



APUSIC
固若长城
睿比世界

Performance Optimization Guide

Apusic Load Balancer

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 Introduction
 - 1.1 Intended Audience
 - 1.2 Related Documents
 - 1.3 Technical Support
- 2 Performance Bottleneck Source Analysis (Detailed Explanation from Two Dimensions: Operating System + ALB)
 - 2.1 Operating System Level Bottlenecks
 - 2.1.1 File Descriptor (FD) Limit
 - 2.1.1.1 Why Does It Exist?
 - 2.1.1.2 How to Modify?
 - 2.1.1.3 How to Verify?
 - 2.1.2 TCP Connection Queue and Network Stack Parameters
 - 2.1.2.1 Why Does It Exist?
 - 2.1.2.2 How to Modify?
 - 2.1.2.3 How to Verify?
 - 2.1.3 TIME_WAIT Connection Accumulation
 - 2.1.3.1 Why Does It Exist?
 - 2.1.3.2 How to Mitigate?
 - 2.1.3.3 How to Verify?
 - 2.1.4 CPU and Interrupt Affinity
 - 2.1.4.1 Why Does It Exist?
 - 2.1.4.2 How to Optimize?
 - 2.1.4.3 How to Verify?
 - 2.2 ALB Configuration Bottlenecks
 - 2.2.1 Improper Worker Process and Connection Count Configuration
 - 2.2.1.1 Why Does It Exist?
 - 2.2.1.2 How to Optimize?
 - 2.2.1.3 How to Verify?
 - 2.2.2 Unreasonable Buffer Configuration
 - 2.2.2.1 Why Does It Exist?
 - 2.2.2.2 How to Optimize?
 - 2.2.2.3 How to Verify?

- 2.2.3 Frequent Log Writing Causes I/O Bottleneck
 - 2.2.3.1 Why Does It Exist?
 - 2.2.3.2 How to Verify?
- 2.2.4 High SSL/TLS Handshake Overhead
 - 2.2.4.1 Why Does It Exist?
 - 2.2.4.2 How to Optimize?
 - 2.2.4.3 How to Verify?
- 2.2.5 No Efficient Transmission Mechanism Enabled for Static Resources
 - 2.2.5.1 Why Does It Exist?
 - 2.2.5.2 How to Optimize?
 - 2.2.5.3 How to Verify?
- 2.3 Comprehensive Verification Methods
- 2.4 Summary
- 3 Timeout Control Optimization
 - 3.1 Performance Bottlenecks Caused by Improper Timeout Settings
 - 3.2 Client-side Timeouts
 - 3.2.1  Optimization Suggestions:
 - 3.2.2  Verification Method:
 - 3.3 Upstream Service Timeouts
 - 3.3.1  Optimization Suggestions (Taking an API Gateway as an Example):
 - 3.3.2  Verification Method:
- 4 Keep-Alive Configuration Guide
 - 4.1 Client ↔ ALB: Enable HTTP Keep-Alive
 - 4.2 ALB ↔ Upstream: Enable Upstream Keep-Alive (Key!)
 - 4.3 Backend Service: Needs to Support HTTP/1.1 and Maintain Connections
 - 4.4 Comprehensive Verification: How to Confirm That Keep-Alive and Timeout Configurations Are in Effect?
- 5 Typical Scenario Optimization Cases
 - 5.1 Scenario 1: Web Site Access Acceleration Optimization
 - 5.1.1 Business Scenario Description
 - 5.1.2 Optimization Goal
 - 5.1.3 Optimization Solution
 - 5.1.4 Optimization Effect
 - 5.2 Scenario 2: API Gateway Performance Optimization

- 5.2.1 Business Scenario Description
- 5.2.2 Optimization Goal
- 5.2.3 Optimization Solution
- 5.3 Scenario 3: File Download Service Optimization
 - 5.3.1 Business Scenario Description
 - 5.3.2 Optimization Goal
 - 5.3.3 Optimization Solution
 - 5.3.4 Optimization Effect
- 5.4 Scenario 4: ALB Reverse Proxying AAS Service Performance Optimization
 - 5.4.1 Operating System Level Tuning (Linux)
 - 5.4.2 ALB Configuration Optimization
 - 5.4.3 Stress Test Verification
- 5.5 Scenario 5: ALB Million-Level Concurrent Connection Test (C1000K)
 - 5.5.1 Background
 - 5.5.1.1  Typical Business Scenarios:
 - 5.5.1.2  Important Prerequisite:
 - 5.5.1.3  Core Difficulty:
 - 5.5.2 Solution
 - 5.5.3 Comprehensive Operating System Tuning (Linux)
 - 5.5.3.1 File Descriptor (FD) — The Most Critical!
 - 5.5.3.2 Network Stack Tuning
 - 5.5.3.3 Kernel and Hardware Recommendations
 - 5.5.4 ALB Configuration Optimization
 - 5.5.5 Backend Optimization
 - 5.5.6 Common Problems and Avoidance
 - 5.5.7 Summary
- 6 Performance Testing and Troubleshooting
 - 6.1 Summary of Diagnostic Commands
 - 6.2 Error Symptom: A large number of requests return 502 Bad Gateway during stress testing
 - 6.2.1  Error Analysis
 - 6.2.2  Solution
 - 6.2.3  Verification Method

- 6.3 Error Symptom: Stress testing is normal at the beginning, then QPS drops sharply and connections accumulate
 - 6.3.1  Error Analysis
 - 6.3.2  Solution
 - 6.3.3  Verification Method
- 6.4 Error Symptom: A large number of 504 Gateway Timeout occur during stress testing
 - 6.4.1  Error Analysis
 - 6.4.2  Solution
 - 6.4.3  Verification Method
- 6.5 Error Symptom: The client reports "Connection reset by peer" or "Connection closed" during stress testing
 - 6.5.1  Error Analysis
 - 6.5.2  Solution
 - 6.5.3  Verification Method
- 6.6 Error Symptom: After enabling upstream keepalive, QPS does not improve or even decreases
 - 6.6.1  Error Analysis
 - 6.6.2  Solution
 - 6.6.3  Verification Method
- 6.7 Error Symptom: The system CPU sys% (kernel mode) is too high during stress testing, reaching 70%+
 - 6.7.1  Error Analysis
 - 6.7.2  Solution
 - 6.7.3  Verification Method
- 6.8 Error Symptom: Under multi-core CPU, only CPU0 is 100% loaded, and other cores are idle
 - 6.8.1  Error Analysis
 - 6.8.2  Solution
 - 6.8.3  Verification Method
- 7 Performance Testing Error Quick Reference Table
- 8 Performance Optimization Key Parameter List
 - 8.1 Global and Process Control
 - 8.2 Event Model (Events)
 - 8.3 HTTP Global Configuration

- 8.4 Client Connection and Timeout
- 8.5 Reverse Proxy (Proxy)
- 8.6 Upstream Keep-Alive Pool
- 8.7 Logs

1 Introduction

This document is intended to provide comprehensive and systematic performance optimization guidance for users of Apusic Load Balancer V2.0.6 Standard Edition. By deeply analyzing performance bottlenecks at both the operating system level and the ALB configuration level, combined with optimization cases for typical business scenarios, it helps users solve the performance issues that may be encountered in high-concurrency environments.

The document covers a wide range of topics, from basic system parameter tuning to complex million-level concurrent connection optimization, from common timeout configurations to key keep-alive reuse strategies, as well as troubleshooting methods for various problems that may be encountered during performance testing. Through the guidance in this document, users can:

- Identify and resolve performance bottlenecks of ALB in high-concurrency scenarios
- Reasonably configure the operating system and ALB parameters to improve overall system performance
- Master keep-alive optimization strategies to effectively reduce system resource consumption
- Quickly locate and resolve common problems encountered during performance testing
- Implement corresponding optimization solutions according to specific business scenarios

1.1 Intended Audience

This document is mainly intended for developers, operations personnel, and related management staff of Apusic Load Balancer, and applies to Linux system environments (including Kylin, UOS, OpenEuler, CentOS, Ubuntu, and other distributions). By systematically studying and applying the optimization recommendations in this document, users can significantly improve the performance of ALB, ensuring that the system maintains stable and efficient service capabilities under high-load conditions.

1.2 Related Documents

To learn more about ALB V2.0.6 product information, please refer to the following ALB V2.0.6 product manual document set:

No.	Manual Document	Description
1	ALB V2.0.6 Release Notes	Introduces the ALB product version update content.
2	ALB V2.0.6 Quick Start Guide	Briefly introduces how to quickly get started with ALB.
3	ALB V2.0.6 Installation Guide	Describes in detail how to install ALB on various operating systems, ALB service start/stop operations, and the product registration process.

4	ALB V2.0.6 Proxy Function Manual	Describes in detail the use, configuration, and management of ALB-related functions, as well as the use of supporting tools.
5	ALB V2.0.6 Console User Manual	Describes in detail the use and operation instructions of ALB console-related functions.
6	ALB V2.0.6 Developer Manual	Describes in detail ALB external APIs, the alb-cli command-line tool, and AI Agent integration methods.
7	ALB V2.0.6 Migration Guide	Describes in detail the instructions for migrating and upgrading from historical ALB versions to V2.0.6.
8	ALB V2.0.6 Operations Manual	Describes in detail ALB monitoring, operations, security hardening, and other operations instructions.
9	ALB V2.0.6 Performance Optimization Guide	Describes in detail ALB performance tuning instructions.

1.3 Technical Support

ALB products provide comprehensive technical support services. You can obtain technical support through the following methods:

- 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 obtaining 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 logs and screenshots of the exceptions and errors that occurred

2 Performance Bottleneck Source Analysis (Detailed Explanation from Two Dimensions: Operating System + ALB)

Although ALB is known for its high performance, in high-concurrency, high-traffic, or complex business scenarios, its actual performance is often limited by two key aspects: operating system resource scheduling policies and the rationality of ALB's own configuration. These two influence each other and need to be tuned together.

Overall principles:

The operating system provides the "stage" (CPU, memory, network, file system, and other resources);

ALB is the "actor", and its performance depends on the stage conditions and its own choreography (configuration).

Ignoring either aspect may cause the performance to fall short of expectations.

2.1 Operating System Level Bottlenecks

2.1.1 File Descriptor (FD) Limit

2.1.1.1 Why Does It Exist?

Linux limits the number of file descriptors that each process can open by default (usually 1024) to prevent resource exhaustion. Each TCP connection and static file handle occupies one FD.

When the number of concurrent connections exceeds the limit, ALB will reject new connections and record `Too many open files` in the `error log`.

2.1.1.2 How to Modify?

```
# Edit the alb startup script and add the following two lines at the
beginning of the script
# [ALB installation directory]/bin/start-alb.sh or start-alb-and-
console.sh
ulimit -n 65536
ulimit -Hn 65536

# If using systemd, add LimitNOFILE=65536 at the end of [Service]
# Edit /etc/systemd/system/alb.service
[Service]
```

```
....
User=root
Type=forking
LimitNOFILE=65536 # Add this line
# After modification, use systemctl daemon-reload and then restart
ALB
```

2.1.1.3 How to Verify?

```
# View worker process FD limit
cat /proc/$(pgrep alb | tail -1)/limits | grep "open files"

# Real-time FD usage
lsof -p $(pgrep alb | tail -1) | wc -l
```

2.1.2 TCP Connection Queue and Network Stack Parameters

2.1.2.1 Why Does It Exist?

The kernel limits the number of incomplete connections through `somaxconn` (accept queue) and `tcp_max_syn_backlog` (SYN queue). The default value (such as 128) is very easy to become a bottleneck in high-concurrency scenarios, causing new connections to be dropped, which manifests as client connection timeouts.

2.1.2.2 How to Modify?

```
# Edit /etc/sysctl.conf
net.core.somaxconn = 65535
net.ipv4.tcp_max_syn_backlog = 65535
net.ipv4.tcp_abort_on_overflow = 1 # Optional: RST immediately when
the queue is full, making it easier to identify issues
```

Run `sysctl -p` to apply.

Meanwhile, in the ALB configuration file `alb.conf`, add the `backlog` parameter to the listening port configuration within the `server` block:

```
listen 80 backlog=65535;
```

Note: The `backlog` parameter of `listen` and the system `somaxconn` together determine the accept queue length. The actual effective value is `min(somaxconn, backlog)`. It is recommended to keep them consistent.

2.1.2.3 How to Verify?

```
ss -lnt # View Recv-Q (current accept queue length). If it
continues to be close to the backlog value, the queue needs to be
expanded
```

2.1.3 TIME_WAIT Connection Accumulation

2.1.3.1 Why Does It Exist?

The party that actively closes the connection enters the TIME_WAIT state (about 60 seconds) to ensure that residual data packets in the network will not interfere with new connections. In short-connection high-frequency request scenarios (such as mobile API), a large number of TIME_WAIT will exhaust the local port (ephemeral port range), resulting in `Cannot assign requested address`.

2.1.3.2 How to Mitigate?

```
# Edit /etc/sysctl.conf
net.ipv4.tcp_tw_reuse = 1
net.ipv4.ip_local_port_range = 1024 65535
net.ipv4.tcp_fin_timeout = 15
```

After modification, use `sysctl -p` to apply. ⚠ Disable `tcp_tw_recycle` (deprecated and unsafe under NAT).

2.1.3.3 How to Verify?

```
# Count the number of TIME-WAIT connections
ss -ant | awk '/TIME-WAIT/ {count++} END {print count+0}'

# View total connections and the number of timewait
ss -s
```

2.1.4 CPU and Interrupt Affinity

2.1.4.1 Why Does It Exist?

By default, the network card interrupt is concentrated on CPU0, causing a single core to be overloaded while other cores are idle. The alb multi-worker cannot make balanced use of multiple cores.

2.1.4.2 How to Optimize?

Enable `reuseport` (recommended):

```
listen 80 reuseport;
```

The kernel automatically distributes new connections to each worker's independent socket, achieving load balancing. Or manually bind IRQ to different CPUs (advanced).

2.1.4.3 How to Verify?

```
top # Observe whether the usage rate of each CPU core is balanced
```

2.2 ALB Configuration Bottlenecks

2.2.1 Improper Worker Process and Connection Count Configuration

2.2.1.1 Why Does It Exist?

- `worker_processes` is set too few (such as 1), unable to utilize multiple cores;
- `worker_connections` is set too low, and a single worker cannot handle sufficient concurrency;
- `worker_rlimit_nofile` is not set, resulting in ALB still being subject to internal limits even if the system allows it.

2.2.1.2 How to Optimize?

```
worker_processes auto;           # Automatically match the number
of CPU cores
worker_rlimit_nofile 131072;     # Increase the internal FD limit,
must be ≥ worker_connections × 2

events {
    use epoll;                   # Linux must enable this
    worker_connections 65536;    # Maximum connections per worker
    multi_accept on;             # Accept multiple connections at
```

```
once
}
```

2.2.1.3 How to Verify?

- Theoretical maximum concurrency $\approx$ worker_processes $\times$ worker_connections / 2 (each reverse proxy connection requires two sockets: client and upstream)
- Combine with stress testing to observe whether the expected QPS is achieved

2.2.2 Unreasonable Buffer Configuration

2.2.2.1 Why Does It Exist?

- proxy_buffering off will cause backend responses to be forwarded directly in a streaming manner, increasing the coupling between client and backend and reducing throughput;
- Buffers that are too small (such as the default 4k/8k) cause frequent syscalls in large response body scenarios, increasing CPU overhead;
- Buffers that are too large will waste memory.

2.2.2.2 How to Optimize?

```
proxy_buffering on;
proxy_buffer_size 16k;
proxy_buffers 8 32k;      # Total buffer  $\approx$  256k, adjust according
to business
proxy_busy_buffers_size 64k;
```

2.2.2.3 How to Verify?

- Compare the QPS and latency with buffering enabled/disabled;
- Use strace -p to observe the read/write system call frequency.

2.2.3 Frequent Log Writing Causes I/O Bottleneck

2.2.3.1 Why Does It Exist?

By default, one line of access log is written per request. Under high concurrency, disk I/O becomes a bottleneck, especially when the mechanical hard disk or cloud disk IOPS is limited. How to optimize?

```
# Option 1: Disable non-essential logs (use with caution in
production)
access_log off;
```

```
# Option 2: Enable buffered writing
access_log logs/access.log main buffer=64k flush=5s;
```

2.2.3.2 How to Verify?

```
iostat -x 1 # Observe whether %util and await decrease
```

2.2.4 High SSL/TLS Handshake Overhead

2.2.4.1 Why Does It Exist?

Each new HTTPS connection requires a complete TLS handshake (2-RTT), which is compute-intensive (especially RSA key exchange) and consumes a lot of CPU.

2.2.4.2 How to Optimize?

```
ssl_session_cache shared:SSL:10m; # Sessions shared among
workers
ssl_session_timeout 10m;
ssl_session_tickets on; # Enable ticket for fast
resumption (pay attention to key rotation)
ssl_protocols TLSv1.2 TLSv1.3;
ssl_prefer_server_ciphers off; # Let the client prioritize
efficient ciphers
keepalive_timeout 75s; # Reuse TCP+TLS connections
```

2.2.4.3 How to Verify?

- Use openssl s_client -connect host:443 -reconnect to test session reuse;
- Compare the QPS gap between HTTPS and HTTP during stress testing to see if it narrows;
- Monitor CPU usage (whether %sys in top is too high).

2.2.5 No Efficient Transmission Mechanism Enabled for Static Resources

2.2.5.1 Why Does It Exist?

sendfile, tcp_nopush, etc. are not enabled, resulting in multiple copies of data by the kernel, increasing latency.

2.2.5.2 How to Optimize?

```

sendfile on;           # Zero-copy file transfer
tcp_nopush on;        # Works with sendfile to accumulate data into
                        # complete packets before sending, reducing the number of packet
                        # headers and improving throughput for large files
tcp_nodelay on;        # After sendfile completes or in dynamic
                        # content scenarios, disable the Nagle algorithm to send small packets
                        # immediately, reducing latency
open_file_cache max=10000 inactive=60s; # Cache file metadata

```

Note: `tcp_nopush` and `tcp_nodelay` apply to different scenarios and do not conflict when enabled at the same time. `tcp_nopush` mainly takes effect when `sendfile` sends large files; `tcp_nodelay` takes effect in dynamic content or small-packet scenarios after `sendfile` completes. The combination of the two can simultaneously optimize throughput and latency.

2.2.5.3 How to Verify?

- Compare the static file download speed before and after enabling;
- Use `perf` or `bpftrace` to observe page faults and copy counts.

2.3 Comprehensive Verification Methods

After completing any of the above optimizations, the effect must be verified in the following ways:

Verification Target	Tool/Command	Metrics of Interest
Concurrent connection capability	<code>wrk -c 10000 -t 10 -d 30s</code> URL	Requests/sec, Latency
Connection queue status	<code>ss -lnt</code>	Whether Recv-Q is close to backlog
FD usage	<code>`lsof -p PID</code>	<code>wc -l`</code>
CPU utilization balance	<code>top</code> Or <code>htop</code>	Whether the load of each core is uniform
Network errors	<code>`netstat -s</code>	<code>grep -i drop`</code>
ALB internal status	Enable <code>stub_status</code>	active/accepts/handled/requests
Disk I/O pressure	<code>iostat -x 1</code>	%util, await

2.4 Summary

Dimension	Typical Bottleneck	Optimization Direction
Operating system	FD limit, TCP queue, TIME_WAIT	Adjust limits, sysctl parameters
ALB configuration	Worker count, Buffer, Log, SSL	Properly configure the events/http modules

3 Timeout Control Optimization

3.1 Performance Bottlenecks Caused by Improper Timeout Settings

ALB involves multiple timeout parameters, which act in two directions: Client ↔ ALB and ALB ↔ Upstream Service. Improper configuration will cause:

- Connections cannot be released in time → `worker_connections` exhausted;
- Requests get stuck without returning → Client timeout, poor user experience;
- When the backend is abnormal, ALB accumulates a large number of pending requests → Memory/CPU exhaustion.

3.2 Client-side Timeouts

Parameter	Default	Function	Risk
<code>client_header_timeout</code>	60s	Timeout for reading client request headers	Slowloris attacks can exhaust connections
<code>client_body_timeout</code>	60s	Timeout for reading client request body	When large file uploads are not rate-limited, the worker gets stuck
<code>keepalive_timeout</code>	75s	Client keep-alive connection hold time	If set too large, idle connections will accumulate

3.2.1 Optimization Suggestions:

```
# Adjust according to business; the API scenario can be shortened
client_header_timeout 10s;
client_body_timeout 10s;

# Set keepalive reasonably, balancing reuse and resource reclamation
keepalive_timeout 30s;          # Hold for 30 seconds
keepalive_requests 1000;       # A single connection handles at most
                                1000 requests (to prevent memory leaks)
```

3.2.2 Verification Method:

- Simulate slow requests (such as using curl to limit the speed to send a large file slow.txt[10MB]: `curl --limit-rate 1k --data-binary @slow.txt -X POST http://ip:port/api`);
- Observe whether `client timed out` appears in the error log;
- Use `ss -o state established` to check whether the ESTAB time of the connection exceeds expectations.

3.3 Upstream Service Timeouts

Parameter	Default	Function	Risk
<code>proxy_connect_timeout</code>	60s	Timeout for establishing a TCP connection with the upstream	Worker is blocked when the backend is down
<code>proxy_send_timeout</code>	60s	Timeout for sending requests to the upstream	Network congestion or slow backend reception
<code>proxy_read_timeout</code>	60s	Timeout for reading responses from the upstream	Backend processing is slow or hangs

 *The default 60s is extremely dangerous in high-concurrency scenarios! A slow interface may cause thousands of connections to be occupied.*

3.3.1 Optimization Suggestions (Taking an API Gateway as an Example):

```
location /api/ {
    proxy_pass http://backend;

    # Connection timeout: 1~3s is usually enough for a healthy
    backend on the intranet. If the backend is across data centers or
    the network quality is poor, it can be relaxed to 10s
    proxy_connect_timeout 10s;

    # Send/Read timeout: It is recommended to set it according to
    the business P99 in API scenarios (such as 500ms ~ 5s); for large
    file uploads or high-concurrency time-consuming business scenarios,
    it can be relaxed to 60s ~ 120s
    proxy_send_timeout 60s;
    proxy_read_timeout 60s;
```

```
# Other necessary configurations
proxy_http_version 1.1;
proxy_set_header Connection "";
}
```

3.3.2 🔍 Verification Method:

- Set `proxy_read_timeout` to 5s, then let the backend sleep for 10s, and observe whether ALB returns 504 Gateway Timeout after 5s;
- Monitor whether the active connections in `alb_status` surge during anomalies;
- The log should contain `upstream timed out (110: Connection timed out)`.

💡 Best Practice:

The timeout should be slightly greater than the backend P99 response time, and should be combined with a circuit breaker/degradation mechanism.

4 Keep-Alive Configuration Guide

Keep-alive needs to be configured cooperatively in three links to take effect:

```
[Client] ←(HTTP Keep-Alive)→ [ALB] ←(Upstream Keep-Alive)→ [Upstream Service]
```

4.1 Client ↔ ALB: Enable HTTP Keep-Alive

ALB supports client keep-alive by default, but it needs to be configured reasonably:

```
http {
    # Global keepalive settings
    keepalive_timeout 30s;          # Connection hold for 30 seconds
    keepalive_requests 1000;        # A single connection handles at
    most 1000 requests (to prevent memory leaks)

    # HTTP/2 multiplexing is automatically enabled, no additional
    configuration required
}
```

✓ *Applicable to Web browsers, mobile apps, internal service calls, and other clients that support keep-alive.*

4.2 ALB ↔ Upstream: Enable Upstream Keep-Alive (Key!)

It has been described in detail earlier. Here it is integrated into a complete template:

```
upstream backend {
    server 192.168.1.10:8080;
    server 192.168.1.11:8080;
    keepalive 64; # Each worker caches 64 idle connections
}

server {
    location /api/ {
```

```

proxy_pass http://backend;
proxy_http_version 1.1;
proxy_set_header Connection ""; # Clear the Connection
header

# Timeout control (must be combined with keep-alive)
proxy_connect_timeout 3s;
proxy_send_timeout 120s;
proxy_read_timeout 120s;
}
}

```

4.3 Backend Service: Needs to Support HTTP/1.1 and Maintain Connections

- Backend frameworks (such as Spring Boot, Gin) usually support it by default; frameworks such as Flask need to confirm the configuration and enable multi-thread/multi-process mode;
- Ensure that Connection: close is not actively sent;
- If TLS is used, it is recommended to enable session resumption.

4.4 Comprehensive Verification: How to Confirm That Keep-Alive and Timeout Configurations Are in Effect?

Verification Target	Method
Client keep-alive	Use browser developer tools or <code>curl -v</code> to check whether connections are reused; or use <code>tcpdump</code> to observe multiple HTTP requests in the same TCP stream
Upstream keep-alive	<code>ss -tnp grep alb grep :8080</code> to check whether ESTAB connections are stably present
Timeout in effect	Simulate backend latency > timeout, and confirm that ALB returns 504 instead of hanging
Connection reuse effect	Compare the QPS and CPU usage with keepalive enabled/disabled through stress testing

Example stress test command (verifying upstream keepalive):

```
# High-concurrency short requests
wrk -t16 -c1000 -d30s --latency http://alb/api/fast-endpoint

# Observe metrics:
# - Requests/sec increases
# - The number of timewait in ss -s drops significantly
# - The number of backend accept() decreases
```

5 Typical Scenario Optimization Cases

5.1 Scenario 1: Web Site Access Acceleration Optimization

5.1.1 Business Scenario Description

A company has deployed a portal website that contains a large number of static resources (images, CSS, JS files). The traffic has gradually increased, and users report that the page loads slowly.

5.1.2 Optimization Goal

Improve the access speed of static resources, reduce server load, and improve user experience.

5.1.3 Optimization Solution

- Enable static resource caching

```
http {
    # Enable file caching
    open_file_cache max=1000 inactive=20s;
    open_file_cache_valid 30s;
    open_file_cache_min Uses 2;
    open_file_cache_errors on;

    server {
        listen 80;
        root /var/www/html;

        # Set caching for static resources
        location ~* \.(jpg|jpeg|png|gif|ico|css|js)$ {
            expires 1y;
            add_header Cache-Control "public, immutable";
        }
    }
}
```

- Enable gzip compression

```
http {
    gzip on;
    gzip_vary on;
    gzip_min_length 1024;
    gzip_types text/plain text/css application/json
application/javascript text/xml application/xml;
}
```

5.1.4 Optimization Effect

- Static resource loading speed increased by more than 50%
- Server CPU usage reduced by 20%
- User page load time significantly shortened

5.2 Scenario 2: API Gateway Performance Optimization

5.2.1 Business Scenario Description

A company's API gateway needs to handle a large number of mobile requests, and during peak hours, responses are slow and some requests fail.

5.2.2 Optimization Goal

Improve the concurrent processing capability of the API gateway and ensure service stability.

5.2.3 Optimization Solution

- 1. Adjust the system file descriptor limit

```
# Edit the alb startup script and add the following two
lines at the beginning of the script
# [ALB installation directory]/bin/start-alb.sh or start-
alb-and-console.sh
ulimit -n 65536
ulimit -Hn 65536
```

- 2. Optimize ALB configuration

```

worker_processes auto;
worker_rlimit_nofile 65536;

events {
    use epoll;
    worker_connections 4096;
    multi_accept on;
}

http {
    # Enable keep-alive reuse
    upstream api_backend {
        server 192.168.1.30:8080 max_fails=3 fail_timeout=10s;
        server 192.168.1.31:8080 max_fails=3 fail_timeout=10s;
        keepalive 32;
    }

    # Limit the request rate of a single IP (to prevent
    malicious interface scraping)
    limit_req_zone $binary_remote_addr zone=api:10m rate=10r/s;

    server {
        listen 80 reuseport;

        # API routing
        location /api/ {
            proxy_pass http://api_backend;
            # Enable HTTP/1.1 keep-alive
            proxy_http_version 1.1;
            proxy_set_header Connection "";

            # Set timeouts reasonably

```

```

        proxy_connect_timeout 5s;
        proxy_send_timeout 120s;
        proxy_read_timeout 120s;

        # Limit the request rate of a single IP (to prevent
        malicious interface scraping)
        limit_req zone=api burst=20 nodelay;
    }
}

```

5.3 Scenario 3: File Download Service Optimization

5.3.1 Business Scenario Description

The enterprise needs to provide a large file download service. Users report that the download speed is slow and server resource consumption is high.

5.3.2 Optimization Goal

Improve the large file download speed and reduce server resource consumption.

5.3.3 Optimization Solution

- 1. Enable large file optimization configuration

```

http {
    # Adjust buffer size (applicable to reverse proxy large
    file download scenarios)
    proxy_buffering on;
    proxy_buffer_size 128k;
    proxy_buffers 4 256k;
    proxy_busy_buffers_size 256k;

    # Enable chunked download support
    proxy_http_version 1.1;
}

```

```

server {
    listen 80;

    # Static file direct download (use root, no proxy_*
parameters required)

    location /downloads/ {
        root /data;

        # Support resumable downloads
        add_header Accept-Ranges bytes;

        # Special configuration for large file downloads
        location ~* \.(iso|zip|exe)$ {
            # Limit download speed (to prevent bandwidth
from being fully occupied)

            limit_rate 2m; # Limit to 2MB/s

            # Enable zero-copy transfer to improve static
file download performance

            sendfile on;
            tcp_nopush on;
        }
    }
}

```

5.3.4 Optimization Effect

- Large file download speed increased by 60%
- Server memory consumption reduced by 30%
- Support for resumable downloads, improving user experience

5.4 Scenario 4: ALB Reverse Proxying AAS Service Performance

Optimization

Goal: Through operating system tuning + ALB configuration optimization, enable ALB to efficiently proxy the backend AAS service and support high-concurrency API requests.

5.4.1 Operating System Level Tuning (Linux)

Increase the file descriptor limit

```
# Edit the alb startup script and add the following two lines at the
beginning of the script
# [ALB installation directory]/bin/start-alb.sh or start-alb-and-
console.sh
ulimit -n 100000
ulimit -Hn 100000
```

Optimize the TCP network stack

```
# Edit /etc/sysctl.conf and add the following content:
net.core.somaxconn = 65535
net.ipv4.tcp_max_syn_backlog = 65535
net.ipv4.ip_local_port_range = 1024 65535
net.ipv4.tcp_tw_reuse = 1
net.ipv4.tcp_fin_timeout = 15
fs.file-max = 2097152

# Apply
sysctl -p
```

AAS Performance Optimization: Ensure that the AAS application server has keep-alive support enabled, and adjust the thread pool and connection pool size according to the actual business. For detailed optimization, please refer to the AAS product performance tuning document.

5.4.2 ALB Configuration Optimization

```

user root;

worker_processes auto;
worker_rlimit_nofile 131072;      # Must be ≥ worker_connections × 2

events {
    use epoll;
    worker_connections 65536;
    multi_accept on;
}

http {
    # Global timeout control (applies to client ↔ ALB)
    client_header_timeout 120s;    # Relax appropriately for large
file upload scenarios
    client_body_timeout 120s;
    keepalive_timeout 300s;        # Client keep-alive connection
hold time
    keepalive_requests 1000;       # A single connection handles at
most 1000 requests

    upstream aas_backend {
        server 192.168.1.100:8080 max_fails=3 fail_timeout=10s;
        server 192.168.1.101:8080 max_fails=3 fail_timeout=10s;
        server 192.168.1.102:8080 max_fails=3 fail_timeout=10s;
        keepalive 128;             # Each worker maintains 128 idle
keep-alive connections
        keepalive_requests 1000;   # A single upstream connection
handles at most 1000 requests
        keepalive_timeout 75s;     # Upstream idle connection timeout
(applies to ALB ↔ backend)
    }

    server {
        listen 8080 reuseport;

```

```

location / {
    proxy_pass http://aas_backend/;

    # Enable keep-alive with AAS
    proxy_http_version 1.1;
    proxy_set_header Connection "";

    # Timeout settings (slightly greater than AAS processing
time)

    proxy_connect_timeout 10s;
    proxy_send_timeout 120s;
    proxy_read_timeout 120s;

    # Pass client information
    proxy_set_header Host $host;
    proxy_set_header X-Real-IP $remote_addr;
    proxy_set_header X-Forwarded-For
$proxy_add_x_forwarded_for;

}

# Log buffered writing
access_log access.log main buffer=64k flush=5s;
}
}

```

5.4.3 Stress Test Verification

```

# Use wrk to simulate high-concurrency requests
wrk -t16 -c2000 -d60s --latency http://your-alb-server:8080/api/test

# Observe metrics:
# - Whether Requests/sec is stably improved
# - ALB error log has no "too many open files"

```

```
# - ss -tnp | grep :8080 to check whether ESTAB connections are
maintained (instead of a large number of TIME-WAIT)
```

5.5 Scenario 5: ALB Million-Level Concurrent Connection Test (C1000K)

5.5.1 Background

5.5.1.1 Typical Business Scenarios:

- WebSocket long-connection gateway (such as chat, IoT device access)
- HTTP long polling service
- Server-Sent Events (SSE) push platform
- Microservice API gateway (high keep-alive connections)

5.5.1.2 Important Prerequisite:

Million-level concurrency $\neq$ Million-level QPS. Here, it means simultaneously maintaining 1,000,000 TCP connections, each of which may communicate at a low frequency (such as a heartbeat once per minute), but must remain alive.

5.5.1.3 Core Difficulty:

ALB needs to maintain two sockets for each connection: client $\leftrightarrow$ alb + alb $\leftrightarrow$ backend $\rightarrow$ Total FD count $\approx 2 \times$ number of concurrent connections If upstream keepalive is not enabled, the backend has to bear the pressure of millions of connections (which is almost infeasible) It is necessary to compress the number of backend connections to a reasonable range through connection reuse (keep-alive pool)

5.5.2 Solution

- ALB layer: Undertake million-level connection access, TLS termination, and routing distribution
- Backend layer: Only needs to handle active requests and communicates with ALB through a small-scale keep-alive pool
- Key strategy: Massive front-end connections + limited back-end connection pool

 *Ideal ratio:*

1,000,000 client connections $\rightarrow$ ALB $\rightarrow$ 1,000 ~ 10,000 upstream keep-alive connections (reuse ratio 100:1 ~ 1000:1)

5.5.3 Comprehensive Operating System Tuning (Linux)

5.5.3.1 File Descriptor (FD) — The Most Critical!

```
# Edit the alb startup script and add the following two lines at the
beginning of the script
# [ALB installation directory]/bin/start-alb.sh or start-alb-and-
console.sh
ulimit -n 150000
ulimit -Hn 150000

# Global file handle upper limit
echo 'fs.file-max = 2500000' >> /etc/sysctl.conf
echo 'fs.nr_open = 2500000' >> /etc/sysctl.conf
```

 *ulimit is set to 150,000, but the global setting is 2,500,000?*

Minimum number of connections for ALB million-level connections : million connections × 2 (client + upstream) = 2,000,000.

Number of ALB workers : ulimit sets the number of files that each process can open to 150,000, so 2,000,000 / 150,000 = 14 worker processes (that is, 16 or more CPU cores are recommended).

Set 2,500,000: In addition to the 2,000,000 file numbers required by ALB, the system itself and other services also need to open files, that is, the 2,500,000 open file count is the upper limit of the entire system file handle count.

5.5.3.2 Network Stack Tuning

```
# /etc/sysctl.conf
net.core.somaxconn = 1000000
net.core.netdev_max_backlog = 50000
net.ipv4.tcp_max_syn_backlog = 300000
net.ipv4.tcp_syncookies = 1
net.ipv4.tcp_synack_retries = 2

# TIME-WAIT optimization (generated when ALB actively closes the
client connection)
net.ipv4.tcp_tw_reuse = 1
```

```

net.ipv4.tcp_fin_timeout = 10
net.ipv4.tcp_max_tw_buckets = 2000000

# Memory buffers (support large concurrency)
net.core.rmem_default = 262144
net.core.wmem_default = 262144
net.core.rmem_max = 67108864
net.core.wmem_max = 67108864
net.ipv4.tcp_rmem = 4096 16384 67108864
net.ipv4.tcp_wmem = 4096 16384 67108864

# Port range (if ALB also acts as a client to initiate connections)
net.ipv4.ip_local_port_range = 1024 65535

```

After modifying 1 and 2, run the following command to apply:

```
sysctl -p
```

5.5.3.3 Kernel and Hardware Recommendations

- Linux kernel ≥ 4.5 (stable support for reuseport + epoll)
- CPU ≥ 16 cores (the context switch overhead of million-level connections is large)
- Memory $\geq 64\text{GB}$ (each connection is about 4~8KB, million connections are about 8GB, plus buffer and backend cache)
- Disable swap: `swapoff -a`
- Use SSD: reduce the I/O impact of logs or temporary files

5.5.4 ALB Configuration Optimization

```

user root;
worker_processes auto;          # Match the number of CPU cores
                                 (such as 16)
worker_rlimit_nofile 300000;    # Number of files opened by a
single worker, must be  $\geq$  worker_connections  $\times$  2

error_log error.log warn;

```

```

events {
    use epoll;
    worker_connections 150000;          # A single worker supports
150,000 connections (16 × 150,000 = 2,400,000)
    multi_accept on;
}

http {
    # Disable access log (during stress testing), or use
asynchronous buffering
    access_log off;
    error_log error.log crit;

    # Efficient transmission
    sendfile on;
    tcp_nopush on;
    tcp_nodelay on;

    # Client keep-alive (maintain long connections)
    keepalive_timeout 300s;             # Disconnect after 5 minutes of
no operation
    keepalive_requests 1000000;        # Prevent single-connection
memory leaks

    # Upstream keep-alive pool (core!)
    upstream backend_pool {
        # Assuming 10 backends, each with max_conns=1000, the total
backend connection upper limit is 10,000
        server 192.168.1.20:8080 max_conns=1000 max_fails=3
fail_timeout=10s;
        server 192.168.1.21:8080 max_conns=1000 max_fails=3
fail_timeout=10s;
        server 192.168.1.22:8080 max_conns=1000 max_fails=3
fail_timeout=10s;
        server 192.168.1.23:8080 max_conns=1000 max_fails=3
fail_timeout=10s;
    }
}

```

```

        server 192.168.1.24:8080 max_conns=1000 max_fails=3
fail_timeout=10s;
        server 192.168.1.25:8080 max_conns=1000 max_fails=3
fail_timeout=10s;
        server 192.168.1.26:8080 max_conns=1000 max_fails=3
fail_timeout=10s;
        server 192.168.1.27:8080 max_conns=1000 max_fails=3
fail_timeout=10s;
        server 192.168.1.28:8080 max_conns=1000 max_fails=3
fail_timeout=10s;
        server 192.168.1.29:8080 max_conns=1000 max_fails=3
fail_timeout=10s;

        # Each worker maintains 625 idle keep-alive connections (16
workers × 625 = 10,000, matching the total max_conns)
        keepalive 625;
    }

    server {
        listen 8080 reuseport backlog=1000000;

        location / {
            proxy_pass http://backend_pool/;

            # Enable upstream keep-alive (required!)
            proxy_http_version 1.1;
            proxy_set_header Connection "";

            # Timeout settings (according to the business heartbeat
interval)
            proxy_connect_timeout 15s;
            proxy_send_timeout 300s;      # Set according to the
request body size and network conditions
            proxy_read_timeout 300s;      # Set according to the
backend business processing time, should be slightly greater than
the backend P99 response time

```

```

        # Pass necessary headers
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For
$proxy_add_x_forwarded_for;
        proxy_set_header X-Forwarded-Proto $scheme;
    }
}
}

```

✓ Key Point Description:

- *keepalive 1000*: each worker maintains 1000 upstream idle connections;
- *max_conns=1000*: limits the maximum number of connections to a single backend to prevent overload;
- *proxy_send/read_timeout ≥ keepalive_timeout*: avoid ALB closing the connection prematurely;
- *reuseport*: enables kernel-level load balancing and avoids accept lock contention.

5.5.5 Backend Optimization

- Support keep-alive

5.5.6 Common Problems and Avoidance

Problem	Cause	Solution
ALB reports "too many open files"	Insufficient FD limit	Increase LimitNOFILE + worker_rlimit_nofile
Backend connection count surges	Upstream keepalive not enabled	Check proxy_http_version 1.1 + Connection ""
Connections are frequently disconnected	Timeout settings do not match	Ensure proxy_read_timeout ≥ keepalive_timeout
Single-core CPU 100%	reuseport not enabled	Add listen ... reuseport;
Slow TLS handshake	Session not reused	Enable ssl_session_cache + ssl_session_tickets

5.5.7 Summary

Summary: The Core Formula for Million-Level Concurrency in Reverse Proxy

Million-level concurrency = Operating system resources × ALB connection management × Upstream keep-alive reuse × Backend asynchronous capability

Layer	Key Action
OS	Increase FD, optimize TCP, disable swap
ALB	reuseport + large worker_connections + upstream keepalive
Backend	Asynchronous model + keep-alive support + externalized state
Verification	Stress test + monitoring + log analysis

6 Performance Testing and Troubleshooting

6.1 Summary of Diagnostic Commands

```
# View ALB error logs (key focus)
tail -f ALB installation directory/logs/error.log

# View the current connection status
ss -s

# View ALB process FD usage
lsof -p $(pgrep alb | head -1) | wc -l

# View the system FD limit
ulimit -n
cat /proc/sys/fs/file-max

# View TCP queue overflow (key!)
netstat -s | grep -i "listen.*overflows\|drop"

# View CPU core load
top

# View ALB version
ALB installation directory/sbin/alb -v
```

6.2 Error Symptom: A large number of requests return 502 Bad Gateway during stress testing

6.2.1 🔍 Error Analysis

ALB cannot establish a connection with the upstream backend or communication fails;

Common causes:

- The backend service is down or the port is not listening;
- proxy_connect_timeout is too short, and the backend starts slowly;
- The number of backend connections reaches the upper limit (such as Tomcat maxConnections exhausted);
- DNS resolution fails (if the upstream uses a domain name).

6.2.2 Solution

```
location / {
    proxy_pass http://backend;
    proxy_connect_timeout 15s;    # Adjust according to the backend
    startup time
    proxy_read_timeout 30s;
    resolver 8.8.8.8 valid=30s; # If a domain name is used,
    explicitly specify DNS
}
```

6.2.3 Verification Method

- View the ALB error log:

```
grep "502" logs/error.log | head -5
# Typical log: connect() failed (111: Connection refused)
```

- Manually use curl to test whether the upstream is reachable;
- Monitor the backend accept queue during stress testing: use ss -lnt to check whether Recv-Q overflows.

6.3 Error Symptom: Stress testing is normal at the beginning, then QPS drops sharply and connections accumulate

6.3.1 Error Analysis

- ALB worker_connections or system file descriptor (FD) exhausted;
- accept() failed (24: Too many open files) appears in the log;
- Reason: The default worker_connections=1024 + system ulimit -n=1024 cannot support high concurrency.

6.3.2 Solution

- Operating system level:

```
# Edit the alb startup script and add the following two lines at the
beginning of the script
# [ALB installation directory]/bin/start-alb.sh or start-alb-and-
console.sh
ulimit -n 100000
ulimit -Hn 100000

# If using systemd, add LimitNOFILE=100000 at the end of [Service]
# Edit /etc/systemd/system/alb.service
[Service]
....
User=root
Type=forking
LimitNOFILE=100000 # Add this line
# After modification, use systemctl daemon-reload and then restart
ALB
```

ALB configuration

```
worker_rlimit_nofile 100000;
events {
    worker_connections 65536;
}
```

6.3.3 Verification Method

- Check the process FD limit:

```
cat /proc/$(pgrep alb | head -1)/limits | grep "open files"
```

- Monitor FD usage in real time:

```
lsof -p $(pgrep alb | head -1) | wc -l
```

- Observe whether "Too many open files" still appears in the error log during stress testing.

6.4 Error Symptom: A large number of 504 Gateway Timeout occur during stress testing

6.4.1 🔍 Error Analysis

- ALB timed out waiting for the upstream response;
- The default proxy_read_timeout=60s, but the backend processing is slow or hangs;
- Under high concurrency, a small number of slow requests can bring down the entire worker.

6.4.2 ⚙️ Solution

- Set timeouts reasonably (according to the business P99 response time):

```
proxy_connect_timeout 15s;
proxy_send_timeout 60s;
proxy_read_timeout 60s;
```

- Backend optimization: Introduce asynchronous processing and circuit breaker mechanism;
- Avoid long-tail requests: protect the backend through rate limiting (limit_req).

6.4.3 ✅ Verification Method

- Set proxy_read_timeout to 5s, then let the backend sleep for 10s, and confirm whether ALB returns 504 after 5s;
- View the error log:

```
grep "upstream timed out" error.log
```

6.5 Error Symptom: The client reports "Connection reset by peer" or "Connection closed" during stress testing

6.5.1 🔍 Error Analysis

- ALB actively closed the connection, commonly seen in:
 - The client sent an incomplete request (such as sending only the header but not the body);
 - client_body_timeout or client_header_timeout is triggered;
 - The backend returns an illegal response (such as a chunked encoding error).

6.5.2 ⚙️ Solution

- Adjust client behavior (ensure a complete HTTP request);
- Relax timeouts appropriately (only for trusted intranets):

```
client_header_timeout 30s;
client_body_timeout 30s;
```

- Enable more detailed logs to locate problems:

```
error_log error.log debug;
```

6.5.3 Verification Method

- Use tcpdump to capture packets and analyze the source of the RST packets;
- Compare the error details with debug log enabled/disabled.

6.6 Error Symptom: After enabling upstream keepalive, QPS does not improve or even decreases

6.6.1 Error Analysis

- Configuration omission: proxy_http_version 1.1 is not set or the Connection header is not cleared;
- The backend does not support keep-alive: returns Connection: close;
- The keepalive connection pool is too small: new connections are frequently created, and the reuse rate is low.

6.6.2 Solution

```
upstream backend {
    server 192.168.1.10:8080;
    keepalive 128; # Adjust according to QPS
}

location / {
    proxy_pass http://backend;
    proxy_http_version 1.1;
    proxy_set_header Connection ""; # Key!
}
```

6.6.3 Verification Method

- Check the upstream connection status:
 - `ss -tnp | grep ':8080' | grep alb | grep ESTAB | wc -l`
 - If the value is stably $\approx \text{worker_processes} \times \text{keepalive}$, it indicates that it is in effect;
- Stress test comparison: QPS and TIME-WAIT count before and after enabling.

6.7 Error Symptom: The system CPU sys% (kernel mode) is too high during stress testing, reaching 70%+

6.7.1 Error Analysis

- Frequent syscalls (such as read/write) cause large context switch overhead;
- Zero-copy mechanisms such as sendfile and tcp_nopush are not enabled;
- Frequent sending of small packets (Nagle algorithm is not optimized).

6.7.2 Solution

```
sendfile on;
tcp_nopush on;    # Send after accumulating data, reducing packets
tcp_nodelay on;   # Send small packets immediately (applicable to
dynamic content)
```

- Prefer `sendfile` for static resources;
- For dynamic APIs, `tcp_nopush` can be disabled and `tcp_nodelay` enabled.

6.7.3 Verification Method

- Use `perf top` to observe whether the proportion of kernel functions (such as `tcp_sendmsg`) decreases;
- Compare the `%sys` metric in `sar -u` before and after optimization.

6.8 Error Symptom: Under multi-core CPU, only CPU0 is 100% loaded, and other cores are idle

6.8.1 Error Analysis

- reuseport is not enabled, and all connections are accepted by a single worker;

- Network card interrupts are concentrated on CPU0.

6.8.2 Solution

```
listen 80 reuseport;  
listen 443 ssl http2 reuseport;
```

6.8.3 Verification Method

```
top # Observe whether the load of each CPU core is balanced  
ss -lnt | grep :80 # Check whether there are multiple listening  
sockets (characteristic of reuseport)
```

7 Performance Testing Error Quick Reference Table

Error Symptom	Possible Cause	Key Check Command	Solution
A large number of 502 Bad Gateway	Backend unreachable / connection refused / DNS failure	<code>curl -v http://backend grep "connect.*failed" logs/error.log</code>	Check the backend service status Set resolver (if a domain name is used) Increase <code>proxy_connect_timeout</code>
A large number of 504 Gateway Timeout	Backend response is slow / hangs	<code>grep "upstream timed out" logs/error.log `ss -tnp</code>	<code>grep :backend_port`</code>
QPS cannot go up, connections accumulate	FD exhausted / <code>worker_connections</code> insufficient	<code>`cat /proc/\$(pgrep alb)/limits</code>	<code>grep open
lsof -p \$(pgrep alb)</code>
Stress test reports "Too many open files"	System or ALB file descriptor limit is too low	<code>`dmesg</code>	<code>tail
systemctl show alb</code>
HTTPS QPS is much lower than HTTP	TLS handshake not reused / cipher is inefficient	<code>`openssl s_client -connect host:443 -reconnect 2>/dev/null</code>	<code>grep Reused`</code>
Only a single CPU core is 100%, the rest are idle	<code>reuseport</code> not enabled / <code>accept lock contention</code>	<code>top `ss -lnt</code>	<code>grep :80` (check whether there are multiple listening sockets)</code>
Enabling upstream keepalive is invalid	Missing HTTP/1.1 or Connection header not cleared	<code>`ss -tnp</code>	<code>grep :backend_port</code>
The client reports "Connection reset by peer"	Timeout triggered / incomplete request / backend abnormally closed	<code>tcpdump -i any -nn port 80 -w debug.pcap grep "client timed out" logs/error.log</code>	Adjust <code>client_header_timeout / client_body_timeout</code> Ensure that the stress test tool sends complete HTTP requests

TIME-WAIT connection count surges (>100,000)	Connections not reused / ALB is the active closer	<code>ss -s</code> <code>`netstat -ant</code>	<code>awk '/TIME-WAIT/ {count++} END {print count+0}'`</code>
Poor static file performance	Zero-copy not enabled / buffer is small	<code>`strace -p \$(pgrep alb) 2>&1</code>	<code>grep -E 'read</code>

8 Performance Optimization Key Parameter List

Table description:

- Directive: ALB configuration directive, referring to the parameter name of the ALB function configuration.
- Context: The scope of the parameter.

Scope Identifier	Corresponding Configuration Block	Description
main	Global configuration area	Global parameter configuration area at the top level of the configuration file, not contained in any block
events	Event model configuration block	Configured within the <code>events{}</code> block, used to set ALB event processing model parameters
http	HTTP service configuration block	Configured within the <code>http{}</code> block, used to set HTTP service and proxy related parameters
server	Virtual server configuration block	Configured within the <code>server{}</code> block, used to set HTTP virtual server parameters
location	URL matching configuration block	Configured within the <code>location{}</code> block, used to set processing parameters for specific URL paths
upstream	Upstream service configuration block	Configured within the <code>upstream{}</code> block, used to set the parameters of the backend server group for reverse proxy
stream	Layer 4 proxy configuration block	Configured within the <code>stream{}</code> block, used to set TCP/UDP layer 4 proxy parameters
main/http/server/location	Multi-level configuration area	Can be configured at any level of main, http, server, and location
http/server/location	HTTP configuration area	Can only be configured at the http, server, and location levels

8.1 Global and Process Control

Directive	Context	Function Description	Default	Recommended Value	Notes
<code>worker_processes</code>	main	Number of ALB worker	auto	auto or number of CPU	auto automatically matches the number

		processes		cores (such as 8)	of CPU cores; must be set for multi-core
<code>worker_rlimit_nofile</code>	main	Maximum number of file descriptors per worker	No limit (inherited from the system)	65536 ~ 262144	Must be $\geq$ <code>worker_connections</code> $\times$ 2 (client + upstream)
<code>worker_cpu_affinity</code>	main	Bind workers to specified CPU cores	Not bound	auto (or manual mask)	Reduce context switches, improve cache hit rate (optional)

8.2 Event Model (Events)

Directive	Context	Function Description	Default	Recommended Value	Notes
<code>worker_connections</code>	events	Maximum number of concurrent connections per worker	512	10000 ~ 200000	Total concurrency $\approx$ <code>worker_processes</code> $\times$ <code>worker_connections</code>
<code>use</code>	events	Event-driven model	Automatically selected	epoll (Linux)	Must be explicitly set to epoll on Linux (although it is the default, it is recommended to write it out)
<code>multi_accept</code>	events	Accept multiple connections at once	off	on	Cooperate with epoll to improve throughput and avoid starvation

8.3 HTTP Global Configuration

Directive	Context	Function Description	Default	Recommended Value	Not
<code>sendfile</code>	http/server/location	Zero-copy transfer of	off	on	Must be enabled

		static files			static res services, reducing kernel/us mode co
tcp_nopush	http/server/location	Send after accumulating data (cooperate with sendfile)	off	on	Improve efficiency large file transfer, reducing packets
tcp_nodelay	http/server/location	Disable the Nagle algorithm, send small packets immediately	off	on (API scenario)	Recomm to enable dynamic content/, reducing latency
open_file_cache	http/server/location	Cache file metadata (fd/stat)	off	max=10000 inactive=60s	Required high-frec static file
open_file_cache_valid	http/server/location	Cache validation interval	60s	30s ~ 120s	Used in conjuncti with open_file
open_file_cache_min_uses	http/server/location	Minimum number of file accesses to cache	1	2	Prevent caching c files

8.4 Client Connection and Timeout

Directive	Context	Function Description	Default	Recommended Value	Notes
keepalive_timeout	http/server/location	Client keep-alive connection hold time	75s	15s ~ 30s (Web API) 300s (long connection)	If set too large, idle connectio will accumulat

keepalive_requests	http/server/location	Maximum number of requests per connection	100	1000 ~ 1000000	Prevent single-connection memory leaks
client_header_timeout	http/server/location	Timeout for reading request headers	60s	10s ~ 30s Can be increased under high concurrency, such as 120s	Prevent Slowloris attacks
client_body_timeout	http/server/location	Timeout for reading request body	60s	10s ~ 120s For file uploads, it can be set according to the actual time	Needs to be increased for large file uploads, can be reduced for APIs
reset_timeout_connection	http/server/location	Send RST instead of FIN after timeout	off	on	Quickly release TIME-WAIT (use with caution)

8.5 Reverse Proxy (Proxy)

Directive	Context	Function Description	Default	Recommended Value	Notes
proxy_http_version	http/server/location	HTTP version used by the proxy	1.0	1.1	Must be set to 1.1 to enable upstream keepalive
proxy_set_header Connection ""	location	Clear the Connection header	None	""	Used in conjunction with keepalive to prevent close

proxy_connect_timeout	http/server/location	Timeout for establishing a connection with the upstream	60s	1s ~ 10s Can be increased under high concurrency, such as 120s	Fast failure on backend health check
proxy_send_timeout	http/server/location	Timeout for sending requests to the upstream	60s	5s ~ 30s Can be increased under high concurrency, such as 120s	Set according to business p99
proxy_read_timeout	http/server/location	Timeout for reading responses from the upstream	60s	5s ~ 300s Can be increased for high concurrency or time-consuming business	Can be set larger for long-connection scenarios
proxy_buffering	http/server/location	Whether to buffer backend responses	on	on (default)	Disabling will increase backend pressure, generally not disabled
proxy_buffer_size	http/server/location	Size of the first buffer	4k 8k	16k	Contains the response header, recommended to be $\geq 16k$
proxy_buffers	http/server/location	Number and size of response body buffers	8 4k 8k	8 32k	Increase appropriately in high-throughput scenarios

8.6 Upstream Keep-Alive Pool

Directive	Context	Function Description	Default	Recommended Setting Range	Notes
<code>keepalive</code>	upstream	Number of idle keep-alive connections per worker	None	32 ~ 256	Total connection pool = <code>worker_processes × keepalive</code>
<code>keepalive_requests</code>	upstream	Maximum number of requests per upstream connection	1000	1000 (default, reasonable)	Prevent connection memory leaks
<code>keepalive_timeout</code>	upstream	Upstream idle connection timeout	60s	60s (default, reasonable)	Usually no need to modify

8.7 Logs

Directive	Context	Function Description	Default	Recommended Value	Notes
<code>access_log</code>	http/server/location	Access log	<code>logs/access.log</code> combined	<code>off</code> (stress testing) Or <code>buffer=64k</code> <code>flush=5s</code>	Disable or buffer writes under high concurrency
<code>error_log</code>	main/http/server	Error log level	<code>error</code>	<code>warn</code> (production) <code>crit</code> (stress testing)	Avoid debug logs slowing down performance

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

