



APUSIC
固若长城
睿比世界

Performance Tuning

金蝶Apusic分布式消息队列V2.0.6_for_rabbitmq

Copyright Notice

All intellectual property rights including software copyrights involved in this document have been legally registered and duly owned by Kingdee Apusic Cloud Computing Co., Ltd. Such rights are protected by Copyright Law of the People's Republic of China, Regulations on Protection of Computer Software, Regulations on Intellectual Property Protection, relevant international copyright conventions, laws, regulations and other intellectual property legislations. Unauthorized use is strictly prohibited.

Disclaimer

All copyright information contained herein is legally owned by Kingdee Apusic Cloud Computing Co., Ltd. and protected by applicable laws. The Company shall bear no liability for any information unrelated to Kingdee Apusic contained in this document. You may view, copy and print this document only within the scope permitted by Copyright Law of the People's Republic of China. No entity or individual may use, modify or republish any part or content of this document without written authorization from Kingdee Apusic Cloud Computing Co., Ltd. Any violation shall constitute infringement, and the Company reserves the right to pursue legal liabilities accordingly.

Updates to this document shall not be notified separately. You may consult the Company regarding issues related to this document. All rights not expressly granted by the Company are reserved.

Trademark Notice

 is a registered trademark filed with the National Intellectual Property Administration of PRC by Kingdee Apusic Cloud Computing Co., Ltd. Shenzhen. Exclusive rights to the registered trademark are legally owned by Kingdee Apusic and protected by law. Without written consent of Kingdee Apusic, no entity or individual shall use, copy, modify, disseminate, transcribe, or bundle and sell any part of this trademark with other products in any manner or for any reason. Kingdee Apusic shall pursue legal liabilities against any infringement of its trademark rights. All other trademarks or registered trademarks mentioned in this document belong to their respective owners.

Contents

- 1 Preface
 - 1.1 Target Audience
 - 1.2 Related Documents
 - 1.3 Technical Support
- 2 System-Level Optimization
 - 2.1 Operating System Optimization
 - 2.1.1 File Descriptors
 - 2.1.2 Kernel Parameters
 - 2.1.3 Disk Optimization
 - 2.2 Network Optimization
- 3 RabbitMQ Configuration Optimization
 - 3.1 Memory Management
 - 3.2 Disk Management
 - 3.3 Connection and Channel Optimization
 - 3.4 Queue Optimization
 - 3.4.1 Queue Type Selection
 - 3.4.2 Queue Parameter Optimization
 - 3.5 Message Persistence Optimization
 - 3.5.1 When to Use Persistence
 - 3.5.2 Batch Persistence
- 4 Client Optimization
 - 4.1 Connection Management
 - 4.1.1 Connection Pool
 - 4.1.2 Automatic Recovery
 - 4.2 Producer Optimization
 - 4.2.1 Batch Sending
 - 4.2.2 Async Sending
 - 4.3 Consumer Optimization
 - 4.3.1 QoS Prefetch
 - 4.3.2 Batch Acknowledgment
 - 4.3.3 Multiple Consumers
- 5 Cluster Optimization

- 5.1 Node Deployment Strategy
- 5.2 Mirrored Queue Optimization
- 5.3 Quorum Queue Optimization
- 5.4 Stream Queue Optimization
- 6 Monitoring and Tuning
 - 6.1 Performance Metrics
 - 6.2 Performance Testing
 - 6.3 Bottleneck Analysis
- 7 Common Issue Optimization
 - 7.1 Message Backlog Optimization
 - 7.2 Insufficient Memory Optimization
 - 7.3 High Latency Optimization
- 8 Performance Optimization Checklist
 - 8.1 System Level
 - 8.2 RabbitMQ Configuration
 - 8.3 Client Optimization
 - 8.4 Cluster Optimization

1 Preface

This document describes the best practices for performance optimization of Kingdee Apusic Distributed Message Queue for RabbitMQ (Apusic Distributed Message Queue, abbreviated as: ADMQ for RabbitMQ), helping users fully leverage system performance.

1.1 Target Audience

This document is applicable to IT information business leaders, R&D managers, software project managers, software architects, and operations engineers.

1.2 Related Documents

For more information about ADMQ for RabbitMQ products, please refer to the following ADMQ for RabbitMQ product documentation set:

No.	Document	Description
1	Kingdee Apusic Distributed Message Queue for RabbitMQ Quick Start Guide	Briefly introduces how to get started with ADMQ for RabbitMQ.
2	Kingdee Apusic Distributed Message Queue for RabbitMQ Installation Guide	Details how to install ADMQ for RabbitMQ on various operating systems, as well as service start/stop operations.
3	Kingdee Apusic Distributed Message Queue for RabbitMQ Message Engine User Guide	Details the usage, configuration, management, and supporting tools of the ADMQ for RabbitMQ message engine.
4	Kingdee Apusic Distributed Message Queue for RabbitMQ Management Console User Guide	Details the usage and operation instructions of the ADMQ for RabbitMQ management console.
5	Kingdee Apusic Distributed Message Queue for RabbitMQ Developer Guide	Details the development of ADMQ for RabbitMQ client applications using various programming languages.
6	Kingdee Apusic Distributed Message Queue for RabbitMQ Migration Guide	Details the migration from RabbitMQ to ADMQ for RabbitMQ.
7	Kingdee Apusic Distributed Message Queue for RabbitMQ Operations Manual	Details monitoring, operations, security hardening, and other operational aspects of ADMQ for RabbitMQ.

8	Kingdee Apusic Distributed Message Queue for RabbitMQ Performance Optimization Guide	Details performance tuning for ADMQ for RabbitMQ.
---	--	---

1.3 Technical Support

ADMQ for RabbitMQ provides comprehensive technical support services. You can obtain technical support through the following channels:

- Website: www.apusic.com
- Phone: 400-855-5800
- Email: support@apusic.com
- Kingdee Cloud Community: <https://vip.kingdee.com/?productId=73&productLineId=14&lang=zh-CN>

When contacting technical support, please provide the following information:

1. Your name
2. Company information and contact details
3. Operating system and its version
4. Product version number
5. Detailed information such as error logs and screenshots

2 System-Level Optimization

2.1 Operating System Optimization

2.1.1 File Descriptors

```
# View current limits
ulimit -n

# Modify limits (/etc/security/limits.conf)
* soft nofile 65535
* hard nofile 65535
```

2.1.2 Kernel Parameters

```
# /etc/sysctl.conf

# Increase port range
net.ipv4.ip_local_port_range = 1024 65535

# TCP connection optimization
net.ipv4.tcp_tw_reuse = 1
net.ipv4.tcp_fin_timeout = 30
net.core.somaxconn = 65535

# Memory optimization
vm.swappiness = 10
vm.dirty_ratio = 40
vm.dirty_background_ratio = 10

# Apply configuration
sysctl -p
```

2.1.3 Disk Optimization

Optimization Item	Recommendation
File System	Use XFS file system
Mount Options	noatime,nodiratime
Disk Scheduler	Use noop or deadline
RAID Configuration	Use RAID 10 or RAID 0
SSD Optimization	Enable TRIM, reserve over-provisioning

```
# View disk scheduler algorithm
cat /sys/block/sda/queue/scheduler

# Set to deadline
echo deadline > /sys/block/sda/queue/scheduler
```

2.2 Network Optimization

```
# Network buffers
net.core.rmem_max = 134217728
net.core.wmem_max = 134217728
net.ipv4.tcp_rmem = 4096 87380 134217728
net.ipv4.tcp_wmem = 4096 65536 134217728

# TCP optimization
net.ipv4.tcp_window_scaling = 1
net.ipv4.tcp_timestamps = 1
net.ipv4.tcp_sack = 1
net.ipv4.tcp_no_metrics_save = 1
```

3 RabbitMQ Configuration Optimization

3.1 Memory Management

Parameter	Default	Recommended	Description
vm_memory_high_watermark.relative	0.4	0.6-0.7	Memory usage upper limit ratio
vm_memory_high_watermark_paging_ratio	0.5	0.5	Memory ratio to start paging
vm_memory_high_watermark.absolute	-	4G	Absolute memory limit

```
# rabbitmq.conf
vm_memory_high_watermark.relative = 0.6
vm_memory_high_watermark_paging_ratio = 0.5
```

3.2 Disk Management

Parameter	Default	Recommended	Description
disk_free_limit.relative	2.0	1.5	Disk space lower limit ratio
disk_free_limit.absolute	50MB	1GB	Absolute disk space lower limit

```
# rabbitmq.conf
disk_free_limit.relative = 1.5
disk_free_limit.absolute = 1GB
```

3.3 Connection and Channel Optimization

Parameter	Default	Recommended	Description
channel_max	2047	Adjust as needed	Maximum channels per connection
connection_max	infinity	Adjust as needed	Maximum connections
heartbeat	60	30-60	Heartbeat interval (seconds)
handshake_timeout	10000	10000	Handshake timeout (milliseconds)

```
# rabbitmq.conf
channel_max = 2047
heartbeat = 30
handshake_timeout = 10000
```

3.4 Queue Optimization

3.4.1 Queue Type Selection

Queue Type	Throughput	Reliability	Applicable Scenario
Classic Queue	Medium	Medium (mirroring required)	General business scenarios
Quorum Queue	Medium	High	High availability scenarios
Stream Queue	Very High	High	High throughput, logging scenarios

3.4.2 Queue Parameter Optimization

```
Map<String, Object> args = new HashMap<>();

# Queue length limit, prevent infinite backlog
args.put("x-max-length", 100000);
args.put("x-max-length-bytes", 1073741824); # 1GB

# Message TTL
args.put("x-message-ttl", 3600000); # 1 hour

# Queue TTL
args.put("x-expires", 7200000); # Delete after 2 hours with no
consumers

# Lazy queue (reduce memory usage)
args.put("x-queue-mode", "lazy");

channel.queueDeclare("optimized.queue", true, false, false, args);
```

3.5 Message Persistence Optimization

3.5.1 When to Use Persistence

Scenario	Recommendation
Critical business messages	Must persist (delivery_mode=2)
Log messages	Can be non-persistent
High throughput scenarios	Trade off reliability and performance

3.5.2 Batch Persistence

Use Confirm mode + batch acknowledgment to improve persistence performance:

```
channel.confirmSelect();
channel.setConfirmCallback(...);

# Batch send
for (int i = 0; i < 1000; i++) {
    channel.basicPublish(EXCHANGE, ROUTING_KEY,
        MessageProperties.PERSISTENT_TEXT_PLAIN,
        message.getBytes());
}

# Wait for all confirmations
channel.waitForConfirmsOrDie(5000);
```

4 Client Optimization

4.1 Connection Management

4.1.1 Connection Pool

```
# Use connection pool to manage connections
ConnectionFactory factory = new ConnectionFactory();
factory.setHost("localhost");
factory.setConnectionTimeout(30000);
factory.setRequestedHeartbeat(30);
factory.setAutomaticRecoveryEnabled(true);

# Single connection, multiple channels (recommended)
ConnectionFactory factory = new ConnectionFactory();
Connection connection = factory.newConnection();
for (int i = 0; i < 10; i++) {
    Channel channel = connection.createChannel();
    # Use channel to process messages
}
```

4.1.2 Automatic Recovery

```
factory.setAutomaticRecoveryEnabled(true);
factory.setNetworkRecoveryInterval(5000);
factory.setTopologyRecoveryEnabled(true);
factory.setChannelRpcTimeout(10000);
```

4.2 Producer Optimization

4.2.1 Batch Sending

```
# Batch send messages
List<Message> messages = new ArrayList<>();
for (int i = 0; i < 1000; i++) {
```

```

        messages.add(new Message(...));
    }

    # Use transaction or Confirm mode to ensure reliability
    channel.confirmSelect();
    for (Message msg : messages) {
        channel.basicPublish(EXCHANGE, ROUTING_KEY, null,
            msg.toBytes());
    }
    channel.waitForConfirmsOrDie();

```

4.2.2 Async Sending

```

# Use thread pool for async sending
ExecutorService executor = Executors.newFixedThreadPool(10);
for (Message msg : messages) {
    executor.submit(() -> {
        channel.basicPublish(EXCHANGE, ROUTING_KEY, null,
            msg.toBytes());
    });
}

```

4.3 Consumer Optimization

4.3.1 QoS Prefetch

```

# Set QoS, control unacknowledged message count
channel.basicQos(100); # Maximum 100 unacknowledged messages per
consumer

# Global QoS
channel.basicQos(0, 100, true); # Maximum 100 across entire channel

```

4.3.2 Batch Acknowledgment

```
# Consume and batch acknowledge
List<Long> deliveryTags = new ArrayList<>();
for (int i = 0; i < 100; i++) {
    GetResponse response = channel.basicGet(Queue, false);
    if (response != null) {
        deliveryTags.add(response.getEnvelope().getDeliveryTag());
        processMessage(response);
    }
}

# Batch acknowledge
if (!deliveryTags.isEmpty()) {
    channel.basicAck(deliveryTags.get(deliveryTags.size() - 1),
true);
}
```

4.3.3 Multiple Consumers

```
# Create multiple consumers for parallel processing
for (int i = 0; i < 10; i++) {
    Channel channel = connection.createChannel();
    channel.basicQos(50);

    DeliverCallback callback = (tag, delivery) -> {
        processMessage(delivery);
        channel.basicAck(delivery.getEnvelope().getDeliveryTag(),
false);
    };

    channel.basicConsume(Queue, false, callback, tag -> {});
}
```

5 Cluster Optimization

5.1 Node Deployment Strategy

Deployment Strategy	Description	Applicable Scenario
Same Datacenter	All nodes in same datacenter	Low latency, high throughput
Cross-Datacenter	Nodes distributed across datacenters	High availability, disaster recovery
Heterogeneous	Mixed nodes with different configurations	Cost optimization

5.2 Mirrored Queue Optimization

```
# Optimize mirror sync strategy
bin/admq rabbitmq admin ctl set_policy ha-optimized "^important-.*"
'
{
    "ha-mode": "exactly",
    "ha-params": 2,
    "ha-sync-mode": "automatic",
    "ha-promote-on-shutdown": "when-synced"
}'
```

5.3 Quorum Queue Optimization

```
# Adjust Raft parameters
bin/admq rabbitmq admin ctl eval 'application:set_env(rabbit,
quorum_commands_soft_limit, 256).'
bin/admq rabbitmq admin ctl eval 'application:set_env(rabbit,
quorum_commands_hard_limit, 512).'
```

5.4 Stream Queue Optimization

```
Map<String, Object> args = new HashMap<>();  
# Segment file size  
args.put("x-stream-max-segment-size-bytes", 104857600); # 100MB  
  
# Retention policy  
args.put("x-stream-max-retention-size-bytes", 10737418240L); # 10GB  
args.put("x-stream-max-retention-time", 604800000); # 7 days  
  
channel.queueDeclare("stream.queue", true, false, false, args);
```

6 Monitoring and Tuning

6.1 Performance Metrics

Metric	Healthy Range	Optimization Suggestion
CPU Usage	< 70%	Scale up or optimize business logic
Memory Usage	< vm_memory_high_watermark	Increase memory or optimize queue configuration
Disk Usage	< 80%	Scale up or clean data
Message Publish Rate	Based on business needs	Optimize producers or scale up
Message Consume Rate	Close to publish rate	Optimize consumers or add more consumers
Unacknowledged Messages	< QoS prefetch count	Optimize consumer processing speed
Connection Count	< connection_max	Optimize connection usage or use connection pool

6.2 Performance Testing

```
# Test using perf-test tool
rabbitmq-perf-test -x 10 -y 10 -u "test-queue" -a --auto-delete false

# Parameter description
# -x: Number of producers
# -y: Number of consumers
# -u: Queue name
# -a: Auto acknowledgment
# --auto-delete: Whether to auto-delete queue
```

6.3 Bottleneck Analysis

1. **CPU Bottleneck:** Complex message processing logic, encryption/decryption, large number of connections
2. **Memory Bottleneck:** Queue backlog, large number of connections, lazy queues not enabled
3. **Disk Bottleneck:** Large volume of persistent messages, insufficient disk I/O
4. **Network Bottleneck:** Cross-datacenter deployment, large data transfer

7 Common Issue Optimization

7.1 Message Backlog Optimization

1. Increase consumer count
2. Optimize consumer processing logic
3. Set queue length limits
4. Use dead letter queue to handle expired messages

7.2 Insufficient Memory Optimization

1. Increase physical memory
2. Adjust `vm_memory_high_watermark`
3. Enable lazy queues
4. Clean up unnecessary queues and messages
5. Reduce connection count

7.3 High Latency Optimization

1. Use auto acknowledgment (non-critical messages)
2. Reduce persistence frequency
3. Optimize network configuration
4. Use closer nodes
5. Reduce message size

8 Performance Optimization Checklist

8.1 System Level

- ☐ File descriptor limits adjusted
- ☐ Kernel parameters optimized
- ☐ Disk using XFS file system
- ☐ Network parameters optimized
- ☐ System resources sufficient (CPU, memory, disk)

8.2 RabbitMQ Configuration

- ☐ Memory watermark set appropriately
- ☐ Disk space limit set appropriately
- ☐ Heartbeat interval appropriate
- ☐ Connection and channel limits appropriate
- ☐ Queue leader allocation strategy appropriate

8.3 Client Optimization

- ☐ Use connection pool
- ☐ Use single connection with multiple channels
- ☐ Enable automatic recovery
- ☐ Set QoS appropriately
- ☐ Use batch acknowledgment
- ☐ Choose appropriate acknowledgment mode

8.4 Cluster Optimization

- ☐ Nodes select appropriate queue types
- ☐ Mirrored queue policy reasonable
- ☐ Quorum Queue parameters optimized
- ☐ Stream Queue retention policy reasonable
- ☐ Cluster scale meets business requirements

UNIFIED SERVICE HOTLINE

4008-555-800

Kingdee Apusic Cloud Computing Co., Ltd. (referred to as “Kingdee Apusic”) was founded in 2000. Its predecessor was “Kingdee Middleware Company.” It is a basic software platform provider under Kingdee Group, a national standards-setting enterprise for cloud computing, and a core software enterprise in the national IT application innovation industry. Kingdee Apusic is an enterprise undertaking the National 863 Key R&D Program and the National Major Science and Technology Project for Core Electronic Components, High-end General-purpose Chips and Fundamental Software, and is also a basic platform provider for the national key projects of “Two Networks, One Website, Four Databases, and Twelve Golden Projects.” Its products are widely used in key industries such as government, military, finance, and energy, and it has cumulatively served more than 100,000 customers.



National standard setter for cloud computing
Basic software enterprise under Kingdee Group
Core enterprise for IT application innovation
Website: www.apusic.com

