DBFCDX DIRTYREAD 無預警崩潰解剖與 driver 可控參數總整理

2026-08-05 · HBSTK 專案技術報告(依 Harbour 官方 dbfcdx1.c / dbf1.c 原始碼查證)

一、訴求與現象

為根治 GTWVT 版(Harbour DBFCDX)網路瀏覽比舊 Clipper/HyperDOS 慢的問題,啟用了 Harbour 官方的 RDDI_DIRTYREAD(stk.ini CDX_DIRTYREAD=1,Clipper/SIx Turbo 相容模式)。實測速度確實回來,但客戶站台開始「無預警 CRASH」:程式直接消失、完全沒有任何錯誤訊息。本報告解剖崩潰根因(附原始碼證據)、回答「為什麼 Clipper 可以、Harbour 不行」,並整理 DBFCDX driver 所有可控參數與安全的替代提速路線。

二、崩潰根因:原始碼直接證據

Harbour 在每一次索引移動(skip/seek)都會經過 dbfcdx1.c 的讀鎖函式 hb_cdxIndexLockRead()(第 1184 行,全檔 23 個呼叫點):

```
if( ... || HB_DIRTYREAD( &pIndex->pArea->dbfarea ) )
{
   pIndex->lockRead++;
   return HB_TRUE;   /* DIRTYREAD 開啟時直接返回 */
}
...
ret = hb_dbfLockIdxFile( ... );   /* 被跳過(1):OS 位元組範圍讀鎖 */
if( ret )
   hb_cdxIndexCheckVersion( pIndex );   /* 被跳過(2):版本驗證 = 真正凶手 */
```

hb_cdxIndexCheckVersion()(第 1150 行)讀取 CDX 檔頭 8 bytes 版本號,發現別台改過索引就呼叫 hb_cdxIndexDiscardBuffers() 把「程序內快取的 B-tree 頁」全部丟棄。這是讀取路徑上唯一會使快取頁失效的機制。

因此 DIRTYREAD 開啟後的必然結果:程序內 CDX 頁快取永遠不再失效 → 別台寫入造成頁分裂/頁釋放重用時,本機快取樹與磁碟上的樹分家 → 走訪時拿「舊頁中的子頁編號」讀新樹,讀到已被重用或寫到一半的頁 → 垃圾指標 → C 層 access violation。因為發生在 Harbour 錯誤系統之下,所以程式直接消失、不會出現任何錯誤畫面——與客戶回報的「無預警、無錯誤」

完全吻合。

結論:這是結構性設計而非 bug。RDDI_DIRTYREAD 只是「跳過鎖+跳過驗證」的 SIx 相容開關,Harbour
並沒有(在共享檔案架構下也無法)附帶無鎖一致性協定。

三、為什麼 Clipper 可以、Harbour 不行

層面	Clipper 年代	Harbour + Windows SMB
網路檔案系統	主流 Novell NetWare:重導器不做客戶端資料快取,每次讀頁都真的問伺服器,拿到當下最新頁 = 作業系統免費提供一致性	SMB:byte-range lock 正是強制快取一致的訊號;不上鎖 = 客戶端放心使用舊快取 → stale
程式內快取	Clipper 每次讀頁是真實 DOS read,沒有自己的頁池	Harbour 有 CDX_PAGECACHESIZE 程序內頁池,靠 CheckVersion 維持一致;DIRTYREAD 把唯一失效機制關掉 → stale 疊兩層
當年的警語	SIx/Comix 文件明講 Sx_SetTurbo「更新進行中不安全」,當年也會 GPF;只因 DOS 低併發、寫入頻率低,踩中機率小	多站持續寫入的進銷貨環境,踩中機率大幅提高

一句話:「Clipper 可以」大部分是環境(NetWare 一致性 + 低併發)給的,不是演算法有魔法。

四、DBFCDX driver 可控參數總表(rddInfo / dbInfo)

參數	作用	對本系統的價值
DBI_DIRTYREAD (per-workarea)	dbInfo(DBI_DIRTYREAD, .T./.F./NIL) 三態:強制開/強制關/跟隨全域(dbf1.c:3847)。位元設計:全域關閉時個別工作區仍可強制開啟	★真正可用的折衷:全域關死,只對幾乎不被寫入的表(SET 檔、代碼表 CHAR/REFER/TYPE、唯讀報表暫存)開啟。沒人寫該表的索引 = 沒有撕裂頁 = 安全吃到加速
RDDI_INDEXPAGESIZE	CDX 頁大小 512~8192(2 的冪,dbf1.c:6740);需重建索引才生效	★安全提速:頁變大 → B-tree 變淺 → 每次 seek 讀的頁數變少 → SMB 來回變少。照常鎖定,零崩潰風險。代價:全站同版 EXE、非標準頁第三方工具讀不懂
RDDI_SETHEADER	DBF 檔頭更新時機:MINIMAL(關檔才更)/CLIPPER(commit+replace 旗標)/FULL 等(dbinfo.ch DB_SETHEADER_*)	設 MINIMAL 減少寫端檔頭寫入 → 減少讀端快取失效風暴
RDDI_LOCKRETRY	鎖等待逾時	調整鎖衝突行為
RDDI_LOCKSCHEME	CLIPPER/COMIX/VFP/HB32/HB64 鎖位移方案	互通性與大檔(HB64 避免鎖區重疊),非提速
RDDI_OPTIMIZE / FORCEOPT	filter 查詢最佳化	有 SET FILTER 的瀏覽場景
RDDI_AUTOOPEN / AUTOORDER / AUTOSHARE	結構索引自動開啟/預設順序/網路自動共享	本系統已以 cmxAutoOpen 管理
DBOI_INDEXPAGESIZE	查詢某索引目前頁大小(dbfcdx1.c:8781)	驗證重建後頁大小

五、官方沒有「安全版 DIRTYREAD」,但存在可自製的中間檔位

CheckVersion 的跳過與鎖的跳過寫死在同一個 if,官方沒有「不上鎖但仍驗版本」的檔位。但看懂原始碼後,存在一個理論中間點:不上 OS 鎖(省 2 個來回),但每次仍讀 8 bytes 檔頭驗版本(1 個來回),版本變了照樣丟快取。如此一次索引移動從 3~4 個 SMB 來回降到 1

個(約 3 倍提速),而「stale 頁池」這個最大殺手被治好;殘餘風險只剩「恰好讀到別台寫到一半的撕裂頁」的極小視窗。作法:把 dbfcdx 原始碼拉進專案自編(修改 hb_cdxIndexLockRead 的該 if,DIRTYREAD 時仍呼叫 CheckVersion),可再加 200ms 節流(200ms 內免驗)。

六、四條建議路線

路線	內容	風險/代價	時機
1. per-area dirty read	DBI_DIRTYREAD 只對代碼表/SET 表開(relation 狂 seek 的正是這些表),主檔全關	極低(該表無人寫索引);改動小	立即
2. INDEXPAGESIZE 2048	rddInfo 設頁大小 + 全面重建索引	零崩潰風險;需全站同版 + 重建索引部署	中期
3. 自製 C-patch	無鎖但驗版本(+節流)自編 dbfcdx	工程較大;殘留撕裂頁極小視窗	想要接近 Clipper 速度時
4. letodb	client/server RDD,不再直接走共享檔	架構改動最大;又快又穩	終極解

七、事故時序與教訓

項目	內容
部署	stk.ini CDX_DIRTYREAD=1(LoadIni→hb_SetEnv→GETE,純執行期旗標,免重編)
現象	客戶站台無預警 crash,完全無錯誤訊息(C 層 AV,在 Harbour 錯誤系統之下)
止血	推送 CDX_DIRTYREAD=0 到全部客戶站台 + 本機 PoC 旗標關回 0;重啟即生效
教訓 1	「無預警且無錯誤」的崩潰 = 原生層 AV,方向先找 C 層(driver/快取/指標),不是 PRG 層
教訓 2	相容旗標不等於行為等價:名字叫 Clipper 相容,不代表在現代環境(SMB+頁池)有 Clipper 當年的安全條件
教訓 3	危險 feature flag 一律預設關 + 可遠端推送關閉,本次能快速止血正因如此