literary network case

Last updated : 2020-05-12 21:11:45

CMQ meets the three key needs of Qidian.com operated by China Literature:

1. In the **Zhangyishucai** operation system, consumer crediting in the feature of grabbing red packet monthly tickets is async. The credit information will be first written to a message queue and then pulled by the consumer. After the consumer confirms that the message is successfully consumed, the callback API will delete the message from the queue.

2. In another scenario, major systems of Qidian.com such as OPS, alarming, and operations generate massive volumes of logs. The logs will be first aggregated into CMQ, and the backend big data analysis clusters will continuously pull them out of CMQ and analyze them based on the processing capabilities. CMQ can theoretically retain an unlimited number of messages, bringing you complete peace of mind when using it.

3. A feature similar to message rewind in Kafka is provided. After business consumption is successfully completed and the messages are deleted, message rewind can be used to consume the deleted messages again. The offset position can be specified for flexible adjustment. This feature facilitates Qidian.com in reconciliation and business system retry.

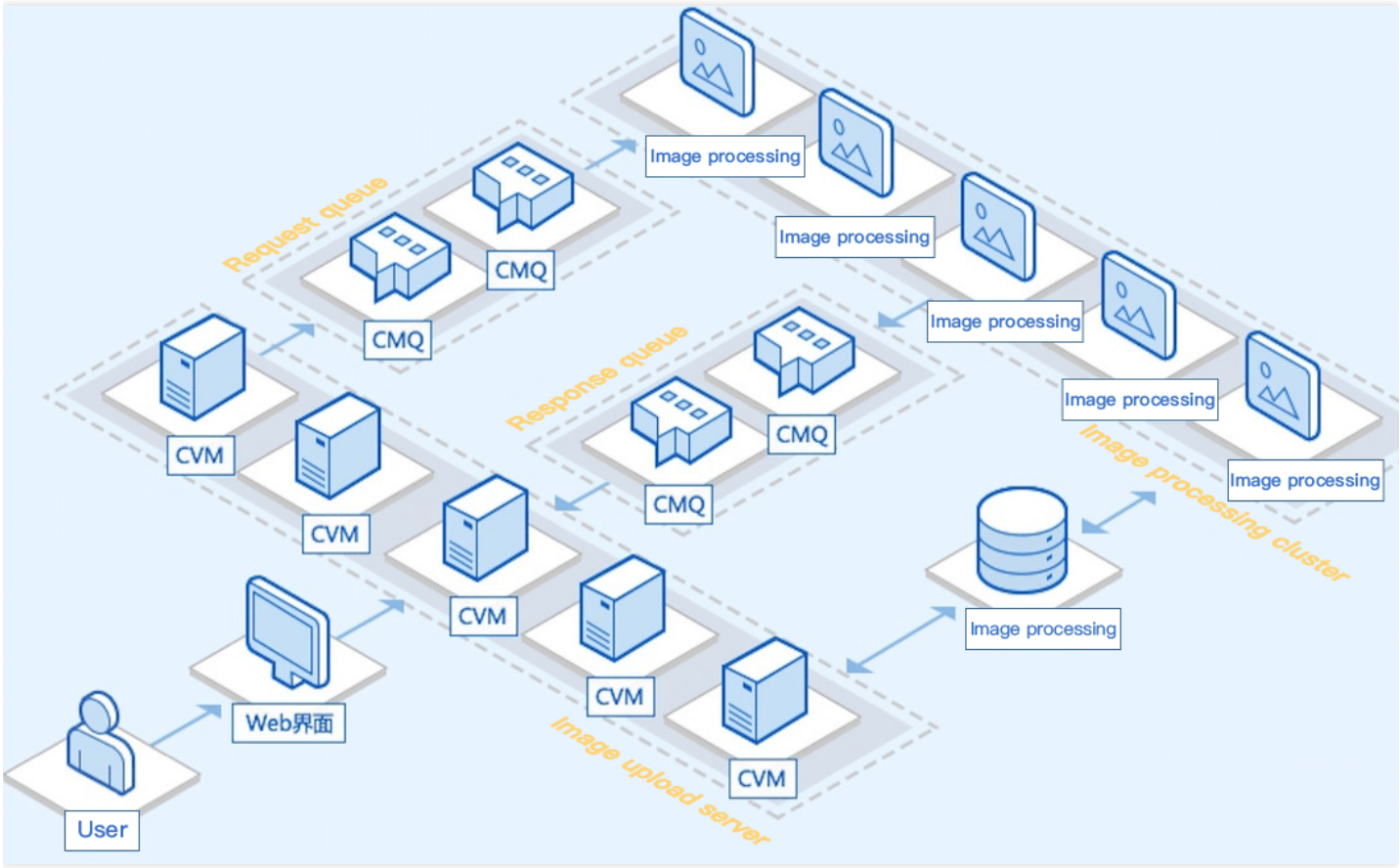
The overall business of Qidian.com presents high pressure on CMQ, as the API request QPS exceeds 100,000, and the total number of daily requests exceeds 1 billion. CMQ can easily support Qidian.com with high stability under such huge business pressure.

The CMQ backend cluster is imperceptible to users, and the CMQ controller server can schedule and relocate queues in real time based on the load of the cluster. If the request volume of a queue exceeds the service threshold of the current cluster, the controller server can distribute the queue routes to multiple clusters to increase the number of processable concurrent requests. In theory, CMQ can achieve unlimited message retention and extremely high QPS.

Online Image Processing

Last updated : 2020-12-18 14:16:39

An image processing company builds an online image processing service in Tencent Cloud which enables users to upload their images and specify operations to be performed on them, such as cropping, red eye removal, teeth whitening, colorization, contrast adjustment, and thumbnail generation. A user can upload an image, submit a task, wait for the image to be processed, and download the output image. The time taken for processing varies by operation, from several seconds to several minutes, and the user may upload several, dozens of, or hundreds of images at a time. Therefore, the total processing time is subject to the number of uploaded images, image size, and selected operations.



After CMQ is integrated to implement the abovementioned needs, user images will be stored in Tencent Cloud storage (such as CBS and COS), and each user operation request will be stored in the request queue as a message. The message content is an image index composed of elements such as the image name, operation type in user request, and image storage location index key.

The image processing service running on CVM gets a message (image index) from the request queue. The image processing server downloads the data from the cloud and edits the image. Then, it sends the processing result to the response queue and stores the output image in cloud storage. After this process is completed, the user has stored all the original and output images in cloud storage and can download them for use at any time.

More details about scalability and high reliability:

- Even if the image processing service is temporarily unavailable due to bugs or other problems, as CMQ is used, the crash will be imperceptible to the user. In this case, on one hand, the user can still upload images, and the web server can still send messages to the request queue where the messages will be retained and can be fetched out only after the image processing service is back online; on the other hand, the image processing service does not need to record the messages being processed before the crash when it is implemented, and such messages can be processed again, as the message (including messages received sequentially and concurrently in the queue) receipt feature of CMQ ensures that messages can remain in the queue after being received until they are explicitly deleted by the recipient. This feature ensures decoupling of the image processing service and the image upload service.

- If a single image processing service cannot meet user needs (i.e., users can upload images but cannot get the processing results after waiting for a long time), you can use CMQ to start multiple image processing services to satisfy ever-increasing user access needs based on the following two characteristics of CMQ:
  - A single CMQ queue can be accessed by multiple servers simultaneously (i.e., message sending, receipt, and deletion can be concurrent).
  - A message will not be received by multiple services, which is implemented by the temporary message lock. The message recipient can specify the time during which the message is locked and needs to proactively delete the message after processing it. If the recipient fails to process the message, another service can get the message again after the lock expires.

These two characteristics ensure that the number of processing servers can be dynamically adjusted by load.

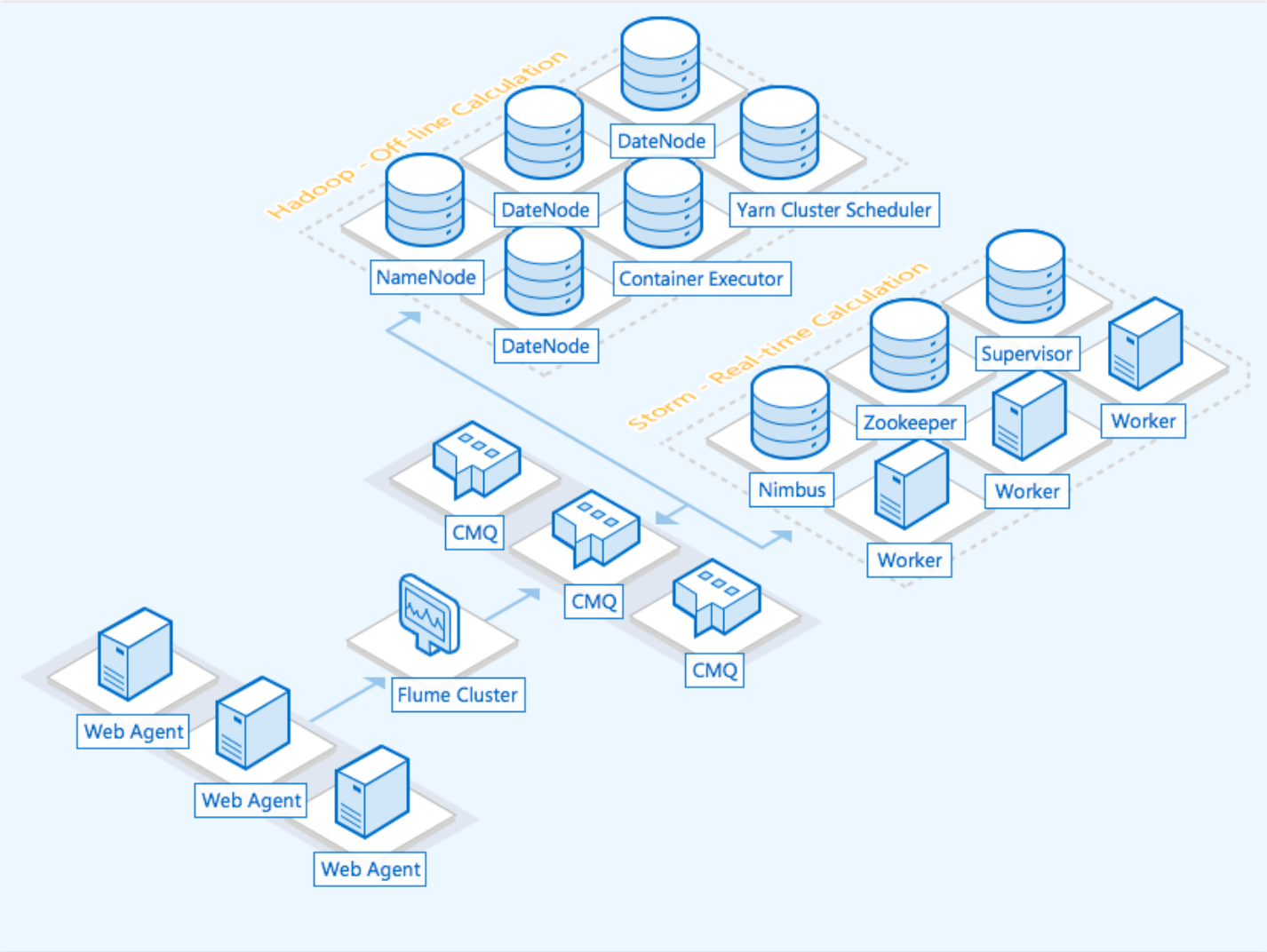
Massive Data Processing

Last updated : 2020-07-20 16:23:53

The first step in big data processing is to mine and analyze massive amounts of data in order to extract useful results and guide future business models. For example, Dianping.com and Didi Chuxing have made in-depth practices in Tencent Cloud services: the mobile app of Dianping.com pushes restaurants that are frequently shared by users on WeChat and Mobile QQ to its own consumers as recommendations.

The data analysis system features can be mainly divided into the following modules: data collection, data ingestion, stream computing, offline computing, and data persistence.

- **Data collection**
  It collects data from all nodes in real time by using open-source Flume. Data such as logs of all business servers will flow to the CMQ pipeline in the form of a funnel.
- **Data ingestion**
  As data collection and processing may be async in speed, to ensure stability and reliability of data write and analysis, the CMQ messaging middleware can be added as a buffer.
- **Stream computing and offline data analysis**
  Collected data is analyzed in real time with Apache Storm. Data mining is performed through offline data analysis based on Spark.
- **Data output**
  Analysis results can be persistently stored in services such as TencentDB for MySQL.



In data processing scenarios, the input of massive amounts of log data is the message producer, and the Storm cluster for online analysis is the message consumer. Practical experience shows that the message processing business logic of Storm as the message consumer may be complicated (which involves real-time computing, data stream processing, and topology data processing). Moreover, faults occur at a relatively high probability on Storm, which results in temporary consumption failures or instability. In summary, the efficiency of the message producer is much higher than that of the message 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 if the push model is used when the Storm cluster is under high load, 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.

CMQ will launch the topic model with the pull and push data acquisition modes in the future. As a buffer between producer data and consumer data, CMQ enables data to be read only when the

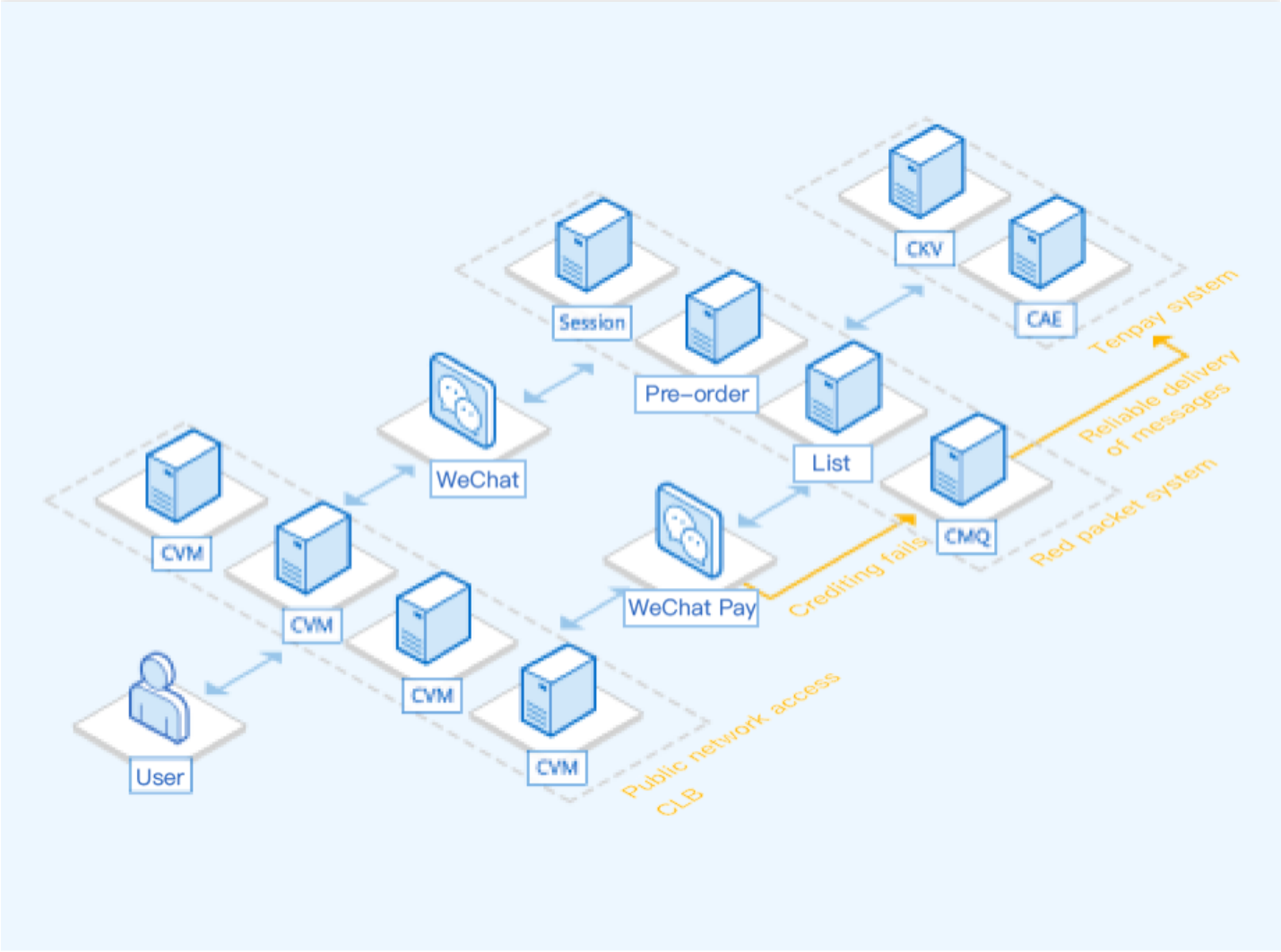
consumer is available and ready, which alleviates the out-of-sync issues between message producer and consumer.

WeChat Red Packet During Spring Festival Gala

Last updated : 2020-12-18 12:57:37

The Spring Festival Gala red packet campaign involves interactions between four major systems, namely, WeChat, WeChat Pay, red packet system, and Tenpay, which are described in detail below:

- Red packet system: it allows users to send, grab, and open personal red packets and view related lists.
- Tenpay: it supports payment of orders, high-performance storage of asynchronously credited transactions, and real-time display of user account balance and bills.
- WeChat: it ensures the quality of WeChat user access over the internet.
- WeChat Pay: it is the entry to online transactions.



Distributed transaction processing similar to the red packet system is the key focus. For example, **user A sends a red packet of 10 CNY to user B** with the following steps involved:

1. Read the balance of user A's account
2. Deduct 10 CNY from user A's account (debit)
3. Write the result to user A's account (ACK)
4. Read the balance of user B's account
5. Open the red packet sent by user A to user B and read the amount
6. Add 10 CNY to user B's account (credit)
7. Write the result to user B's account

To ensure data consistency, the steps above have only two results: all steps either succeed or fail and then get rolled back. In this process, the distributed lock mechanism needs to be applied to the accounts of both user A and user B to avoid dirty data. In the WeChat red packet system, which is a huge distributed cluster, this issue may become extremely complicated.

Fortunately, the WeChat red packet system utilizes Tencent Cloud CMQ to avoid excessive overheads caused by distributed transactions. In the same scenario where user A sends a 10 CNY red packet to user B with CMQ used, things are much simpler as follows:

- In step 7 above, user B opens the red packet and finds 10 CNY in it. However, the final crediting may fail due to high concurrency pressure on that day.
- The red packet system forwards all the requests with crediting failures to CMQ. When the account balance fails to be updated for user B, the client on the mobile phone will display the waiting state. Then, the account system will continuously pull messages from CMQ to retry updating. CMQ ensures that the 10 CNY crediting message will never be lost until it is fetched out.
- On Chinese New Year's Eve, user actions of sending and opening red packets and crediting are all converted into billions of requests. If a traditional transaction method is used, the concurrency pressure will be aggravated and crash the system.
- CMQ ensures reliable storage and transfer of red packet messages and can write three copies in real time to avoid data loss. When fund crediting fails, the operation can be retried multiple times so as to avoid disadvantages of traditional methods such as rollback upon failure and frequent database polling.