B-TREE 及其變形的同步控制介紹

林良哲 吳光閔
南華大學資訊管理學系

摘要

B-樹狀資料結構 (B-TREE) 經常成爲描述資料庫檔案組織的模式[1]，從 1970 年代起，樹狀資料結構與其變形一直持續被人們廣泛的研究；另一方面，同步控制 (Concurrent Control) 是指在資料庫管理系統上的一種控制機制，利用 Lock 或類似的機制，使資料在進行異動時，預防其不會互相的干擾，由此動機，使得我們想探討 B-TREE 是如何在資料庫中進行同步控制演算法的，由目前的文獻當中，已知不同 B-TREE 又有不同的變形[1]，因此，可能會衍生更多種不同的解法。因此，我們由近幾年的資料作整理，由歸類出三種代表的演算類型作探討[2]，分別爲: Naive Lock-Coupling、Optimistic Descent 與 Link-Type，並且配合範例替各位解說，希望能讓有興趣的愛好者，可在短時間內，透過本文能對於整個 B-TREE 同步控制的情形有概括性的認識。

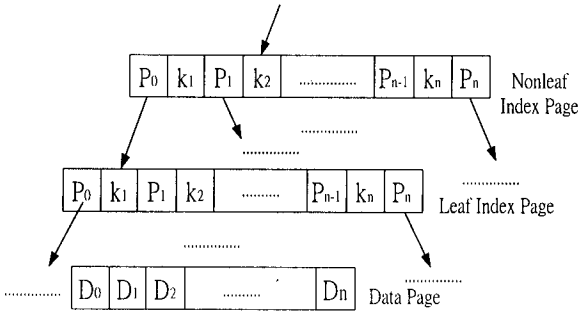
關鍵字：B-TREE、同步控制、Lock-Coupling、 Optimistic Descent、Link-Type

一、B-TREE 與同步控制的背景介紹

由於分散式系統的盛行，在設計資料庫時，往往爲了兼顧同時存取共享資源所可能發生的競賽狀況，所以在規劃存取共享資源時，都會將同步控制機制納入，避免這類問題的發生。在資料庫系統當中，B-TREE 爲最主要的資料表現方式[1]，因此在這一章當中，我們必需把一些相關背景知識介紹給諸位，以讓各位先進能夠清楚快速的明瞭 B-TREE 的定義與同步控制的精神:

TREE 是指由一個或多個節點所構成的有限集合，但是，何謂 B-TREE ? 其在資料庫系統當中又如何能扮演資料同步的角色呢? 這一點，我們手先要瞭解 Order 爲 m 的 B-TREE 性質[1, 14] :

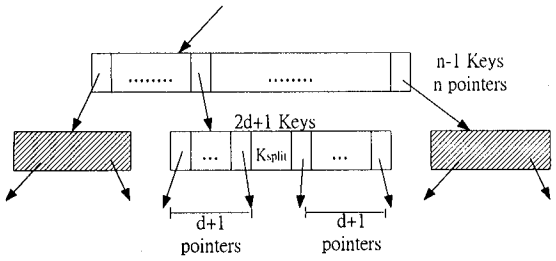
(1) 每一個節點節最大的分支度爲 m (即最多 m 個兒子)。
(2) 除了根節點與葉節點外，分支度至少都需要大於 $m/2$。
(3) 根節點的分支度至少爲 2。
(4) 所有的葉節點都具有相同的階度(Level)，且其值爲由左往右遞增排列。
(5) 需要兩種記錄，一是記錄節點的數值，另外一個則是記錄其



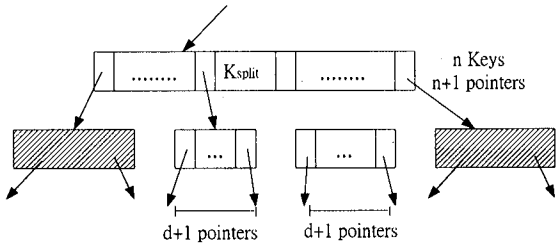
圖一 B-TREE 的構造

聯結指標，令記錄節點的數值爲 <K_0, K_1, K_2, K_3, …. , K_n>，記錄其鍊結的指標爲 <P_0, P_1, P_2, P_3,…. , P_{n+1} >，若是同一層有 n 個節點，則應該有 n+1 個指標 (如圖一所示)。

(6) 當分支度大於 m (溢位的情況發生) 時，則節點就要考慮進行分裂 (如圖二所示)。



a) 分裂前



a) 分裂後

圖二 B-TREE 的 Split

在資料庫管理系統 (DBMS) 當中，通常可允許多筆的異動以存取相同的資料，爲了達到此目標，將安排某種同步控制的機制，以保證異動之間不會相互的干擾，因此，爲了達到某種層次上同步控制的程度，而必須運用到各類鎖定的型態與技巧[2]，例如: 在 Lock-Coupling 系統當中通常使用共享鎖定 (Shared lock, 簡稱 S-lock) 與互斥鎖定 (Exclusive-lock, 簡稱 X-lock) 的型態[1]，當其中一筆異動 A 想要截取某一組值時，必須先對該值取得 S-lock，但若另一異動 B 同時對此一組值請求作異動時，則共享鎖定將會接納異動 B 的 S-lock 而排拒異動 B 的 X-lock；若是異動 A 已經事前取得 X-lock，則任何對於的異動 B 的要求都將會被拒絕。

Optimistic Descent Algorithm 運用意圖互斥鎖定 (Intent exclusive-lock, 簡稱 IX-lock)與共享意圖互斥鎖定 (Shared intent exclusive-lock, 簡稱 SIX-lock) 的型態，當一筆異動 C 某一值取得 IS-lock 時，除了只會排拒其它的異動 X-lock 要求外，其它的鎖定要求都是被容許的 (如表 1 所示)；若是某一組值異動 D 已經取得 SIX-lock 時，則除了 IS-lock 的要求外，其他的鎖定要求都將會被否絕。運用以 IX-lock 來取代 S-lock，使得 Optimistic Descent Algorithm 僅需執行部份區域鎖定，同時，也提高了同步控制的層次。

表1 擴充相容矩陣

	X	IS	SIX	X
X	Y	Y	Y	N
IS	Y	Y	N	N
SIX	Y	N	N	N
X	N	N	N	N

二、 B-TREE 同步控制的介紹

最簡單的 B-Tree 同步控制的方法，利用在任何異動發生時，把 B-Tree 上的所有節點鎖定，免得在同一節點內發生未提交相依的困擾情況。然而，在大多數的 Update 動作 (執行 Insertion 與 Deletion) 時，往往改變 B-TREE 中部份區域的節點，如果鎖定涉及全部區域，則將使其它的異動無

[1]共享鎖定 (S-lock) 與互斥鎖定 (x-lock) 有時也分別稱爲讀取鎖定 (Read-lock,簡稱 r-lock) 與寫入鎖定 (Write lock,簡稱 W-lock)。

法並行，因此，爲了考慮同步控制的程度的提昇而使用各種的動作分析與鎖定的分級，如此一來，產生了僅需鎖定住部份區域的可行性，在 B-Tree 同步控制運用中，整體來說，可以概括的區分爲 3 種方法[2, 3]，分別爲: Naive Lock-coupling Algorithm 系列、Optimistic Descent Algorithm 與 Link-Type Algorithm 系列，本章就以這三個演算法的架構爲基礎，在以下爲大家作探討:

2.1 Naive Lock-Coupling Algorithm 系列

本系列可以說是最受世人青睞且被提出討論最多者，這個演算法是在 1977 年時，首先被 Bayer 與 Schkolnick 所提出[4]，所以這系列的演算法也被稱作 Bayer-Schkolnick Algorithm，其後人又結合同步控制的模型發展成 Two phase locking 理論[2]，演算法主要的特徵如下:

A. 首先，規定每個節點在作存取 (Access 動作前必須要先被鎖定，Lock-Coupling 就是利用這項策略來進行同步的控制，無論在何種狀況下，子節點必須先要被鎖定住後，其父節點的 Lock 才有機會被考慮釋放。

B. 假設執行 Search 的動作，開始時，共享鎖定 (S-lock) 會要求鎖定住根節點，然後繼續往下搜索，在這之間，可經由的適當的路徑 (Path) 找到相關的子節點，S-lock 則會暫時將路徑上所有的子節點鎖定住，直到找尋到目標節點並鎖定住它時，其目標節點以上的所有 Ancestor 才能被釋放。

C. 假設執行 Update 動作 (執行 Insertion 或 Deletion) 時，前半段執行程序與之前所提及的 Search 動作相似，但是，Update 動作則使用的是互斥鎖定 (X-lock) ，當找到目標節點時，會將 S-lock 替換成 X-lock，並且也不會立即釋放目標節點以上已鎖定的 Ancestor，而需等待此一異動結束時，其 X-lock 才會從所有節點上釋放。

2.2 Optimistic Descent Algorithm 系列

此系列與 Lock-Coupling Algorithm 同是皆由 Bayer 等人在 1977 年的同一篇論文[6]中所提出的，因此，也有人把 Lock-Coupling 與 Optimistic Descent Algorithm 視爲爲同一類型，但是，兩種演算法所強調的精神還是略有差異的，Optimistic Descent Algorithm 所使用的策略是: 遭遇到資料需要更新時，前半段使用的模式與 Lock-Coupling 相似，但是，鎖定的機制卻略有不同，在找尋目標節點時，Optimistic Descent Algorithm 使用的方法是以 IX-lock 來取代 S-lock，當 IX-locks 一直作用而延續至目標節點時，再轉換成 X-lock，執行更新的動作；如果，發現其目標節點爲 Unsafe Node (已被其他 lock 預先鎖定)，其 X-lock 就隨即下釋放命令，而使用 SIX-lock 代替 (因爲 SIX-lock 能允許別的動作同時進行讀取，但是，卻不允許其他的動作同時來進行資料更新)，所以通常我們會很樂觀假設，需要使用 SIX-lock 的情況是很少的，由此假設的模擬實驗當中發現，此系列的演算法提供的效率很理想，這也是爲什麼被稱作 Optimistic Descent Algorithm 的原因[2]。

2.3 Link-Type Algorithm 系列

此系列最先的提出者是 P. L. Lehman and S. B. Yao[5]，所以一般也可以寫成 Lehman-Yao Algorithm 或是 LY Algorithm, 在這之後，此類方法較著名提出者還有 V. Lanin and D. Shasha[6] 和Y. Sagiv[7]等人，Link-Type Algorithm 主要是使用 Link 的方式以取代原先使用 Lock-Coupling 的模式，其特徵除了最右邊的節點外，在同一層的每個節點皆會有連接至右邊的兄弟 (Sibling)，主要鎖定操作的方式如下所示：

A. 開始時，先使用 r-lock 從根節點找到目標節點（由於 B^{link}-TREE 的構造，所以必定是葉節點），接著使用 w-lock 鎖定住所有的葉節點 (同時保持至多只有一個 Lock 存在)，然後才方可執行更新的動作。

B. 假設在插入新節點後，發現有溢位 (Overflow) 的情況，則須要產生新的兄弟 (New Sibling) 以平衡 B-TREE，平衡時，先將此層節點平均對分，再與新兄弟的指標產生關係性的連接，當指標連接後，此層鎖定的節點才會被釋放，值得注意的是，其父節點在被鎖定住時，要確定至新兄弟指標被指定好時，才方可釋放其父節點的鎖定。

C. 假設在 Insert 後，父節點也有溢位的情況發生，此時，將重覆上述 B 程序，往上進行 B-TREE 的結構重新組合，在這期間，將會發生其父節點指標無法標示正確子節點路徑的情況，但是，我們還是依然可以利用其 Link 的特性，由其他的路徑來找到所指定的節點以作補救。

三、演算法則說明

在 B-TREE 上最早提出具有同步控制的雛型的，可以追溯至 Parr[8] 與 Samadi[9]，當時提出的控制理論非常的簡單，他們分別把鎖定的範圍定義爲：執行 Pure-reader 時，只需鎖定住一個節點，但是執行 Update 則限於某一個特定範圍之內，對於可能發生結構重組的節點，利用訊號通知再加以鎖定，配合強制異動的次序與集中鎖定方式，而達到無干擾的目的，但是，他們皆無法支援對同一節點進行異動，之後，Bayer 與 Schkolnick 首先提出[4] Lock-Coupling 同步控制的理論，使得同步控制朝向另一個新里程碑邁進。

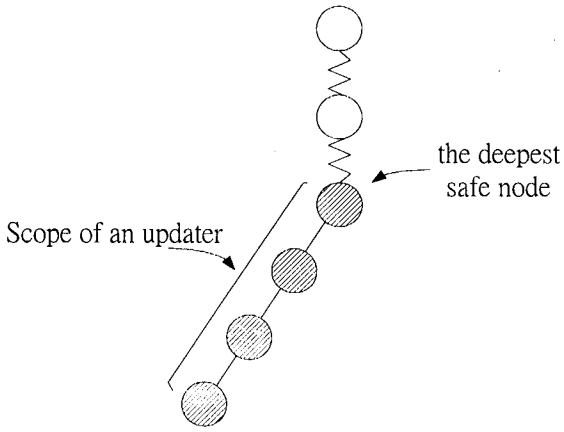
3.1 Naive Lock-Coupling Algorithm 系列

Bayer 與 Schkolnick 在 1977 年所提出的 Lock-Coupling Algorithm，定義 Lock-Coupling 是 "在每個節點在作存取動作 (Access) 時，子節點必須先被鎖定後，其父節點的 Lock 才有可考慮被釋放 "。Bayer 等人爲了提昇同步控制的層級，他們把事件 Updateing-reader 與 Updateing-writer 視爲兩種行爲，當執行資料更新的動作時，起初，執行 Updateing-reader ，當動作完成時，馬上轉變鎖定機制而進行 Updating-writer，其演算法解說如下：

A. 在執行 Pure-reader 時，先以 Read-lock 鎖定根節點與其路徑上的子節點，直到目標節點被 Read-lock 鎖定時，其目標節點以上的祖先才會被釋放。

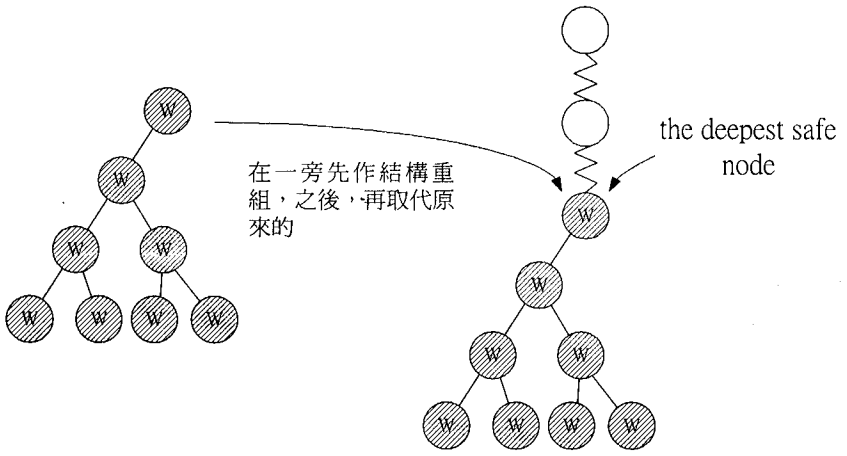
B. 當執行資料更新時，與執行 Pure-reader 相類似，事先鎖定根節點與其路徑上的子節點，但是在找到目標節點之後，將會使用 Exclusive-lock，並且不會立即釋放其目標節點以上的祖先，僅釋放 " The deepest safe node "以上的的節點 (如圖三所示)，以確保未來可能執行結構重組的程序時，其他的異動不會誤闖此範圍 (造成同時存取共享的資源，發生的競賽狀況)，最後，要等待此一更新動作被完成後，其 Lock 才會從所有節點釋放出來。

C. 換句話說，執行 Pure-reader，最後的程序僅僅只有鎖定住單一個目標節點，但是執行 Update-write 卻需要鎖定住 The deepest safe node 以下的範圍，以避免衝突發生，假設找到目標節點時，卻發現正在執行之前的 Exclusive-lock，此時 Read-lock 就要利用 Driving off reader 的方式，"撤退"至指定的限制點 (The deepest safe node) 才可以。



圖三 "The deepest safe node"示意圖

利用 Lock-Coupling Algorithm 中 Read-lock 與 Exclusive-lock 兩種型態的鎖定機制，使得兩個以上指令要求對同一個節點作 search 處理時，能夠突破前人需要排序的限制，再利用同等的鎖定層級緊條件，使得任何較晚發生 search 動作都具有相同等級的優先權，所以可忽略時間的先後次序而行越級 (Overtaking) 的處理。Bayer 與 Schkolnick 利用鎖定的改變方式，成功達到越級和同步控制目的。



圖四 side-banching

我們所選擇的這個代表範例是1982 年 Yat-Sang Kwong 與 Derick Wood[10]所提出的，他們參考 Eill [11]提出把動作區分爲 Pure-reader、Updateing-reader 和 Updating-writer 三種行爲的模式，使之可分別對應到 Read-locks、Write-locks 與 Exclusive-locks，這樣作法可以促進 Lock-Coupling Algorithm 同步控制更具有完整性的機制，這篇論文同時把一般 B-TREE 運作的情況都考慮進去，可以說是十分完備的一篇代表，其說明如下:

A. Searching 和 Insertion 的狀況

1. 當進行 searching 的程序時，首先鎖定根節點，然後依序往下搜索，使用 S-lock 來鎖定下一個路徑上的子節點，再釋放上一個節點，如此一直重覆作用，直到尋找到目標節點，並且鎖定住它爲止，如果找尋至葉節點仍舊找不到這個值，就表示其值沒有資料記錄在這個 B-TREE 當中。
2. 當進行 Insertion 的處理程序時，其方式與上一節介紹 Bayer 等人的模式相同，但是 Yat-Sang Kwong 等人特別再提出新的樹狀結構重組模式，當其同一層的節點數超過其最大容量 m 時(發生 Overflow)，其 "The deepest safe node" 以下的範圍內就要進行節點重新排列，利用發生時間戳記 (Timestamp) 不同的方式提出 Side-Branching (如圖四所示) 以修正前人的演算法，其步驟爲：1)Read-lock 其所有可能重組的範圍 2) 建立 Side-Branch: 先複製一個相同的子樹，並在一旁作內部重組 3) 部重組完畢，則套用 Driving off reader 的限制條件 4)利用重組完畢的 Side-Branch 取代原先的子樹並加入至 "The deepest safe node" 之下。

B. Deletion

Deletion 與 Insertion 前半部的執行程序是非常相似，於皆是使用 Updating-reader 來截取其值，如果確定其值存在，則使用 w-lock 鎖定其範圍，當其同一層 (level) 的節點數少於其最大容量的 1/2 時 (即發生 Underflow)，這時也是使用 Side-Branch 的方法作修正。

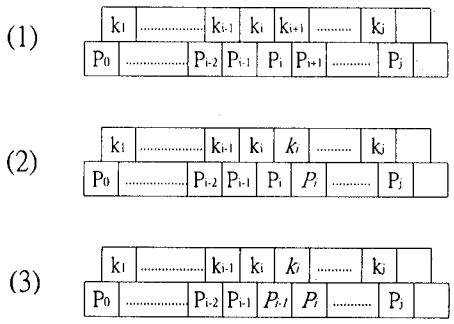
C. Intranodal Concurrency

爲了使重組完畢的 Side-branch 能順利取代原先的子樹，在取代至 "The deepest safe node" 之下時得需要考慮指標與資料的一致性，這樣可免除死結的情況發生，Yat-Sang Kwong 等人提出 Repeated reading technique 的模式(如圖五所示)，利用多複製一組 Key-point，解決結構重組後在取代時，指標與其值無法相同的問題，同時也能滿足了資料庫語意上的一致性。

3.2 Optimistic Descent Algorithm 系列

Bayer 等人在 1977 年同一篇論文[6]中所提出同步控制架構可分爲的三個部份，Lock-Coupling Algorithm 是在前兩個部份被提及，而 Optimistic Descent Algorithm 則強調是利用第三部份 Driving off reader 的方式推導出來的衍生類型，Eill[11]從這個觀點再衍生出新的 Algorithm (即 Optimistic Descent Algorithm 系列)，他把鎖定的方式區分爲 Read-locks 、 Write-locks 與 Exclusive-locks ，使其分別對應到 Pure-reader 、 Updateing-reader 和 Updating-write，打破了以往 *FIFO* 的原則，使得不只是 Search 對 Search 而是 Search (Pure-reader) 對 Update (Updateing-reader)時，任何較晚發生 Search 動作都可以不依據時間次序的先後而進行越級 (Overtaking) 的處理，當兩個執行事件發生互相衝突時，衝突的一方 (較晚發生的) 即會停止而不運作，並 "Rollback" 至指定的限制點，使之前的事件能夠繼續的完成。

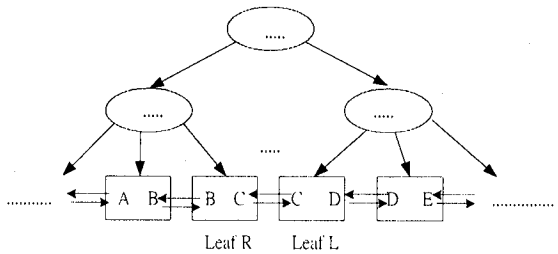
這個方法稱作樂觀法(Optimistic)，因爲它的假設前提是:預期只有少數的執行事件會發生衝突，才需要 Rollback 這樣的機制，另一方面，執行異動 Updateing-reader 時，雖然"The deepest safe node " 以下的範圍爲了避免衝突發生已經被事先鎖定了，但是 Read 動作仍然可以進入其間作 access ，當節點遇上正在執行 Exclusive-lock 動作時，就可使運用 Driving off reader 的條件，使 Read 動作 "Rollback" 至指定的限制點，如此方式能使得同步控制的層級更上一層樓。



圖五 Intranodal Concurrency

在 Optimistic Descent Algorithm 系列代表範當中，我們選擇了 1999 年 Sungchae Lim 、 Yoon Joon Lee 與 Myoung Ho Kim[12]所提出的論文作爲介紹，B-tree 的同步控制發展至這幾年，遭遇到一些瓶頸，所以就有些人發展出各式各樣的 B-tree，其中又以我們在下節提及的 Link-Type Algorithm ，其整體系列執行效率爲最好，但是爲了不使用特殊的樹狀結構的方式，Sungchae Lim 等人於是設計出使用在 B^+-Tree 上，以 LC-SIX (共享意圖互斥鎖定與 Lock-Coupling 合併) 爲基礎的演算法，其內容簡要說明如下:

B^+-Tree 的特性爲資料儲存於葉節點，同時，其葉節點與葉節點間是有指標互相連接的，在此 Sungchae Lim 等人提出 Key-Range 的觀念，指出儲存於葉節點的資料型式有最大值與最小值的範圍，且其左節點的最大值必等於右節點的最小值 (如圖六所示)。



圖六 B^+-Tree 結構圖

LC-SIX 的精神在於，找尋目標節點起使用的是 SIX-lock，當子節點被鎖定住後，其父節點的 Lock 才方可有機會被釋放。

A. Insertion 的狀況:

當 Update-reader 使得目標節點至 Unsafe search path 內的節點皆已經被 SIX-lock 鎖定時，要等到目標節點確定被提交成爲 Safe node 時，Unsafe search path 內鎖定的 SIX-lock 的其才會釋放。

1.當插入新節點時，首先需要進行分裂 (Splitting) 兩個葉節點的程序，Sungchae Lim 等人所提出的步驟如下:(1).啓始狀態(2).插入新節點，令新節點的最大值相等於差入後其右邊節點的最小值(3).令新節點的最小值相等於插入後其左邊節點的最大值(4).節點分裂完成 (如圖七所示)。

2. 進行分裂動作時，若是發生 Overflow 的情況，就得再分裂其內部節點，其原則也是和 Side-branch 相同，先複製一個相同的子樹，並在一旁先作內部重組， 待內部重組完畢，則把已重組好的值取代原先的範圍。

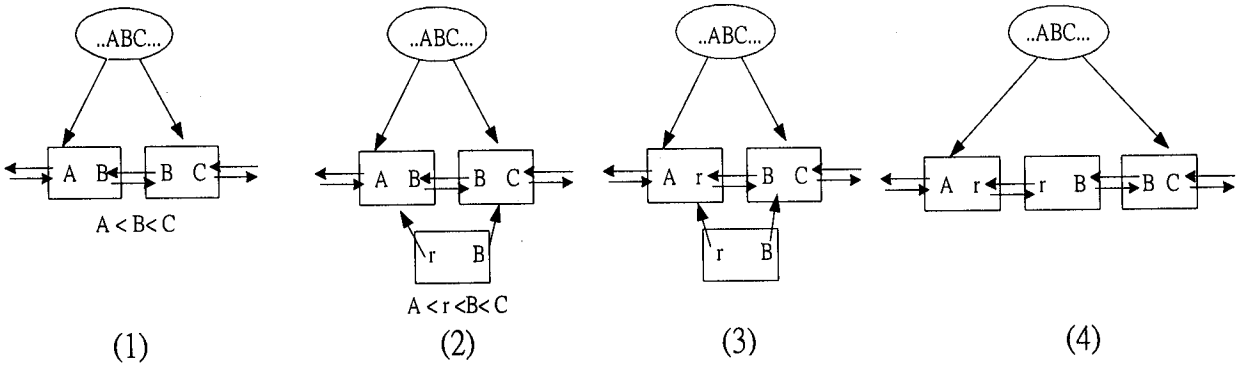
B.Delection 的狀況:

1. 當刪除節點時就要進行合併 (Merging) 兩葉節點的動作，作者提出 4 個步驟:(1).啓始的狀態(2).將預備刪去的節點，令其左邊節點的最大值指標指向右邊節點的最小值(3).令右邊節點的最小值指標指向右邊節點的最大值，建立新連接(4).除去刪除節點的指標，刪除節點後，合併即可完成 (如圖八所示)。

2. 假設進行合併動作時，若是發生 Underflow 的情況，此時就會影響到子樹的內部節點，其過程原理也是和 Insertion 一樣，事先在一旁先複製一個相同的子樹，並在一旁先行作業，待內部重組完畢，則以重組好的子樹取代之。

3.3 Link-Type Algorithm 系列

Link-Type Algorithm 系列是 P. L. Lehman and S. B. Yao[7]所提出，他們修改B*-TREE 的結構，除了每一層最右邊的節點指標其值爲 null 外，令每一個節點的指標 (Pointer) 都要與右邊的節點作連結，形成流動的節點(Current node)，這樣的定義能夠確保在同一層的每個節點都具有一致性，因此，若是在樹狀結構重組時，如果暫時無法由父節點找到正確子節點，此時，就可以利用同一層節點的指標，經由 " 繞路" 找到其目標物，這樣由B*-TREE 修改後的特殊樹狀結構被稱爲



圖七 Splitting a leaf node

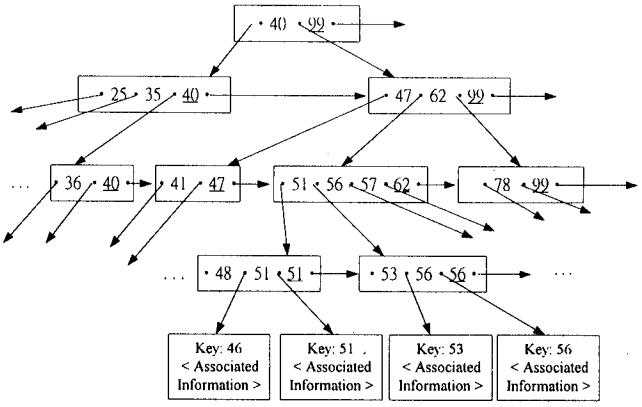
B^{link}-TREE (如圖九所示)。B^{link}-TREE 對於 B^{*}-TREE 或是 B^{+}-TREE 最大的結構不同在於，B^{link}-TREE 的每一層節點都是利用Link Point相連接的，即使我們在樹狀結構重組中可也以利用Link的方式輕易的找到其目標節點，由此可知，Link Point 的方式能夠取代 Lock-Coupling 鎖定的模式，這也是爲什麼Link-Type Algorithm 系列能提供更高層次的同步控制演算法的主因，但是，目前在 Link-Type Algorithm系列，其中直遭遇到Underflow 的狀態下尚無發展出適當的解決對策，這也是最令人感到惋惜的地方。

1.Search

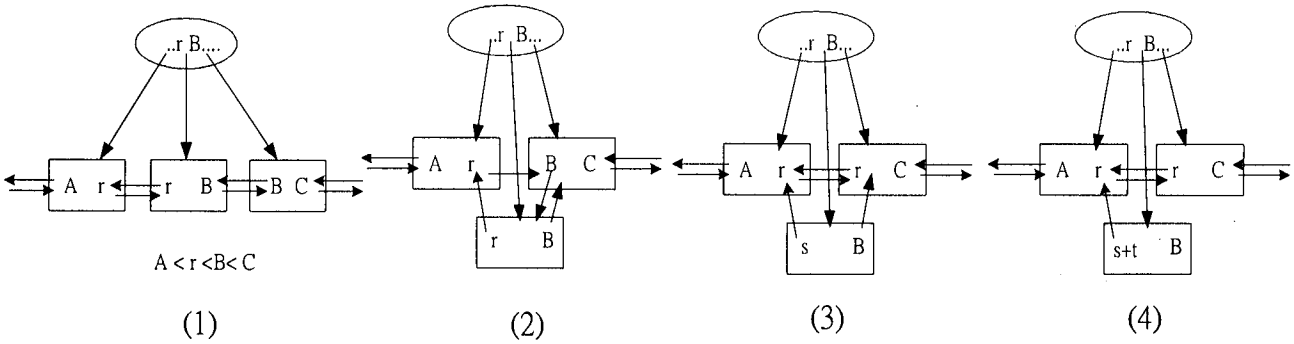
當要找尋一個爲 v 的值時，從根節點開始和 v 值作比較，然後往下搜索，一層接著一層，假設此 v 值存在則我們可經由指標而在葉節點當中找尋到，假設我們在藉由指標的搜尋的過程中發現目標葉節點當中最大值少於 v 值，則可以知道其父節點的指標所指示的原來值可能已經被更動了，而我們在找尋的真正目標值 v 現在一定正在進行分裂，因此，我們可以藉由 Link Point 的找尋來作校正，藉由此法，我們最終仍然可藉由其最接近的 Sibling 找正確的尋到 v 值。

2.Insertion

插入新節點的過程中 Updeate-reader 和搜尋程序頗爲類似，不同的地方在於，如果遭遇到Overflow的狀況，因爲指標安置的程序得有一定的步驟 (如圖十所示)，有這類步驟才能確保同步控制的可行性，當執行Updeate-reader時，當r-lock找到其葉節點，然後會再使用w-lock將所有的葉節點鎖定，直到Updeate-writer執行結束爲止。



圖九 B^{link}-tree



圖八 Merging two leaf node

四、各種同步控制的比較

這一章主要是將以上介紹過的同步控制的演算法作討論，分別從B-TREE家族的分類與同步控制理論的角度，對於 Naive Lock-coupling Algorithm、Optimistic Descent Algorithm與Link-Type Algorithm系列，在以下爲各位作探討:

4.1 各系列對應B-TREE家族的關係

這一節，我們利用三種演算法的分類架構，研究其對應B-TREE家族的關係 (如表2所示)，在Naive Lock-coupling Algorithm 系列當中，Samadi[11]與Parr[10]提出的例子分別對應於一般的B-TREE 與 B^*-TREE 的例子，以區分Pure-reder與Updater的動作而達到同步控制的目標，後來Bayer等人[4]延續使用 B^*-TREE，使用兩種鎖定的機制進行改良，之後 Eill[11]也創新提出同步控制使用於2-3-TREE的結構方式，並且把 Updater 動作區分類爲 Update-reader與 Updater-writer, Kwong 等人[10]則是套用 B^+-TREE 資料的儲存結構，並且再補充Eill沒有提及Delection中結構重組的情況，改善其演算法。

Optimistic Descent Algorithm 系列當中，雖然 Bayer 等人[4]所提出的Driving off reader 的新模式，首次允許Pure-reader程序可以進入Update-readaer的範圍的關念，但是其仍舊與Lock-coupling Algorithm使用的 B^*-TREE結構相同，之後，爲了使得運算的效率更加提昇，Eill[13]套用同樣的2-3-TREE 類型，解釋 "撤退至指定的限制點" 這個模式，在1999年Sungchae Lim 等人[12]所提出的論文作以 B^+-TREE 爲儲存結構，以LC-SIX的上演算法，使同步控制能夠被更有效的完成。

Link-Type Algorithm 系列是 P. L. Lehman and S. B. Yao [5]所提出，他們修改B*-TREE 的結構，利用每一個節點的指標都與右邊的節點作連結，確保在樹狀結構重組時，父節點能正確找到子節點，後來的提出者還有 V. Lanin and D. Shasha[8] 和 Y. Sagiv[9]等人，這

表 2 各系列對應B-TREE家族的關係

		B-TREE	B^*-TREE	B^+-TREE	2-3-TREE	B^{link}-TREE
Naïve Lock-coupling	Samadi[9]	○				
	Parr[8]		○			
	Bayer[4]		○			
	Eill[11]				○	
	Kwong[10]			○		
Optimistic Descent	Bayer[4]		○			
	Eill[11]				○	
	S.Lim[12]			○		
Link-Type	P. L. Lehman[5]					○
	V. Lanin[6]					○
	Y. Sagiv[7]					○

些人皆是使用 B^{link}-TREE ，利用Link 的優點，取代前兩種系列所使用的Lock-Coupling 模式。因此，我們可以另外作一結論，使用 Link-Type Algorithm 系列者，B-TREE 結構較單一化，但是若是涉及使用的 Lock-Coupling 模式者，則 B-TREE 結構就表現較多元性了，這也是爲什麼 Lock-Coupling 被人們較廣泛研究的原因。

4.2 同步控制理論的比較

同步控制 (Concurrency Control) 的研究理論由 Bharat Bhargava[13]的分類方法可區分爲 3 種類型，其定義概述如下:

A.等待機制型 (Wait Mechanism) :當遭遇兩個事件衝突時，引起衝突的一方 (較晚發生) 必須 "等待"，直到對方執行事件皆完成時才得開始執行其事件。

B. 時間戳記機制型 (Timestamp Mechanism):事先在每一個事件安置一個獨一無二的時間戳記，執行事件的優先次序是由此時間戳記所決定，此戳記可能是表示事件的起始點、中間點或是結束點，並且利用出現時間的不同，相對應於不同的版本來解決此類問題。

C. 撤回機制型 (Rollback Mechanism) : 假設兩個執行事件發生互相衝突時，衝突的一方(較晚發生的)即會停止而不運作，並"撤退"至指定的限制點，使之前的事件能夠繼續的完成，這個方法也稱作樂觀法 (Optimistic)，因爲它的假設前提即是:預期只有少數的執行事件會發生衝突，才需要 Rollback 這樣的機制。

由於 Naive Lock-coupling、Optimistic Descent 與 Link-Type 三種系列演算法的發展至近年，其演算法多朝向利用混合三種機制的方式達到並行的目的，因此，我們取出四個著名例子 : 以 Bayer 等人[4]代表 Lock-coupling 與 Optimistic Descent 最初系系列的演算法，以 Y. Kwong and D. Wood [5]代表 Lock-coupling Algorithm 系列，以 Sungchae Lim 等人[12]代表 Descent Algorithm 系列，以 P. L. Lehman and S. B. Yao[5]代表 Link-Type Algorithm 系列，來作以下的歸納 (如表 3 所示):

表 3 同步控制理論的比較

	Lock-coupling 與 Optimistic Descent 最初系系列	Lock-coupling	Optimistic Descent	Link-Type Algorithm
提出者	Baye[4]	Kwong[10]	Sungchae Lim[12]	P. L. Lehman[5]
發表時間	1977	1982	1999	1981
等待機制型	有	有	有	有
時間戳記機制型	無	有	有	無
撤回機制型	方法 1,2:無 方法 3:有	有	有	無

五、結論

B-TREE 使用 Lock 機制或改變其資料結構，使目的都是在其預防資料在進行異動之時發生死結，進而提高同步控制的程度，由 T. Johnson 與 D. Shasha[2] 的模擬結果得知，整體來說，Link-Type Algorithm 系列執行所需要的時間的顯著的低於其他兩者，再者 Optimistic Descent Algorithm 次之、Naïve Lock-coupling Algorithm 系列爲最高；若是以節點數多寡分類，則 Naïve Lock-coupling Algorithm 系列被推薦使用於節點數較少的時機，而 Optimistic Descent Algorithm 則相反，由以上的實驗結果當中，我們可推論，鎖定型態越多，其同步控制的層次就會越低，因此，執行的速度也是越差，因爲 Link-Type Algorithm 系列利用 Link 可繞道的方式來找尋節點，而省卻不必再轉變鎖定的方式，如此一來使得效率大大的提昇。

Link-Type 有別於其他兩者，藉由 Link Point 的方式，在處理程序中我們可發現，在其進行 Update-reader 時，鎖定(r-lock)的節點範圍是隨著節點的搜尋而改變，相反的，而不像是 Lock-Coupling 鎖定的模式，執行 Update 時需要固定鎖住 The deepest safe node 以下的範圍，使得結構重組不受干擾，也促成 Link-Type Algorithm 系列執行效率高，但是，也相對需要額外的的儲存空間來記錄其同一層節點所連接的指標。未來，我們可以朝向 Link-Type Algorithm 系列繼續發展，利用 Link 可繞道的精神，提出更有效率且更完整的演算法。

參考資料

[1] Douglas Comer, "Ubiquitous B-Tree", *ACM Computer Survey* 11, Jun 1979, pp. 121-137.

[2] Theodore Johnson and Dennis Shasha, "A Framework for the Performance Analysis of Concurrent B-tree Algorithms", *Proceedings of the ninth ACM SIGACT-SIGMOD-SIGART symposium on Principles of database systems*, 1990, pp. 273-287.

[3] V. Srinivasan and Michael J. Carey, "Performance of B-Tree Coneurreney Control Algorithms", *Proceedings of the 1991 ACM SIGMOD international conference on Management of data*, 1991, pp. 416-425.

[4] R. Bayer and M. Schkolnick , "Concurrency of operations on B-trees", *Acta Informatica* (9), 1977, pp. 1-21.

[5] P. L. Lehman and S. B. Yao, "Efficient locking for concurrent operations on B-trees", *ACM Transactions on Database Systems*, 6(4), 1981, pp. 650-670.

[6] V. Lanin and D. Shasha, "A symmetric concurrent B-tree Algorithm", *In 1986 Fall Joint Computer Conference*, 1986, pp.380-389.

[7] Y. Sagiv. "Concurrent operations on B*-trees with overtaking.", I*n Fourth Annual ACM SIGACT / SIGMOD Symposium on the*

Principles of Database Systems, 1985, pp. 28-37.

[8] J. R. Parr, "An access method for concurrently sharing a B-tree base index sequential file," *Dep. Computer Science, Univ. of Western Ontario, London, Ont., Tech. Rep. 36*, Apr. 1977.

[9] Samdi B, "B-trees in a system with multiple users", *Inf. Process,* Lett. 54, Oct. 1976, pp. 107-112.

[10] Y. Kwong and D. Wood, "A New Method for Concurrency in B-Tree", I*EEE Transactions on software Engineering*, VOL. SE-8, NO. 3 May 1982.

[11] C.S. Ellis, "Concurrent search and inserts in 2-3 trees", Acta Informatica, 14(1), 1980, pp. 63-86.

[12] Sungchae Lim、Yoon Joon Lee and Myoung Ho Kim, "A Restructuring Method for the Concurrent B^+-Tree based on Semantic Consistency, *IEEE* , 1999 , pp.229-236.

[13] Bharat Bhargava, "Concurrency Control in Database System", *IEEE Transactions on Knowledge and Data Engineering*, VOL.11, NO.1, January/February 1999, pp.3-16.

[14] 劉凱, 資料結構綜合整理, 大碩出版社, 台北市, 民國 84 年。

[15] C. J. DATE 作, 黃加佩譯, 資料庫系統概論, 儒林出版社, 台北市, 民國 86 年。