



APUSIC
固若长城
睿比世界

效能最佳化手冊

金蝶Apusic負載均衡器

版權所有 © 深圳市金蝶天燕雲計算股份有限公司2026。保留所有權利。

版權聲明

本文件所涉及的軟件著作權、版權等知識產權已依法進行了註冊，由金蝶天燕雲計算股份有限公司合法擁有。受《中華人民共和國著作權法》《計算機軟件保護條例》《知識產權保護條例》和相關國際版權條約、法律、法規以及其它知識產權法律和條約的保護。未經授權許可，不得非法使用。

免責聲明

本文件包含的版權資訊由金蝶天燕雲計算股份有限公司合法擁有，受法律的保護，金蝶天燕雲計算股份有限公司對本文件可能涉及到的非金蝶天燕雲計算股份有限公司的資訊不承擔任何責任。在法律允許的範圍內，您可以查閱並僅能夠在《中華人民共和國著作權法》規定的合法範圍內複製和打印本文件。任何單位和個人未經金蝶天燕雲計算股份有限公司書面授權許可，不得使用、修改、再發佈本文件的任何部分和內容，否則將被視為侵權，金蝶天燕雲計算股份有限公司有依法追究其責任的權利。

本文件如有更新，不另行通知。對本文件中的問題您可向金蝶天燕雲計算股份有限公司告知或查詢。未經本公司明確授予的任何權利均予保留。

商標聲明

 是深圳市金蝶天燕雲計算股份有限公司向中華人民共和國國家商標局申請註冊的註冊商標，註冊商標專用權由金蝶天燕合法擁有，受法律保護。未經金蝶天燕的書面許可，任何單位及個人不得以任何方式或理由對該商標的任何部分進行使用、複製、修改、傳播、抄錄或與其它產品捆綁使用銷售。凡侵犯金蝶天燕商標權的，金蝶天燕將依法追究其法律責任。本文件提及的其他所有商標或註冊商標，由各自的所有人擁有。

目錄

- 1 前言
 - 1.1 適用對象
 - 1.2 相關手冊
 - 1.3 技術支援
- 2 效能瓶頸來源分析（作業系統 + ALB雙維度詳解）
 - 2.1 作業系統層面瓶頸
 - 2.1.1 檔案描述符（FD）限制
 - 2.1.1.1 為什麼存在？
 - 2.1.1.2 如何修改？
 - 2.1.1.3 如何驗證？
 - 2.1.2 TCP 連線佇列與網路堆疊參數
 - 2.1.2.1 為什麼存在？
 - 2.1.2.2 如何修改？
 - 2.1.2.3 如何驗證？
 - 2.1.3 TIME_WAIT 連線堆積
 - 2.1.3.1 為什麼存在？
 - 2.1.3.2 如何緩解？
 - 2.1.3.3 如何驗證？
 - 2.1.4 CPU 與中斷親和性
 - 2.1.4.1 為什麼存在？
 - 2.1.4.2 如何最佳化？
 - 2.1.4.3 如何驗證？
 - 2.2 ALB 配置瓶頸
 - 2.2.1 Worker 行程與連線數配置不當
 - 2.2.1.1 為什麼存在？
 - 2.2.1.2 如何最佳化？
 - 2.2.1.3 如何驗證？
 - 2.2.2 緩衝區（Buffer）配置不合理
 - 2.2.2.1 為什麼存在？
 - 2.2.2.2 如何最佳化？
 - 2.2.2.3 如何驗證？
 - 2.2.3 日誌寫入頻繁導致 I/O 瓶頸

- 2.2.3.1 為什麼存在？
- 2.2.3.2 如何驗證？
- 2.2.4 SSL/TLS 交握開銷大
 - 2.2.4.1 為什麼存在？
 - 2.2.4.2 如何最佳化？
 - 2.2.4.3 如何驗證？
- 2.2.5 靜態資源未啟用高效傳輸機制
 - 2.2.5.1 為什麼存在？
 - 2.2.5.2 如何最佳化？
 - 2.2.5.3 如何驗證？
- 2.3 綜合驗證方法
- 2.4 小結
- 3 逾時控制最佳化
 - 3.1 逾時設定不當引發的效能瓶頸
 - 3.2 用戶端相關逾時 (Client-side Timeouts)
 - 3.2.1  最佳化建議：
 - 3.2.2  驗證方式：
 - 3.3 上游服務相關逾時 (Upstream Timeouts)
 - 3.3.1  最佳化建議 (以 API 閘道器為例)：
 - 3.3.2  驗證方式：
- 4 長連線設定指南
 - 4.1 用戶端 ↔ ALB：啟用 HTTP Keep-Alive
 - 4.2 ALB ↔ 上游：啟用 Upstream Keep-Alive (關鍵！)
 - 4.3 後端服務：需支援 HTTP/1.1 並保持連線
 - 4.4 綜合驗證：如何確認長連線與逾時設定生效？
- 5 典型場景最佳化案例
 - 5.1 場景一：Web網站存取加速最佳化
 - 5.1.1 業務場景描述
 - 5.1.2 最佳化目標
 - 5.1.3 最佳化方案
 - 5.1.4 最佳化效果
 - 5.2 場景二：API閘道器效能最佳化
 - 5.2.1 業務場景描述
 - 5.2.2 最佳化目標

- 5.2.3 最佳化方案
- 5.3 場景三：檔案下載服務最佳化
 - 5.3.1 業務場景描述
 - 5.3.2 最佳化目標
 - 5.3.3 最佳化方案
 - 5.3.4 最佳化效果
- 5.4 場景四：ALB反向代理AAS服務效能最佳化
 - 5.4.1 作業系統層面調校（Linux）
 - 5.4.2 ALB 設定最佳化
 - 5.4.3 壓測驗證
- 5.5 場景五：ALB 百萬級並行連線測試（C1000K）
 - 5.5.1 背景
 - 5.5.1.1  典型業務場景：
 - 5.5.1.2  重要前提：
 - 5.5.1.3  核心難點：
 - 5.5.2 解決方案
 - 5.5.3 作業系統全面調校（Linux）
 - 5.5.3.1 檔案描述符（FD）——最關鍵！
 - 5.5.3.2 網路堆疊調校
 - 5.5.3.3 核心與硬體建議
 - 5.5.4 ALB設定最佳化
 - 5.5.5 後端最佳化
 - 5.5.6 常見問題與規避
 - 5.5.7 總結
- 6 效能測試與問題排查
 - 6.1 診斷命令彙整
 - 6.2 錯誤現象：壓測時大量請求返回 502 Bad Gateway
 - 6.2.1  錯誤分析
 - 6.2.2  解決方案
 - 6.2.3  驗證方法
 - 6.3 錯誤現象：壓測初期正常，隨後 QPS 驟降，連線堆積
 - 6.3.1  錯誤分析
 - 6.3.2  解決方案
 - 6.3.3  驗證方法

- 6.4 錯誤現象：壓測時出現大量 504 Gateway Timeout
 - 6.4.1  錯誤分析
 - 6.4.2  解決方案
 - 6.4.3  驗證方法
- 6.5 錯誤現象：壓測時用戶端報 「Connection reset by peer」 或 「Connection closed」
 - 6.5.1  錯誤分析
 - 6.5.2  解決方案
 - 6.5.3  驗證方法
- 6.6 錯誤現象：啟用 upstream keepalive 後，QPS 無提升甚至下降
 - 6.6.1  錯誤分析
 - 6.6.2  解決方案
 - 6.6.3  驗證方法
- 6.7 錯誤現象：壓測時系統 CPU sys%（核心態）過高，達 70%+
 - 6.7.1  錯誤分析
 - 6.7.2  解決方案
 - 6.7.3  驗證方法
- 6.8 錯誤現象：多核 CPU 下，僅 CPU0 負載 100%，其他核閒置
 - 6.8.1  錯誤分析
 - 6.8.2  解決方案
 - 6.8.3  驗證方法
- 7 效能測試錯誤速查表
- 8 效能最佳化關鍵參數清單
 - 8.1 全域與行程控制
 - 8.2 事件模型 (Events)
 - 8.3 HTTP 全域設定
 - 8.4 用戶端連線與逾時
 - 8.5 反向代理 (Proxy)
 - 8.6 Upstream 長連線池
 - 8.7 日誌

1 前言

本手冊旨在為金蝶Apusic負載均衡器V2.0.6標準版使用者提供全面、系統的效能最佳化指引。透過深入分析作業系統層面與ALB自身配置兩個維度的效能瓶頸，結合典型業務情境的最佳化案例，協助使用者解決在高並行環境下可能遇到的效能問題。

手冊內容涵蓋了從基礎的系統參數調校到複雜的百萬級並行連線最佳化，從常見的逾時配置到關鍵的長連線複用策略，以及在效能測試過程中可能遇到的各種問題排查方法。透過本手冊的指引，使用者可以：

- 識別和解決ALB在高並行場景下的效能瓶頸
- 合理配置作業系統與ALB參數以提升系統整體效能
- 掌握長連線最佳化策略，有效降低系統資源消耗
- 快速定位和解決效能測試中遇到的常見問題
- 根據具體業務場景實施相應的最佳化方案

1.1 適用對象

本手冊主要面向金蝶Apusic負載均衡器的開發人員、運維人員及相關管理人員，適用於Linux系統環境（包括Kylin、UOS、OpenEuler、CentOS、Ubuntu等發行版）。透過系統地學習和應用本手冊中的最佳化建議，使用者可以顯著提升ALB的效能表現，確保系統在高負載情況下依然保持穩定、高效的服務能力。

1.2 相關手冊

了解更多ALB V2.0.6產品相關資訊，請參閱以下ALB V2.0.6產品手冊文件集：

序號	手冊文件	說明
1	金蝶Apusic負載均衡器軟體 V2.0.6 發版說明	介紹了ALB 產品版本更新內容。
2	金蝶Apusic負載均衡器軟體 V2.0.6 快速使用手冊	簡單介紹了如何快速上手使用ALB。
3	金蝶Apusic負載均衡器軟體 V2.0.6 安裝手冊	詳細介紹如何在各作業系統上安裝ALB，以及ALB服務啟停操作，產品的註冊過程。
4	金蝶Apusic負載均衡器軟體 V2.0.6 代理功能手冊	詳細介紹 ALB 相關功能的使用、配置、管理及配套工具的使用方法。
5	金蝶Apusic負載均衡器軟體 V2.0.6 管控制台使用者手冊	詳細介紹ALB管控制台相關功能的使用和操作說明。

6	金蝶Apusic負載均衡器軟體 V2.0.6 開發手冊	詳細介紹ALB外部API、alb-cli命令列工具及AI Agent整合方法。
7	金蝶Apusic負載均衡器軟體 V2.0.6 遷移手冊	詳細介紹ALB歷史版本遷移升級到V2.0.6版本的說明。
8	金蝶Apusic負載均衡器軟體 V2.0.6 維運手冊	詳細介紹ALB的監控、運維、安全加固等運維說明。
9	金蝶Apusic負載均衡器軟體 V2.0.6 效能最佳化手冊	詳細介紹ALB效能調校的說明。

1.3 技術支援

ALB產品提供全面的技術支援服務，您可以透過以下方式獲得技術支援：

- 網址：www.apusic.com
- 電話：400-855-5800
- 郵箱：support@apusic.com
- 金蝶雲社區：<https://vip.kingdee.com/?productId=73&productLineId=14&lang=zh-CN>

您在取得技術支援時，請提供如下資訊：

1. 您的姓名
2. 公司資訊與聯絡方式
3. 作業系統及其版本
4. 產品版本號
5. 出現異常及錯誤的日誌、截圖等詳細資訊

2 效能瓶頸來源分析（作業系統 + ALB雙維度詳解）

ALB雖然以高效能著稱，但在高並行、大流量或複雜業務場景下，其實際表現往往受限於兩個關鍵層面：作業系統資源排程策略 與 ALB 自身配置合理性。這兩者相互影響，需協同調校。

總體原則：

作業系統提供「舞台」（CPU、記憶體、網路、檔案系統等資源）；

ALB 是「演員」，其表演效果取決於舞台條件與自身編排（配置）。

忽略任一層面，都可能導致效能無法達到預期。

2.1 作業系統層面瓶頸

2.1.1 檔案描述符（FD）限制

2.1.1.1 為什麼存在？

Linux 預設限制每個行程可開啟的檔案描述符數量（通常為 1024），防止資源耗盡。而每個 TCP 連線、靜態檔案控制代碼均佔用一個 FD。

當並行連線數超過限制，ALB會拒絕新連線，並在 `error log` 中記錄 `Too many open files`。

2.1.1.2 如何修改？

```
# 編輯alb啟動腳本，腳本開頭加入以下兩行內容
# 【ALB安裝目錄】/bin/start-alb.sh或start-alb-and-console.sh
ulimit -n 65536
ulimit -Hn 65536

# 若使用 systemd，則在[Service]結尾加入LimitNOFILE=65536
# 編輯 /etc/systemd/system/alb.service
[Service]
....
User=root
Type=forking
LimitNOFILE=65536 # 加入這一行內容
# 修改完後，使用systemctl daemon-reload 然後重啟ALB
```

2.1.1.3 如何驗證？

```
# 查看 worker 行程 FD 限制
cat /proc/$(pgrep alb | tail -1)/limits | grep "open files"

# 即時 FD 使用量
lsof -p $(pgrep alb | tail -1) | wc -l
```

2.1.2 TCP 連線佇列與網路堆疊參數

2.1.2.1 為什麼存在？

核心透過 `somaxconn` (accept 佇列) 和 `tcp_max_syn_backlog` (SYN 佇列) 限制未完成連線的數量。預設值 (如 128) 在高並行場景下極易成為瓶頸，導致新連線被丟棄，表現為用戶端連線逾時。

2.1.2.2 如何修改？

```
# 編輯 /etc/sysctl.conf
net.core.somaxconn = 65535
net.ipv4.tcp_max_syn_backlog = 65535
net.ipv4.tcp_abort_on_overflow = 1 # 選擇性：佇列滿時立即 RST，便於發現問題
```

執行 `sysctl -p` 生效。

同時在 ALB 配置檔 `alb.conf` 的 `server` 區塊內，針對監聽埠的設定加入 `backlog` 參數：

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

說明： `listen` 的 `backlog` 參數與系統 `somaxconn` 共同決定 `accept` 佇列長度，實際生效值為 `min(somaxconn, backlog)`，建議兩者保持一致。

2.1.2.3 如何驗證？

```
ss -lnt # 查看 Recv-Q (目前 accept 佇列長度)，若持續接近 backlog 值則需擴容
```

2.1.3 TIME_WAIT 連線堆積

2.1.3.1 為什麼存在？

主動關閉連線的一方進入 TIME_WAIT 狀態（約 60 秒），確保網路中殘餘封包不會干擾新連線。在短連線高頻請求場景（如行動端 API），大量 TIME_WAIT 會耗盡本機連接埠（ephemeral port range），導致 `Cannot assign requested address`。

2.1.3.2 如何緩解？

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

修改完使用 `sysctl -p` 生效。⚠️ 禁用 `tcp_tw_recycle`（已棄用且 NAT 下不安全）。

2.1.3.3 如何驗證？

```
# 統計 TIME-WAIT 連線數
ss -ant | awk '/TIME-WAIT/ {count++} END {print count+0}'

# 查看 total connections 和 timewait 數量
ss -s
```

2.1.4 CPU 與中斷親和性

2.1.4.1 為什麼存在？

預設網卡中斷集中在 CPU0，導致單核過載，而其他核心閒置。alb多 worker 無法均衡利用多核。

2.1.4.2 如何最佳化？

啟用 `reuseport`（推薦）：

```
listen 80 reuseport;
```

核心自動將新連線分發到各 worker 的獨立 socket，實現負載平衡。或手動綁定 IRQ 到不同 CPU（進階）。

2.1.4.3 如何驗證？

```
top # 觀察各 CPU 核心使用率是否均衡
```

2.2 ALB 配置瓶頸

2.2.1 Worker 行程與連線數配置不當

2.2.1.1 為什麼存在？

- worker_processes 設定過少（如 1），無法利用多核；
- worker_connections 設定過低，單 worker 無法處理足夠並行；
- 未設定 worker_rlimit_nofile，導致即使系統允許，ALB 仍受內部限制。

2.2.1.2 如何最佳化？

```
worker_processes auto;           # 自動匹配 CPU 核數
worker_rlimit_nofile 131072;    # 提升內部 FD 限制，必須 ≥
worker_connections × 2

events {
    use epoll;                   # Linux 必須啟用
    worker_connections 65536;    # 單 worker 最大連線數
    multi_accept on;             # 一次 accept 多個連線
}
```

2.2.1.3 如何驗證？

- 理論最大並行 $\approx \text{worker_processes} \times \text{worker_connections} / 2$ （每個反向代理連線需要 client 和 upstream 兩個 socket）
- 結合壓測觀察是否達到預期 QPS

2.2.2 緩衝區 (Buffer) 配置不合理

2.2.2.1 為什麼存在？

- proxy_buffering off 會導致後端回應直接流式轉發，增加 client 與 backend 的耦合，降低吞吐；
- Buffer 太小（如預設 4k/8k）在大回應體場景下頻繁 syscall，增加 CPU 開銷；
- Buffer 太大會浪費記憶體。

2.2.2.2 如何最佳化？

```
proxy_buffering on;
proxy_buffer_size 16k;
proxy_buffers 8 32k;           # 總緩衝 ≈ 256k，按業務調整
proxy_busy_buffers_size 64k;
```

2.2.2.3 如何驗證？

- 對比開啟/關閉 buffering 的 QPS 與延遲；
- 使用 `strace -p` 觀察 read/write 系統呼叫頻率。

2.2.3 日誌寫入頻繁導致 I/O 瓶頸

2.2.3.1 為什麼存在？

預設每請求寫一行 access log，高並行下磁碟 I/O 成為瓶頸，尤其在機械硬碟或雲端硬碟 IOPS 受限時。如何最佳化？

```
# 方案1：關閉非必要日誌（生產慎用）
access_log off;

# 方案2：啟用緩衝寫入
access_log logs/access.log main buffer=64k flush=5s;
```

2.2.3.2 如何驗證？

```
iostat -x 1 # 觀察 %util 和 await 是否下降
```

2.2.4 SSL/TLS 交握開銷大

2.2.4.1 為什麼存在？

每次 HTTPS 新連線需完整 TLS 交握（2-RTT），計算密集（尤其 RSA 金鑰交換），消耗大量 CPU。

2.2.4.2 如何最佳化？

```
ssl_session_cache shared:SSL:10m; # 多 worker 共享 session
ssl_session_timeout 10m;
ssl_session_tickets on; # 啟用 ticket 快速恢復（注意 key 輪換）
ssl_protocols TLSv1.2 TLSv1.3;
ssl_prefer_server_ciphers off; # 讓用戶端優先選擇高效 cipher
keepalive_timeout 75s; # 複用 TCP+TLS 連線
```

2.2.4.3 如何驗證？

- 使用 `openssl s_client -connect host:443 -reconnect` 測試 session 複用；

- 壓測對比 HTTPS 與 HTTP 的 QPS 差距是否縮小；
- 監控 CPU 使用率（top 中 %sys 是否過高）。

2.2.5 靜態資源未啟用高效傳輸機制

2.2.5.1 為什麼存在？

未開啟 sendfile、tcp_nopush 等，導致核心多次拷貝資料，增加延遲。

2.2.5.2 如何最佳化？

```
sendfile on;                # 零拷貝傳輸檔案
tcp_nopush on;              # 配合 sendfile, 累積資料到完整包後再發送, 減少包頭數
                              量, 提升大檔案吞吐
tcp_nodelay on;             # 在 sendfile 完成後或動態內容場景下, 禁用 Nagle 演
                              算法, 立即發送小包, 降低延遲
open_file_cache max=10000 inactive=60s; # 快取檔案元資料
```

說明：tcp_nopush 與 tcp_nodelay 作用場景不同，同時開啟不衝突。tcp_nopush 主要在 sendfile 發送大檔案時生效；tcp_nodelay 在動態內容或 sendfile 結束後的小包場景下生效。兩者配合可同時最佳化吞吐與延遲。

2.2.5.3 如何驗證？

- 對比開啟前後靜態檔案下載速度；
- 使用 perf 或 bpftrace 觀察 page fault 和 copy 次數。

2.3 綜合驗證方法

完成上述任一最佳化後，必須透過以下方式驗證效果：

驗證目標	工具/命令	關注指標
並行連線能力	wrk -c 10000 -t 10 -d 30s URL	Requests/sec, Latency
連線佇列狀態	ss -lnt	Recv-Q 是否接近 backlog
FD 使用情況	`lsof -p PID	wc -l`
CPU 利用均衡性	top 或 htop	各核負載是否均勻
網路錯誤	`netstat -s	grep -i drop`
ALB 內部狀態	啟用 stub_status	active/accepts/handled/requests

磁碟 I/O 壓力	<code>iostat -x 1</code>	%util, await
-----------	--------------------------	--------------

2.4 小結

維度	典型瓶頸	最佳化方向
作業系統	FD 限制、TCP 佇列、TIME_WAIT	調整 limits、sysctl 參數
ALB 配置	Worker 數、Buffer、日誌、SSL	合理設定 events/http 模組

3 逾時控制最佳化

3.1 逾時設定不當引發的效能瓶頸

ALB 涉及多個逾時參數，分別作用於用戶端 ↔ ALB 與 ALB ↔ 上游服務兩個方向。若配置不合理，會導致：

- 連線無法及時釋放 → `worker_connections` 耗盡；
- 請求卡住不返回 → 用戶端逾時、使用者體驗差；
- 後端異常時 ALB 積壓大量 pending 請求 → 記憶體/CPU 耗盡。

3.2 用戶端相關逾時 (Client-side Timeouts)

參數	預設值	作用	風險
<code>client_header_timeout</code>	60s	讀取用戶端請求標頭的逾時	慢用戶端攻擊 (Slowloris) 可耗盡連線
<code>client_body_timeout</code>	60s	讀取用戶端請求內體的逾時	大檔案上傳未限速時卡住 worker
<code>keepalive_timeout</code>	75s	用戶端 keep-alive 連線保持時間	設定過大會導致 idle 連線堆積

3.2.1 最佳化建議：

```
# 根據業務調整, API 場景可縮短
client_header_timeout 10s;
client_body_timeout 10s;

# 合理設定 keepalive, 兼顧複用與資源回收
keepalive_timeout 30s;           # 保持 30 秒
keepalive_requests 1000;        # 單連線最多處理 1000 個請求 (防記憶體洩漏)
```

3.2.2 驗證方式：

- 模擬慢請求 (如使用 curl 限制速度發送大檔案 slow.txt[10MB]: `curl --limit-rate 1k --data-binary @slow.txt -X POST http://ip:port/api`) ；
- 觀察 error log 是否出現 `client timed out` ；

- 使用 `ss -o state established` 查看連線 ESTAB 時間是否超過預期。

3.3 上游服務相關逾時 (Upstream Timeouts)

參數	預設值	作用	風險
<code>proxy_connect_timeout</code>	60s	與 upstream 建立 TCP 連線的逾時	後端宕機時阻塞 worker
<code>proxy_send_timeout</code>	60s	向 upstream 發送請求的逾時	網路壅塞或後端接收慢
<code>proxy_read_timeout</code>	60s	從 upstream 讀取回應的逾時	後端處理慢或 hang 死

⚠ 預設 60s 在高並行場景下極其危險！一個慢介面可能導致數千連線被佔用。

3.3.1 最佳化建議 (以 API 閘道器為例)：

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

    # 連線逾時：內網健康後端通常 1~3s 足夠，若後端跨機房或網路品質差，可放寬至 10s
    proxy_connect_timeout 10s;

    # 發送/讀取逾時：API 場景建議根據業務 P99 設定 (如 500ms ~ 5s)；大檔案上傳或高並行耗時業務場景可放寬至 60s ~ 120s
    proxy_send_timeout 60s;
    proxy_read_timeout 60s;

    # 其他必要設定
    proxy_http_version 1.1;
    proxy_set_header Connection "";
}
```

3.3.2 驗證方式：

- 將 `proxy_read_timeout` 設為 5s，然後讓後端 sleep 10s，觀察 ALB 是否在 5s 後返回 504 Gateway Timeout；
- 監控 `alb_status` 中 active connections 是否在異常時飆升；
- 日誌中應出現 `upstream timed out (110: Connection timed out)`。

最佳實務：

逾時時間應 略大於後端 P99 回應時間，並配合熔斷/降級機制

4 長連線設定指南

長連線需在三個環節協同設定才能生效：

[用戶端] ←(HTTP Keep-Alive)→ [ALB] ←(Upstream Keep-Alive)→ [上游服務]

4.1 用戶端 ↔ ALB：啟用 HTTP Keep-Alive

ALB 預設支援用戶端 keep-alive，但需合理設定：

```
http {
    # 全域 keepalive 設定
    keepalive_timeout 30s;           # 連線保持 30 秒
    keepalive_requests 1000;        # 單連線最多處理 1000 個請求 (防記憶體洩漏)

    # 對 HTTP/2 自動啟用多路複用，無需額外設定
}
```

✔ 適用於 Web 瀏覽器、行動端 App、內部服務呼叫等支援 keep-alive 的用戶端。

4.2 ALB ↔ 上游：啟用 Upstream Keep-Alive（關鍵！）

已在前文詳述，此處整合為完整範本：

```
upstream backend {
    server 192.168.1.10:8080;
    server 192.168.1.11:8080;
    keepalive 64; # 每個 worker 快取 64 個空閒連線
}

server {
    location /api/ {
        proxy_pass http://backend;
        proxy_http_version 1.1;
    }
}
```

```

proxy_set_header Connection ""; # 清除 Connection 標頭

# 逾時控制（必須配合長連線）
proxy_connect_timeout 3s;
proxy_send_timeout 120s;
proxy_read_timeout 120s;
}
}

```

4.3 後端服務：需支援 HTTP/1.1 並保持連線

- 後端框架（如 Spring Boot、Gin）預設通常支援；Flask 等框架需確認設定並啟用多執行緒/多行程模式；
- 確保未主動發送 Connection: close；
- 若使用 TLS，建議啟用 session resumption。

4.4 綜合驗證：如何確認長連線與逾時設定生效？

驗證目標	方法
用戶端 keep-alive	用瀏覽器開發者工具或 <code>curl -v</code> 查看是否複用連線；或用 <code>tcpdump</code> 觀察同一 TCP 流中多個 HTTP 請求
upstream keep-alive	<code>ss -tnp grep alb grep :8080</code> 查看 ESTAB 連線是否穩定存在
逾時生效	模擬後端延遲 > timeout，確認 ALB 返回 504 而非 hang 住
連線複用效果	壓測對比開啟/關閉 keepalive 的 QPS 與 CPU 使用率

範例壓測命令（驗證 upstream keepalive）：

```

# 高並行短請求
wrk -t16 -c1000 -d30s --latency http://alb/api/fast-endpoint

# 觀察指標：
# - Requests/sec 提升
# - ss -s 中 timewait 數量顯著下降
# - 後端 accept() 次數減少

```

5 典型場景最佳化案例

5.1 場景一： Web網站存取加速最佳化

5.1.1 業務場景描述

某企業部署了一個入口網站，包含大量靜態資源（圖片、CSS、JS檔案），存取量逐漸增加，使用者反映頁面載入較慢。

5.1.2 最佳化目標

提升靜態資源存取速度，降低伺服器負載，改善使用者體驗。

5.1.3 最佳化方案

- 啟用靜態資源快取

```
http {  
    # 開啟檔案快取  
    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;  
  
        # 針對靜態資源設定快取  
        location ~* \.(jpg|jpeg|png|gif|ico|css|js)$ {  
            expires 1y;  
            add_header Cache-Control "public, immutable";  
        }  
    }  
}
```

- 啟用gzip壓縮

```
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 最佳化效果

- 靜態資源載入速度提升50%以上
- 伺服器CPU使用率降低20%
- 使用者頁面載入時間顯著縮短

5.2 場景二： API閘道器效能最佳化

5.2.1 業務場景描述

某公司API閘道器需要處理大量行動端請求，高峰期出現回應慢、部分請求失敗的情況。

5.2.2 最佳化目標

提升API閘道器並行處理能力，保證服務穩定性。

5.2.3 最佳化方案

- 1. 調整系統檔案描述符限制

```
# 編輯alb啟動腳本，腳本開頭加入以下兩行內容
# 【ALB安裝目錄】/bin/start-alb.sh或start-alb-and-console.sh
ulimit -n 65536
ulimit -Hn 65536
```

- 2. 最佳化ALB設定

```
worker_processes auto;
worker_rlimit_nofile 65536;
```

```

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

http {
    # 啟用長連線複用

    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;
    }

    # 限制單IP請求頻率 (防止惡意刷介面)
    limit_req_zone $binary_remote_addr zone=api:10m rate=10r/s;

    server {
        listen 80 reuseport;

        # API路由
        location /api/ {
            proxy_pass http://api_backend;
            # 啟用 HTTP/1.1 長連線
            proxy_http_version 1.1;
            proxy_set_header Connection "";

            # 合理設定逾時
            proxy_connect_timeout 5s;
            proxy_send_timeout 120s;
            proxy_read_timeout 120s;

            # 限制單IP請求頻率 (防止惡意刷介面)

```

```

        limit_req zone=api burst=20 nodelay;
    }
}
}

```

5.3 場景三：檔案下載服務最佳化

5.3.1 業務場景描述

企業需要提供大檔案下載服務，使用者反映下載速度慢，伺服器資源佔用高。

5.3.2 最佳化目標

提升大檔案下載速度，降低伺服器資源消耗。

5.3.3 最佳化方案

- 1. 啟用大檔案最佳化設定

```

http {
    # 調整緩衝區大小（適用於反向代理大檔案下載場景）
    proxy_buffering on;
    proxy_buffer_size 128k;
    proxy_buffers 4 256k;
    proxy_busy_buffers_size 256k;

    # 啟用分片下載支援
    proxy_http_version 1.1;

    server {
        listen 80;

        # 靜態檔案直接下載（使用 root, 無需 proxy_* 參數）
        location /downloads/ {
            root /data;

```

```

# 支援斷點續傳
add_header Accept-Ranges bytes;

# 大檔案下載專用設定
location ~* \.(iso|zip|exe)$ {
    # 限制下載速度（防止頻寬被佔滿）
    limit_rate 2m; # 限制為2MB/s

    # 啟用零拷貝傳輸，提升靜態檔案下載效能
    sendfile on;
    tcp_nopush on;
}
}
}
}

```

5.3.4 最佳化效果

- 大檔案下載速度提升60%
- 伺服器記憶體佔用降低30%
- 支援斷點續傳，改善使用者體驗

5.4 場景四：ALB反向代理AAS服務效能最佳化

目標：透過作業系統調校 + ALB 設定最佳化，使 ALB 能高效代理後端 AAS 服務，支撐高並行 API 請求。

5.4.1 作業系統層面調校 (Linux)

提升檔案描述限制

```

# 編輯alb啟動腳本，腳本開頭加入以下兩行內容
# 【ALB安裝目錄】/bin/start-alb.sh或start-alb-and-console.sh
ulimit -n 100000
ulimit -Hn 100000

```

最佳化TCP網路堆疊

```
# 編輯/etc/sysctl.conf, 加入以下內容:
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

# 生效
sysctl -p
```

AAS 效能最佳化：確保 AAS 應用伺服器開啟長連線支援，並根據實際業務調整執行緒池和連線池大小。詳細最佳化請參考 AAS 產品效能調校文件。

5.4.2 ALB 設定最佳化

```
user root;
worker_processes auto;
worker_rlimit_nofile 131072;      # 必須 ≥ worker_connections × 2

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

http {
    # 全域逾時控制（作用於用戶端 ↔ ALB）
    client_header_timeout 120s;    # 大檔案上傳場景適當放寬
    client_body_timeout 120s;
    keepalive_timeout 300s;        # 用戶端 keep-alive 連線保持時間
    keepalive_requests 1000;       # 單連線最多處理 1000 個請求

    upstream aas_backend {
        server 192.168.1.100:8080 max_fails=3 fail_timeout=10s;
```

求

後端)

{

```

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;           # 每個 worker 保持 128 個空閒長連線
keepalive_requests 1000; # 單條 upstream 連線最多處理 1000 個請

    keepalive_timeout 75s; # upstream 空閒連線逾時 (作用於 ALB ↔

}

server {
    listen 8080 reuseport;

    location / {
        proxy_pass http://aas_backend/;

        # 啟用與 AAS 的長連線
        proxy_http_version 1.1;
        proxy_set_header Connection "";

        # 逾時設定 (略大於 AAS 處理時間)
        proxy_connect_timeout 10s;
        proxy_send_timeout 120s;
        proxy_read_timeout 120s;

        # 傳遞用戶端資訊
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For
$proxy_add_x_forwarded_for;

    }

    # 日誌緩衝寫入
    access_log access.log main buffer=64k flush=5s;

```

```
}
}
```

5.4.3 壓測驗證

```
# 使用 wrk 模擬高並行請求
wrk -t16 -c2000 -d60s --latency http://your-alb-server:8080/api/test

# 觀察指標：
# - Requests/sec 是否穩定提升
# - ALB error log 無 "too many open files"
# - ss -tnp | grep :8080 查看是否維持 ESTAB 連線 (非大量 TIME-WAIT)
```

5.5 場景五：ALB 百萬級並行連線測試 (C1000K)

5.5.1 背景

5.5.1.1 典型業務場景：

- WebSocket 長連線閘道器（如聊天、IoT 設備接入）
- HTTP 長輪詢（Long Polling）服務
- Server-Sent Events (SSE) 推送平台
- 微服務 API 閘道器（高保活連線）

5.5.1.2 重要前提：

百萬並行 $\neq$ 百萬 QPS。此處指 同時維持 1,000,000 個 TCP 連線，每個連線可能低頻通訊（如每分鐘一次心跳），但必須保持 alive。

5.5.1.3 核心難點：

ALB 需為每個連線維護兩個 socket：client $\leftrightarrow$ alb + alb $\leftrightarrow$ backend $\rightarrow$ 總 FD 數 $\approx 2 \times$ 並行連線數 若未啟用 upstream keepalive，後端需承受百萬級連線壓力（幾乎不可行）必須透過 連線複用（長連線池）將後端連線數壓縮到合理範圍

5.5.2 解決方案

- ALB 層：承擔百萬連線接入、TLS 終止、路由分發
- Backend 層：僅需處理 活躍請求，透過 小規模長連線池 與 ALB 通訊
- 關鍵策略：前端海量連線 + 後端有限連線池

 理想比例：

1,000,000 用戶端連線 → ALB → 1,000 ~ 10,000 個 upstream 長連線 (複用率 100:1 ~ 1000:1)

5.5.3 作業系統全面調校 (Linux)

5.5.3.1 檔案描述符 (FD) ——最關鍵!

```
# 編輯alb啟動腳本，腳本開頭加入以下兩行內容
# 【ALB安裝目錄】/bin/start-alb.sh或start-alb-and-console.sh
ulimit -n 150000
ulimit -Hn 150000

# 全域檔案控制代碼上限
echo 'fs.file-max = 2500000' >> /etc/sysctl.conf
echo 'fs.nr_open = 2500000' >> /etc/sysctl.conf
```

 ulimit 設定 15 萬，全域設定 250 萬？

ALB 百萬連線最低連線數： $\text{百萬連線} \times 2 (\text{client} + \text{upstream}) = 200 \text{ 萬}$ 。

ALB worker 數：ulimit 設定的是每個行程開啟數量為 15 萬，那麼 $200 \text{ 萬} / 15 \text{ 萬} = 14 \text{ worker}$ 行程數（即建議 16 核及以上 CPU）。

設定 250 萬：除 ALB 需要用到的 200 萬檔案數外，系統自身和其他服務也需要開啟檔案，即 250 萬開啟檔案數是整個系統檔案控制代碼數的上限。

5.5.3.2 網路堆疊調校

```
# /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 最佳化 (ALB 主動關閉 client 連線時產生)
net.ipv4.tcp_tw_reuse = 1
net.ipv4.tcp_fin_timeout = 10
```

```
net.ipv4.tcp_max_tw_buckets = 2000000

# 記憶體緩衝區 (支援大並行)
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

# 連接埠範圍 (若 ALB 也作為用戶端發起連線)
net.ipv4.ip_local_port_range = 1024 65535
```

修改完1和2，執行下面命令生效：

```
sysctl -p
```

5.5.3.3 核心與硬體建議

- Linux 核心 ≥ 4.5 (穩定支援 reuseport + epoll)
- CPU ≥ 16 核 (百萬連線內容切換開銷大)
- 記憶體 $\geq 64\text{GB}$ (每個連線約 4~8KB，百萬連線約 8GB，加上 buffer 和後端快取)
- 關閉 swap: `swapoff -a`
- 使用 SSD：減少日誌或臨時檔案 I/O 影響

5.5.4 ALB設定最佳化

```
user root;
worker_processes auto;                # 匹配 CPU 核數 (如 16)
worker_rlimit_nofile 300000;          # 單個 worker 開啟檔案數，必須  $\geq$ 
worker_connections  $\times 2$ 

error_log error.log warn;

events {
    use epoll;
    worker_connections 150000;         # 單 worker 支援 15 萬連線 (16  $\times$ 
```

```

15 萬 = 240 萬)
    multi_accept on;
}

http {
    # 關閉 access log (壓測階段) , 或使用非同步緩衝
    access_log off;
    error_log error.log crit;

    # 高效傳輸
    sendfile on;
    tcp_nopush on;
    tcp_nodelay on;

    # 用戶端 keep-alive (維持長連線)
    keepalive_timeout 300s;           # 5 分鐘無操作斷開
    keepalive_requests 1000000;      # 防止單連線記憶體洩漏

    # Upstream 長連線池 (核心! )
    upstream backend_pool {
        # 假設 10 台後端, 每台 max_conns=1000, 總後端連線上限 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;

    # 每個 worker 保持 625 個空閒長連線 (16 workers × 625 = 10,000,
與總 max_conns 匹配)
    keepalive 625;
}

server {
    listen 8080 reuseport backlog=1000000;

    location / {
        proxy_pass http://backend_pool/;

        # 啟用 upstream 長連線 (必須! )
        proxy_http_version 1.1;
        proxy_set_header Connection "";

        # 逾時設定 (根據業務心跳間隔)
        proxy_connect_timeout 15s;
        proxy_send_timeout 300s;      # 根據請求體大小和網路狀況設定
        proxy_read_timeout 300s;      # 根據後端業務處理時間設定, 應略大
於後端 P99 回應時間

        # 傳遞必要標頭
        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;
    }
}

```

```
}
}
```

✓ 關鍵點說明：

- *keepalive 1000*: 每個 worker 維護 1000 個 upstream 空間連線；
- *max_conns=1000*: 限制單後端最大連線數，防過載；
- *proxy_send/read_timeout ≥ keepalive_timeout*: 避免 ALB 提前關閉連線；
- *reuseport*: 實現核心級負載平衡，避免 accept 鎖競爭。

5.5.5 後端最佳化

- 支援長連線

5.5.6 常見問題與規避

問題	原因	解決方案
ALB 報 "too many open files"	FD 限制不足	提升 LimitNOFILE + worker_rlimit_nofile
後端連線數暴增	未啟用 upstream keepalive	檢查 proxy_http_version 1.1 + Connection ""
連線頻繁斷開	逾時設定不匹配	確保 proxy_read_timeout ≥ keepalive_timeout
CPU 單核 100%	未啟用 reuseport	加入 listen ... reuseport;
TLS 交握慢	未複用 session	啟用 ssl_session_cache + ssl_session_tickets

5.5.7 總結

總結：反向代理百萬並行的核心公式

百萬並行 = 作業系統資源 × ALB 連線管理 × upstream 長連線複用 × 後端非同步能力

層級	關鍵動作
OS	提升 FD、最佳化 TCP、關閉 swap
ALB	reuseport + 大 worker_connections + upstream keepalive
Backend	非同步模型 + 長連線支援 + 狀態外置
驗證	壓測 + 監控 + 日誌分析

6 效能測試與問題排查

6.1 診斷命令彙整

```
# 查看 ALB 錯誤日誌 (重點關注)
tail -f ALB安裝目錄/logs/error.log

# 查看目前連線狀態
ss -s

# 查看 ALB 行程 FD 使用
lsof -p $(pgrep alb | head -1) | wc -l

# 查看系統 FD 限制
ulimit -n
cat /proc/sys/fs/file-max

# 查看 TCP 佇列溢位 (關鍵! )
netstat -s | grep -i "listen.*overflows\|drop"

# 查看 CPU 核心負載
top

# 查看 ALB 版本
ALB安裝目錄/sbin/alb -V
```

6.2 錯誤現象：壓測時大量請求返回 502 Bad Gateway

6.2.1 🔍 錯誤分析

ALB 無法與 upstream 後端建立連線或通訊失敗；

常見原因：

- 後端服務宕機或連接埠未監聽；
- proxy_connect_timeout 過短，後端啟動慢；

- 後端連線數達上限（如同 Tomcat maxConnections 耗盡）；
- DNS 解析失敗（若 upstream 使用域名）。

6.2.2 解決方案

```
location / {
    proxy_pass http://backend;
    proxy_connect_timeout 15s;    # 根據後端啟動時間調整
    proxy_read_timeout 30s;
    resolver 8.8.8.8 valid=30s; # 若用域名，顯式指定 DNS
}
```

6.2.3 驗證方法

- 查看 ALB error log:

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

- 手動 curl 測試 upstream 是否可達；
- 壓測時監控後端 accept 佇列：ss -lnt 查看 Recv-Q 是否溢位。

6.3 錯誤現象：壓測初期正常，隨後 QPS 驟降，連線堆積

6.3.1 錯誤分析

- ALB worker_connections 或系統檔案描述符（FD）耗盡；
- 日誌中出現 accept() failed (24: Too many open files)；
- 原因：預設 worker_connections=1024 + 系統 ulimit -n=1024 無法支撐高並行。

6.3.2 解決方案

- 作業系統層面：

```
# 編輯alb啟動腳本，腳本開頭加入以下兩行內容
# 【ALB安裝目錄】/bin/start-alb.sh或start-alb-and-console.sh
ulimit -n 100000
ulimit -Hn 100000
```

```
# 若使用 systemd, 則在[Service]結尾加入LimitNOFILE=100000
# 編輯 /etc/systemd/system/alb.service
[Service]
....
User=root
Type=forking
LimitNOFILE=100000 # 加入這一行內容
# 修改完後, 使用systemctl daemon-reload 然後重啟ALB
```

ALB設定

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

6.3.3 驗證方法

- 檢查行程 FD 限制：

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

- 即時監控 FD 使用量：

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

- 壓測時觀察 error log 是否仍有 「Too many open files」。

6.4 錯誤現象：壓測時出現大量 504 Gateway Timeout

6.4.1 錯誤分析

- ALB 等待 upstream 回應逾時；
- 預設 proxy_read_timeout=60s, 但後端處理慢或 hang 死；
- 在高並行下, 少量慢請求即可拖垮整個 worker。

6.4.2 解決方案

- 合理設定逾時（根據業務 P99 回應時間）：

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

- 後端最佳化：引入非同步處理、熔斷機制；
- 避免長尾請求：透過限流（limit_req）保護後端。

6.4.3 驗證方法

- 將 `proxy_read_timeout` 設為 5s，然後讓後端 sleep 10s，確認 ALB 是否在 5s 後返回 504；
- 查看error log：

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

6.5 錯誤現象：壓測時用戶端報「Connection reset by peer」或「Connection closed」

6.5.1 錯誤分析

- ALB主動關閉了連線，常見於：
 - 用戶端發送了不完整請求（如只發 header 不發 body）；
 - client_body_timeout 或 client_header_timeout 觸發；
 - 後端返回非法回應（如 chunked 編碼錯誤）。

6.5.2 解決方案

- 調整用戶端行為（確保完整 HTTP 請求）；
- 適當放寬逾時（僅限可信內網）：

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

- 啟用更詳細的日誌定位問題：

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

6.5.3 驗證方法

- 使用 tcpdump 抓包分析 RST 包來源；
- 對比開啟/關閉 debug log 的錯誤詳情。

6.6 錯誤現象：啟用 upstream keepalive 後，QPS 無提升甚至下降

6.6.1 錯誤分析

- 配置遺漏：未設定 proxy_http_version 1.1 或未清除 Connection 標頭；
- 後端不支援 keep-alive：返回 Connection: close；
- keepalive 連線池過小：頻繁新建連線，複用率低。

6.6.2 解決方案

```
upstream backend {
    server 192.168.1.10:8080;
    keepalive 128; # 根據 QPS 調整
}

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

6.6.3 驗證方法

- 檢查 upstream 連線狀態：
 - `ss -tnp | grep ':8080' | grep alb | grep ESTAB | wc -l`
 - 若數值穩定 $\approx$ worker_processes $\times$ keepalive，說明生效；
- 壓測對比：開啟前後 QPS 與 TIME-WAIT 數量。

6.7 錯誤現象：壓測時系統 CPU sys%（核心態）過高，達 70%+

6.7.1 錯誤分析

- 頻繁 syscall（如 read/write）導致內容切換開銷大；
- 未啟用 sendfile、tcp_nopush 等零拷貝機制；

- 小包頻繁發送 (Nagle 演算法未最佳化)。

6.7.2 解決方案

```
sendfile on;
tcp_nopush on;    # 累積資料後發送, 減少 packet
tcp_nodelay on;    # 小包立即發 (動態內容適用)
```

- 靜態資源優先使用 `sendfile` ;
- 動態 API 可關閉 `tcp_nopush` , 啟用 `tcp_nodelay` 。

6.7.3 驗證方法

- 使用 `perf top` 觀察核心函數 (如 `tcp_sendmsg`) 佔比是否下降;
- 對比最佳化前後 `sar -u` 中 `%sys` 指標。

6.8 錯誤現象：多核 CPU 下，僅 CPU0 負載 100%，其他核閒置

6.8.1 錯誤分析

- 未啟用 reuseport, 所有連線由單個 worker accept;
- 網卡中斷集中在 CPU0。

6.8.2 解決方案

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

6.8.3 驗證方法

```
top    # 觀察各 CPU 核心負載是否均衡
ss -lnt | grep :80    # 查看是否有多個 listening socket (reuseport 特徵)
```

7 效能測試錯誤速查表

錯誤現象	可能原因	關鍵檢查命令	解決方案
大量 502 Bad Gateway	後端不可達 / 連線拒絕 / DNS 失敗	<code>curl -v http://backend grep "connect.*failed" logs/error.log</code>	檢查後端服務狀態 設定 resolver (若用域名) 調大 <code>proxy_connect_timeout</code>
大量 504 Gateway Timeout	後端回應慢 / hang 死	<code>grep "upstream timed out" logs/error.log `ss -tnp</code>	<code>grep :backend_port`</code>
QPS 上不去, 連線堆積	FD 耗盡 / <code>worker_connections</code> 不足	<code>`cat /proc/\$(pgrep alb)/limits</code>	<code>grep open
lsof -p \$(pgrep alb)</code>
壓測報 "Too many open files"	系統或 ALB檔案描述符限制過低	<code>`dmesg</code>	<code>tail
systemctl show alb</code>
HTTPS QPS 遠低於 HTTP	TLS 交握未複用 / cipher 低效	<code>`openssl s_client -connect host:443 -reconnect 2>/dev/null</code>	<code>grep Reused`</code>
CPU 僅單核 100%, 其餘閒置	未啟用 <code>reuseport</code> / <code>accept</code> 鎖競爭	<code>top `ss -lnt</code>	<code>grep :80` (看是否多個監聽 socket)</code>
啟用 upstream keepalive 無效	缺少 HTTP/1.1 或 Connection 標頭未清除	<code>`ss -tnp</code>	<code>grep :backend_port</code>
用戶端報 "Connection reset by peer"	逾時觸發 / 請求不完整 / 後端異常關閉	<code>tcpdump -i any -nn port 80 -w debug.pcap grep "client timed out" logs/error.log</code>	調整 <code>client_header_timeout</code> / <code>client_body_timeout</code> 確保壓測工具發送完整 HTTP 請求
TIME-WAIT 連線數暴增 (>10萬)	未複用連線 / 主動關閉方為 ALB	<code>ss -s `netstat -ant</code>	<code>awk '/TIME-WAIT/ {count++} END {print count+0}`</code>
靜態檔案效能差	未啟用零拷貝 / buffer 小	<code>`strace -p \$(pgrep alb) 2>&1</code>	<code>grep -E 'read</code>

8 效能最佳化關鍵參數清單

表格清單說明：

- 指令：ALB的設定指令，指ALB功能設定的參數名。
- 上下文：指該參數的作用域。

作用域識別	對應配置區塊	說明
main	全域配置區域	在配置檔頂層，不包含在任何區塊內的全域參數配置區域
events	事件模型配置區塊	在events{}區塊內設定，用於設定ALB事件處理模型參數
http	HTTP服務配置區塊	在http{}區塊內設定，用於設定HTTP服務和代理相關參數
server	虛擬伺服器配置區塊	在server{}區塊內設定，用於設定HTTP虛擬伺服器參數
location	URL匹配配置區塊	在location{}區塊內設定，用於設定特定URL路徑的處理參數
upstream	上游服務配置區塊	在upstream{}區塊內設定，用於設定反向代理的後端伺服器群組參數
stream	四層代理配置區塊	在stream{}區塊內設定，用於設定TCP/UDP四層代理參數
main/http/server/location	多級配置區域	可在main、http、server、location任意層級中設定
http/server/location	HTTP配置區域	僅可在http、server、location層級中設定

8.1 全域與行程控制

指令	上下文	功能說明	預設值	建議值	注意事項
worker_processes	main	ALB worker 行程數	auto	auto 或 CPU 核數 (如 8)	auto 自動匹配 CPU 核心數；多核必設
worker_rlimit_nofile	main	單個 worker 最大檔案描述符數	無限制 (繼承系統)	65536 ~ 262144	必須 $\geq$ worker_connections $\times$ 2 (client + upstream)

<code>worker_cpu_affinity</code>	main	綁定 worker 到指定 CPU 核	未綁定	auto（或手動遮罩）	減少內容切換，提升快取命中率（選擇性）
----------------------------------	------	---------------------	-----	-------------	---------------------

8.2 事件模型（Events）

指令	上下文	功能說明	預設值	建議值	注意事項
<code>worker_connections</code>	events	單個 worker 最大並行連線數	512	10000 ~ 200000	總並行 $\approx$ worker_processes $\times$ worker_connections
<code>use</code>	events	事件驅動模型	自動選擇	epoll (Linux)	Linux 下必須顯式設為 epoll（雖預設但建議寫明）
<code>multi_accept</code>	events	一次 accept 多個連線	off	on	配合 epoll 提升吞吐，避免飢餓

8.3 HTTP 全域設定

指令	上下文	功能說明	預設值	建議值	注意事項
<code>sendfile</code>	http/server/location	零拷貝傳輸靜態檔案	off	on	靜態資源服務必開，減少核心/使用者空間拷貝
<code>tcp_nopush</code>	http/server/location	累積資料後發送（配合 sendfile）	off	on	提升大檔案傳輸效率，減少小包
<code>tcp_nodelay</code>	http/server/location	禁用 Nagle 演算法，小包立即發	off	on（API 場景）	動態內容/API 建議開啟，降低延遲
<code>open_file_cache</code>	http/server/location	快取檔案元資訊（fd/stat）	off	max=10000 inactive=60s	高頻靜態檔案存取必備

open_file_cache_valid	http/server/location	快取驗證 間隔	60s	30s ~ 120s	配合 open_file_cache 使用
open_file_cache_min_uses	http/server/location	檔案最少 存取次數 才快取	1	2	防止快取冷檔案

8.4 用戶端連線與逾時

指令	上下文	功能說明	預 設 值	建議值	注意事項
keepalive_timeout	http/server/location	用戶端 keep-alive 連線保持 時間	75s	15s ~ 30s (Web API) 300s (長連 線)	過大會導致 idle 連線堆積
keepalive_requests	http/server/location	單連線最 大請求數	100	1000 ~ 1000000	防止單連線記 憶體洩漏
client_header_timeout	http/server/location	讀取請求 標頭逾時	60s	10s ~ 30s 高並行下可 以增大如 120s	防 Slowloris 攻擊
client_body_timeout	http/server/location	讀取請求 內體逾時	60s	10s ~ 120s 檔案上傳可 根據實際時 間設定	大檔案上傳需 調大, API 可 調小
reset_timedout_connection	http/server/location	逾時後發 送 RST 而 非 FIN	off	on	快速釋放 TIME-WAIT (謹慎使用)

8.5 反向代理 (Proxy)

指令	上下文	功能說明	預 設 值	建議值	注意事項
----	-----	------	-------------	-----	------

proxy_http_version	http/server/location	代理使用的 HTTP 版本	1.0	1.1	啟用 upstream keepalive 必須設為 1.1
proxy_set_header Connection ""	location	清除 Connection 標頭	無	""	配合 keepalive 使用，防止 close
proxy_connect_timeout	http/server/location	與 upstream 建連逾時	60s	1s ~ 10s 高並行下可以增大如 120s	後端健康檢查快失敗
proxy_send_timeout	http/server/location	向 upstream 發送請求逾時	60s	5s ~ 30s 高並行下可以增大如 120s	根據業務 P99 設定
proxy_read_timeout	http/server/location	從 upstream 讀回應逾時	60s	5s ~ 300s 高並行或耗時業務可以增大	長連線場景可設大
proxy_buffering	http/server/location	是否緩衝後端回應	on	on (預設)	關閉會增加後端壓力，一般不關
proxy_buffer_size	http/server/location	第一塊 buffer 大小	4k 8k	16k	包含回應標頭，建議 ≥16k
proxy_buffers	http/server/location	回應內體 buffer 數量與大小	8 4k 8k	8 32k	高吞吐場景適當增大

8.6 Upstream 長連線池

指令	上下文	功能說明	預設值	建議設定範圍	注意事項
----	-----	------	-----	--------	------

keepalive	upstream	每個 worker 的空閒長連線數	無	32 ~ 256	總連線池 = worker_processes × keepalive
keepalive_requests	upstream	單個 upstream 連線最大請求數	1000	1000 (預 設, 合 理)	防止連線記憶體洩漏
keepalive_timeout	upstream	upstream 空閒連 線逾時	60s	60s (預 設, 合 理)	通常無需修改

8.7 日誌

指令	上下文	功能說明	預設值	建議值	注意事項
access_log	http/server/location	存取日誌	logs/access.log combined	off (壓測) 或 buffer=64k flush=5s	高並行下關閉 或緩衝寫入
error_log	main/http/server	錯誤日誌層級	error	warn (生產) crit (壓測)	避免 debug 日誌拖慢效能

全國統一服務熱線
4008-555-800

金蝶天燕雲計算股份有限公司（簡稱“金蝶天燕”）成立於2000年，前身為“金蝶中間件公司”，是金蝶集團旗下基礎軟件平台提供商，雲計算國家標準制定企業，國家信創產業核心軟件企業。金蝶天燕是國家863重點研發計劃與核高基重大專項承接企業，也是“兩網一站四庫十二金”國家重點工程的基礎平台提供商，產品廣泛應用於政府、軍工、金融、能源等關鍵行業，累計服務客戶總數超過10萬家。



雲計算國家標準制定企業
金蝶集團旗下基礎軟件企業
資訊技術應用創新核心企業
官網：www.apusic.com

