



APUSIC
固若长城
睿比世界

Performance Tuning Guide

Apusic In-Memory Data Cache

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 Documentation
 - 1.3 Technical Support
- 2 Sources of Performance Bottlenecks
 - 2.1 Memory Management Configuration
 - 2.1.1 Memory Management Optimization Points
 - 2.1.2 Optimization Methods
 - 2.1.2.1 Reserving Space
 - 2.1.2.2 Adjusting Eviction Policies
 - 2.1.3 Verification Methods
 - 2.2 Persistence Configuration
 - 2.2.1 Persistence Optimization Points
 - 2.2.1.1 Optimization Methods
 - 2.2.1.2 Scenario-Based Strategies
 - 2.3 Network Configuration
 - 2.3.1 Optimization Points
 - 2.4 System Configuration

1 Preface

This document is the performance tuning guide for Apusic In-Memory Data Cache (AMDC) V2.0.6, aiming to provide AMDC V2.0.6 users with comprehensive and systematic performance tuning guidance. By deeply analyzing performance bottlenecks in AMDC's own configuration and combining optimization cases from typical business scenarios, it helps users resolve performance issues that may be encountered in high-concurrency environments.

The document covers everything from basic system parameter tuning to complex optimization for millions of concurrent connections, from common timeout configurations to critical long-connection reuse strategies, as well as troubleshooting methods for various issues that may be encountered during performance testing.

With the guidance in this document, users can:

- Identify and resolve AMDC performance bottlenecks in high-concurrency scenarios
- Properly configure operating system and AMDC parameters to improve overall system performance
- Quickly locate and resolve common issues encountered in performance testing
- Implement corresponding optimization solutions based on specific business scenarios

1.1 Target Audience

This document is intended for AMDC product operations engineers, IT system operations engineers, development engineers, software architects, R&D managers, and other personnel.

1.2 Related Documentation

For more information about AMDC V2.0.6, please refer to the following AMDC V2.0.6 product manual documentation set:

No.	Document	Description
1	Apusic In-Memory Data Cache V2.0.6 Quick Start Guide	A brief introduction on how to quickly get started with AMDC.
2	Apusic In-Memory Data Cache V2.0.6 Installation Guide	Detailed introduction on how to install AMDC on various operating systems, AMDC service start/stop operations, and the product registration process.
3	Apusic In-Memory Data Cache V2.0.6 Cache Core User Manual	Detailed introduction on how to use, configure, and manage AMDC features and supporting tools.

4	Apusic In-Memory Data Cache V2.0.6 Console User Manual	Detailed instructions on using and operating AMDC console features.
5	Apusic In-Memory Data Cache V2.0.6 Development Guide	Detailed instructions for developing AMDC client applications using various programming languages.
6	Apusic In-Memory Data Cache V2.0.6 Migration Guide	Detailed instructions for migrating AMDC from historical versions to V2.0.6, as well as migrating from Redis to AMDC.
7	Apusic In-Memory Data Cache V2.0.6 Operations Guide	Detailed instructions for AMDC monitoring, operations, security hardening, and other operational procedures.
8	Apusic In-Memory Data Cache V2.0.6 Performance Tuning Guide	Detailed instructions for AMDC performance tuning.

1.3 Technical Support

AMDC products provide 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 requesting technical support, please provide the following information:

1. Your name
2. Company information and contact details
3. Operating system and version
4. Product version number
5. Detailed information including logs, screenshots, etc. of any exceptions or errors

2 Sources of Performance Bottlenecks

As a high-performance in-memory database, AMDC's performance is constrained by multiple dimensions of factors. Bottleneck sources and optimization paths need to be examined from a systemic perspective. Performance bottlenecks mainly fall into four dimensions:

- The memory dimension is the core constraint, manifested as insufficient physical memory triggering eviction mechanisms, big Key operations causing blocking, memory fragmentation reducing utilization, and copy-on-write overhead during fork operations. These factors together form the performance ceiling of memory-intensive applications.
- The network dimension is particularly critical in high-concurrency scenarios: the number of connections is limited by kernel parameters and file descriptors, improper buffer settings affect throughput, and network latency and bandwidth bottlenecks directly constrain cross-node communication efficiency.
- The CPU dimension is constrained by the single-threaded architecture; compute-intensive tasks such as complex command execution, persistence fork operations, master-slave synchronization, and key expiration cleanup can all cause main thread blocking, leading to latency fluctuations.

Next, we will introduce AMDC tuning approaches in terms of memory management, persistence configuration, network configuration, and system configuration.

2.1 Memory Management Configuration

2.1.1 Memory Management Optimization Points

- All AMDC data resides in memory. If available RAM is exhausted, OOM will be triggered (Linux kills the process directly) or severe swapping will occur, causing QPS to drop sharply.
- Hot data often accounts for only 20% of data but serves 80% of requests. Allocating limited memory to hot keys maximizes the cache hit ratio and reduces average latency.
- The operating system itself, background backups, AOF rewrite, and network buffers also require 30%-40% free memory; otherwise, the instantaneous double memory overhead during fork will cause failures.

2.1.2 Optimization Methods

2.1.2.1 Reserving Space

Set it appropriately based on the total server memory. It is recommended to reserve 30%-40% of memory for the operating system and other processes to avoid memory overflow. For example, if the server has 16GB of memory, set `maxmemory` to 10GB with the configuration statement `maxmemory 10gb`.

2.1.2.2 Adjusting Eviction Policies

Select an appropriate memory eviction policy based on business data characteristics. When hot data is concentrated in the business and needs to be retained preferentially, `volatile-lru` (evicts the least recently used keys among those with an expiration set) or `allkeys-lru` (evicts the least recently used keys among all keys) is recommended.

If the business is sensitive to data expiration time, you can choose `volatile-ttl` (evicts keys with the shortest remaining time among those with an expiration set). Configuration example: `maxmemory-policy volatile-lru`.

2.1.3 Verification Methods

```
# Memory usage
amdc-cli INFO memory | grep used_memory_human
# Observe ≤ 10 GB; also the difference between used_memory_rss_human
and used_memory_human should be < 200 MB (low fragmentation).

# Eviction count
amdc-cli INFO stats | grep evicted_keys
# If it keeps growing while hits decline, 10 GB is too small;
increase it or re-evaluate the hot key ratio.

# Hit ratio
amdc-cli INFO stats | grep keyspace_hits
# keyspace_hits / (hits + misses) ≥ 95 % is healthy.

# Latency spikes
amdc-cli --latency -p 6359
# p99 ≤ 1 ms (on a local machine with 10-Gigabit NIC and SSD).
```

2.2 Persistence Configuration

2.2.1 Persistence Optimization Points

Main issues with RDB persistence:

- Data loss window: a crash between two snapshots loses all writes in that period

- Disk I/O peaks: large amounts of disk writes may occur during saving

Main issues with AOF persistence:

- AOF rewrite bloat: double-write pressure during BGREWRITEAOF (old AOF appending + new AOF rewriting)
- fsync performance loss: appendfsync always reduces performance by about 90%

2.2.1.1 Optimization Methods

Regarding RDB smart save triggering (to avoid frequent bgsave), we need to choose appropriate save trigger conditions

```
save 900 1      # At least 1 key change in 15 minutes
save 300 10     # At least 10 key changes in 5 minutes
save 60 10000   # At least 10000 key changes in 1 minute
```

Conversely, for AOF saving, we need to pay more attention to the pressure during rewriting and the frequency of disk flushing.

```
# 1. A reasonable fsync policy (choose everysec for 95% of
scenarios)
appendfsync everysec  # Balances performance and safety
# appendfsync no      # Only use when minute-level data loss is
tolerable

# 2. Automatic rewrite control
auto-aof-rewrite-percentage 100  # Triggers rewrite when growth
reaches 100%
auto-aof-rewrite-min-size 64mb   # Minimum AOF file size threshold
for rewriting

# 3. Optimize the rewrite process
no-appendfsync-on-rewrite yes    # No fsync during rewrite to avoid
double blocking
aof-rewrite-incremental-fsync yes # Incremental sync to reduce data
volume per fsync
```

Or adopt a more neutral approach: hybrid persistence (RDB + AOF). This mode can achieve data safety close to the AOF level while occupying less disk space than the AOF mode, balancing the pros and cons of both AOF and RDB.

```
aof-use-rdb-preamble yes # Enable hybrid mode
# Used with the following parameters:
aof-rewrite-incremental-fsync yes
rdbcompression yes # The RDB part is still compressed
```

2.2.1.2 Scenario-Based Strategies

Cache-type applications (data can be rebuilt) Recommended strategy: RDB only Configuration example:

```
save 3600 1 # Back up once per hour
stop-writes-on-bgsave-error no # Allow writes to continue
```

However, this is based on fast recovery speed (millisecond-level), minimized disk usage, and acceptable data loss of up to 1 hour. At the same time, we also need to cooperate with multi-level caching (local cache + AMDC) and set expiration times for automatic eviction.

Financial transactions (zero data loss requirement) AOF (always) + RDB cold backup Configuration example:

```
appendfsync always
auto-aof-rewrite-percentage 50 # More frequent rewrites to reduce
recovery time
aof-rewrite-min-size 32mb
```

At this point, we can consider using some performance compensation measures to make up for the performance degradation brought by data safety, such as using better-performance disks, using a master-slave architecture, and completing persistence operations on slave nodes.

2.3 Network Configuration

2.3.1 Optimization Points

To achieve long-term stable operation, our network configuration optimization covers not only performance bottleneck handling, but also configuration optimization in high availability, fault recovery, and other aspects.

Network stability is inseparable from kernel parameter optimization and normal communication between nodes.

The following are several basic and critical network configurations, which need to be adjusted in combination with kernel parameters, and the communication configuration between master-slave or cluster nodes should be adjusted according to business needs.

```
maxclients 10000                # Maximum number of client
connections (subject to system limits)
tcp-backlog 511                # Half-open connection queue
length, must be ≤ somaxconn
timeout 300                    # Idle connection timeout
(seconds), 0 means disabled
repl-timeout 60                # Master-slave replication
timeout
cluster-node-timeout 15000      # Cluster node timeout
(milliseconds)
```

In high availability scenarios, we need to adjust some parameter configurations for interaction between master and slave nodes to ensure the stability of the entire service system. Most of the time, we need to adjust repl-backlog-size according to different business and memory needs: too large will occupy memory; too small may cause incomplete incremental synchronization and thus lead to full synchronization, which may result in high CPU and memory usage.

```
# Master node configuration
repl-backlog-size 512mb        # Replication backlog buffer
(recommended: 10% of memory)
repl-backlog-ttl 3600          # Buffer retention time (1h)
repl-disable-tcp-nodelay no    # Enable Nagle algorithm to
reduce packet count

# Slave node configuration
repl-ping-replica-period 10    # Frequency of slave pinging
master
repl-timeout 60                # Replication timeout (must be >
ping period × 3)
```

At the same time, attention should be paid to the Sentinel configuration Sentinel count formula (to ensure quorum):
Sentinel count = 2N + 1 (N is the number of tolerated faulty Sentinels) Minimum recommended: 3 Sentinels,

tolerating 1 failure

In addition, Sentinel deployment is different from services: we need to deploy it independently and not on the same machine as AMDC instances, otherwise it may cause a single point of failure. The following are recommended Sentinel configuration parameters:

```
# 1. Monitoring configuration (required for each Sentinel)
sentinel monitor mymaster 10.0.1.100 6379 2
# Format: sentinel monitor <master-name> <IP> <port> <quorum>

# 2. Subjective down judgment
sentinel down-after-milliseconds mymaster 30000
# Value description:
# - <10000ms (low-latency intranet environment)
# - 30000ms (standard production environment)
# - >60000ms (cross-data-center or high-latency networks)

# 3. Failover timeout control
sentinel failover-timeout mymaster 180000
# Purpose: controls the timeout for each stage of failover

# 4. Parallel synchronization limit
sentinel parallel-syncs mymaster 1
# Value description:
# - 1: safest, low pressure on the new master
# - 2-3: balances synchronization speed and master load
# - >3: may overload the new master

# 5. Node communication optimization
sentinel resolve-hostnames no          # Disable DNS resolution, use IP
sentinel announce-ip 10.0.1.200       # Sentinel announcement IP
sentinel announce-port 26379          # Sentinel announcement port
```

When using a cluster, pay attention to the timeout settings between nodes. Generally, the default values are fine. It should not be too small, which would cause frequent cluster checks, nor too large, which would cause node downtime to go undetected. cluster-announce-ip explicitly specifies the IP that a node serves externally in cluster deployment

```
# Cluster node configuration
cluster-enabled yes
cluster-config-file nodes.conf
cluster-require-full-coverage no      # Avoid the entire cluster being
unavailable due to a few unavailable slots

# Node timeout and heartbeat
cluster-node-timeout 15000            # Node timeout (affects
failover)
cluster-replica-validity-factor 10    # Slave node data validity
factor
cluster-announce-port 6379
cluster-announce-bus-port 16379      # Cluster bus port (port +
10000)

# Cross-data-center deployment optimization
cluster-announce-ip 192.168.1.100    # Explicitly specify the node IP
```

Typical symptoms and solutions:

```
# Symptom
[ERR] Node ... is not empty. Either the node already knows other
nodes

# Solution steps:
1. Check that cluster-announce-ip is correct on each node
2. Ensure all ports are open in the firewall (6379 and 16379) # This
issue is often overlooked in most default deployments; pay
attention.
3. Clear old cluster configuration: rm nodes.conf, then recreate the
cluster
```

2.4 System Configuration

For a stable and efficient cache system, sometimes we need to extend the tuning scope to the system level. The following parameter adjustments are for reference.

1. vm.overcommit_memory - Memory Overcommit Policy

```
# Check current value
cat /proc/sys/vm/overcommit_memory
# Permanent modification (/etc/sysctl.conf)
vm.overcommit_memory = 1
```

Generally, the default value of this parameter is 0, which means the kernel checks available memory and may refuse allocation if insufficient (which may cause the AMDC service fork to fail) Other values: 1 -- Always allow allocation (even exceeding physical memory) 2 -- Strict limit; allocation cannot exceed physical memory + swap In this regard, our recommendation is to set it to 1. Otherwise, when AMDC executes bgsave, the fork child process may fail due to "insufficient memory" (double memory is not actually needed, but the kernel will check)

2. Transparent HugePages

```
# Check status
cat /sys/kernel/mm/transparent_hugepage/enabled

# Disable command
echo never > /sys/kernel/mm/transparent_hugepage/enabled
echo never > /sys/kernel/mm/transparent_hugepage/defrag

# Permanently disable (CentOS/RHEL 7+)
grubby --update-kernel=ALL --args="transparent_hugepage=never"
```

It is recommended to disable it. Otherwise, fork latency during bgsave may increase several times, memory usage may increase by 30%-40%, and unpredictable latency spikes may occur during operation

3. net.core.somaxconn - Connection Queue Size

```
# Check current value
cat /proc/sys/net/core/somaxconn
```

```
# Permanent modification
net.core.somaxconn = 65535
```

For this parameter, our recommendation is to increase it, because AMDC's tcp-backlog configuration (default 511) is limited by this parameter. In high concurrency, if the queue is too small, clients will receive "Connection refused". Typical scenario: during flash sale events, a large number of connections arrive instantly.

4. ulimit File Descriptor Limit

```
# Check AMDC process limits
cat /proc/$(pidof amdc-server)/limits | grep "open files"

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

The default value of this parameter is usually 1024. Its function is to limit the number of files a single process can open (each connection occupies one file descriptor). We recommend increasing it because: each client connection consumes 1 file descriptor, and persistence also opens files (RDB/AOF). The default 1024 can only support about 1000 concurrent connections.

Production recommendation: at least 65535; large clusters may need more. Note that most clients now come with connection pools, so actual consumption needs to be analyzed on a case-by-case basis.

5. Network Buffer Adjustment

```
# Permanent modification (/etc/sysctl.conf)
net.core.rmem_max = 134217728
net.core.wmem_max = 134217728
net.ipv4.tcp_rmem = 4096 87380 134217728
net.ipv4.tcp_wmem = 4096 65536 134217728
```

The above parameters are mainly used to control the size of TCP send and receive buffers. Because the default values are generally small, our recommendation is to increase them, because our service may need to return large amounts of data, either big values or batch operations. If the buffer is too small, the TCP window will shrink and throughput will decrease, which is especially important in high-bandwidth, high-latency networks.

Production recommendations:

- `rmem_max/wmem_max`: 128MB
- `tcp_rmem/tcp_wmem`: minimum 4KB , default 64KB/85KB , maximum 128MB

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

