運用邏輯版本與實體版本之物件導向資料庫系統並行控制機制

A Concurrency Control Mechanism with Logical and Physical Versions in OODBMS

賈坤芳　　陳世穎
Kuen-Fang J. Jea and Shih-Ying Chen

國立中興大學資訊科學研究所
Institute of Computer Science
National Chung-Hsing University

{kfjea, sychen}@cs.nchu.edu.tw

摘要

支援物件版本是物件導向資料庫的重要功能，而並行控制更是資料庫系統改善效率所必須具備的功能，目前物件導向資料庫將此兩項功能分置於不同的模組，本論文提出了一個物件導向資料庫之版本模型，可以結合版本控制機制與並行控制機制。其方式是利用邏輯版本與實體版本、永久物件與暫時物件等概念來建立版本模型，並結合 Timestamp Orderin 機制而達成並行控制。在我們的分析中發現，利用此版本模型建立的並行控制機制較傳統的 Ree 模型（結合版本與 Timestamp Ordering ）並行控制機制有更佳的並行控制能力。

關鍵詞：版本控制、並行控制、多版技巧、物件版本模型、物件導向資料庫

一、簡介

物件導向資料庫的--項重要特徵就是支援物件版本(Object Version)的功能，許多應用，例如CAD/CAM/CAE/CASE, GroupWork, Office Information System, 等，均須依賴良好的物件版本模型(version model)，提供適當的表示方式來處理資料庫中的物件及其衍生的版本。所謂物件版本的觀念就是同一個物件可能有許多不同的版本，例如在 VLSI 或軟體開發的環境中，一個物件可能代表某個電路設計或是程式模組，隨時間的演進，此物件可能被修正或是增強功能而出現不同的版本。爲了表示這類的物件資料，物件導向資料庫必須提供一個物件版本模型。同時爲了儲存及處理版本模型所表示的資料，物件導向資料庫必須具備一個高效率的版本控制機制(version control mechanism)。另一方面，資料庫管理系統利用並行控制(concurrency control)的技巧，讓交易(transaction)得以並行執行，增加系統效能，若交易間有存取衝突時，並行控制機制必須決定那一交易有權優先使用該項衝突的物件。通常資料庫系統使用鎖定(lock)的方式，例如當一交易欲讀取某一物件時，必須先取得對該物件讀取鎖定(read lock)的權利，以防止讀取正在被其它交易修改中的資料，或是讀入資料後被其它交易修改。

然而，目前的物件導向資料庫系統將上述兩項功能分置於不同的模組（或機制）中。由版本的觀念來看，物件導向資料庫的物件版本屬於應用層，提供使用者取得適當的物件版本；而多版並行控制的版本屬於系統層，用以增加系統效能。兩者雖然層次不同，卻都必須在資料庫系統中建置版本控制機制(version control mechanism)，以完成兩個層次的工作。因此，本研究主要目的是設計一個整合的模型，整合這兩個層次的機制爲一，以完成這兩個層次的工作；亦即，此模型可提供版本控制以管理物件版本，也提供並行控制以增加系統效能。

本論文的架構如下：第二節回顧相關的研究，而第三節敘述本研究之問題。第四節介紹一個結合邏輯版本與實體版本的版本模型，第五節提出利用此版本模型的 Timestamp Ordering 並行控制機制（TOLPV），並加以分析。第六節爲本文所提出TOLPV 並行控制規則與 Ree 模型的比較。最後，第七節爲結論與未來研究方向。

二、相關研究

並行控制的主要功能是使多筆交易能夠同時在資料庫系統中執行，以增進系統效能 大部分的並行控制機制都遵循 serializabili t 理論[3,5,6]，也就是要求多筆交易同時運作時，其結果與交易依序列排程(serial schedule)執行的結果是相同的。

Lock-based 並行控制機制要求交易對於物件的使用必須事先取得適當的鎖定（lock），基本上鎖定可分爲 shared

與 exclusive 兩種，利用物件目前被某交易鎖定的型態與另一交易欲使用此物件的方式是否相容，來確保交易間的 serializabilit y特性；其中最著名的是 Two-Phase Locking（2PL）機制[3]，將整個交易執行分爲鎖定取得階段（growing phase）－取得所有欲使用物件的權力，以及釋放階段（shrinking phase）－釋放所有取得的物件，當交易一旦釋放第一個鎖定就進入釋放階段，不可再取得任何新的鎖定。2PL 有兩個缺點：第一是必須有死結（deadloc ）處理機制來處理發生死結的情形；第二是系統的並行程度較低，因爲交易必須等待欲使用的物件被其他交易在釋放階段釋放後才可存取，而非在物件使用結束後立即可以存取。

Timestamp Ordering 並行控制機制[9,10]則是利用賦予每筆交易不同時戳（timestamp），針對物件的讀取時戳（read timestamp）及寫入時戳（write timestamp）比較，判斷交易與物件間時間順序的關係，來決定交易可以繼續執行或是必須回復，以維持交易間的 serializabilit 。Timestamp Ordering 控制機制的優點是可避免死結的情形發生，而在多版並行控制機制中，系統可依據時戳給予交易適當的物件版本，避免交易等待的時間。然而，此機制會有連鎖回復效應（cascading rolled back）的情形產生。

Optimist i 並行控制[4]則在系統大多是唯讀交易，交易間不會頻繁的產生存取衝突的情況下，將交易執行區分爲三階段：讀取（read）階段允許讀取物件或是將準備寫回資料庫的資料在自己的工作區中修改；驗證（validatio ）階段驗證資料寫入後是否違反 serializability；寫入（write）階段依據驗證結果將資料寫回或是將交易回復，重新執行。Optimisti 並行控制機制並不會產生死結或連鎖回復效應的情形。

多版並行控制是由[1]所提出，應用同一資料不同的版本來解決交易執行時讀寫衝突的問題，但建立的版本模型爲一維的模型，未與較複雜的版本模型合併討論。[2]曾經針對多版並行控制演算法的效能做過分析。而首先將版本與 Timestamp Ordering 結合的則是[8, 9]，並應用於 AVANCE 系統。AVANCE 利用 Ree 模型中的版本做並行控制，存取適當的版本，每個物件是由一串的物件版本所組成，每一個物件版本 Vc,e 有一合法範圍[c,e] (c≤e)，此範圍表示此版本在時戳 c 產生，在時戳 e 結束。爲了提高並行程度，Reed 模型允許經交易修改過的資料以暫時性版本存在該物件 possibilit 中，以後在交易存取物件版本時再依據此暫時性版本的合法範圍[c,e]是否破壞資料庫一致性以決定交易成功或是必須回復，重新執行。在 Reed 模型中的版本衍生關係爲簡單的一維時間序列，侷限了應用的領域。

Rastogi 等人[8]提出，若使用者可以接受讀取到稍微過時資料的情況下，利用兩種方式提高 main memory database 中讀取的並行能力：第一是利用原有的版本控制機制提供適當的版本給交易使用（該論文稱此版本爲 logical version）；第二是寫入交易將欲修改的物件複製到交易的工作區做修改（在工作區中的版本爲該論文所稱的 physical version），而不影響其他交易讀取在 main memory database 中的該物件，以提高讀取運算的並行能力。該論文之版本控制機制仍未與並行控制機制實際結合爲一，僅藉由區分 logical 與 physical version 而減少影響讀取運算的衝突。

綜觀上述研究，除了 Reed 模型所應用之 AVANCE 系統外，都僅解決並行控制與版本控制一部份的問題，尚未由一整合性的觀點，提出適當的模型，結合此兩種機制爲一，以更進一步提高並行控制之效率。

三、問題描述

物件導向資料庫提供物件版本模型及版本控制機制處理儲存的物件與其衍生版本。另一方面，資料庫管理系統也使用並行控制的技術，讓許多交易得以同時執行，增加系統效能。然而，目前的物件導向資料庫系統將上述兩項功能分置於不同的模組中。由版本的觀念來看，物件導向資料庫的物件版本屬於應用層，提供使用者存取適當的物件版本；而多版並行控制的版本屬於系統層，用以增加系統效能。兩者雖然層次不同，卻都必須在資料庫系統中建置版本控制機制(version control mechanism)，以分別完成兩個層次的工作。

Ree 模型所使用的版本模型爲以時戳順序所建立的一維版本模型。當交易進入系統時，以交易的時戳決定一適當的版本供使用者使用，藉物件的多版本與交易時戳的關係以提高並行的能力並維持資料庫的一致性。此法並未提供版本模型中常見的由物件衍生出多個物件版本所形成的二維階層，因此侷限了此法的應用。而寫入交易不斷的產生新版本亦造成系統儲存及擷取的複雜度。另一方面，Reed 模型的同步（synchronization）動作會延遲交易的執行；也因爲同步動作必須等待，而有死結的情形產生。

因此，本論文主要研究問題是設計一個整合的模型，提供版本控制機制管理物件版本，並提供並行控制機制以增加系統效能；此一模型將系統層與應用層的機制整合爲一，以完成這兩個層次的工作，解決 AVANCE 系統中 Reed 模型並行控制機制的缺點。我們將以傳統版本控制機制管理的版本（稱爲邏輯版本），以及儲存在資料庫中或系統工作區中的物件版本（稱爲實體版本）兩種方式來建立版本模型，如此可以利用傳統的版本控制技術管理版本，亦可以利用實體版本以增加並行控制能力。以下二節，分別就我們提出的版本模型與其並行控

制機制做一介紹。

四、版本模型

一般物件版本模型中，最常見的運算包括讀取(read)、更新(update)、新增(derive)或刪除(delete)版本等運算，而 Ree 模型中僅提供交易讀取（rea ）和寫入（write）兩種運算，因此只有一維的產生版本及管理版本能力。版本管理爲物件導向資料庫系統的一個重要特徵，因此，在我們的版本模型中，包含了支援讀取、更新、新增、刪除版本這四種運算。這些運算除了切合版本管理的需要外（以二維方式產生版本），也可藉由運算間的關係，提高並行控制能力。

另一個提高系統並行能力的技巧是 允許交易使用儲存在系統工作區中的暫時物件（實體版本），因此，我們將物件版本的觀念擴展爲包含邏輯版本（代表資料庫中的版本）與實體版本（代表系統工作區中之版本），藉此版本模型來管理邏輯版本以及使用實體版本中的 (永久與暫時）物件，來提高交易執行的並行性。以下是我們的版本模型的說明。

4.1 永久物件與暫時物件

定義 1□永久物件□□—物件版本的永久物件(Permanent Object)是在交易完成後產生並儲存在物件導向資料庫中的物件版本。

定義 2□暫時物件□□—物件版本的暫時物件(Temporary Object)是暫存在系統工作區中的物件版本，它在交易的每一個更新運算後產生或由原先的永久物件因被取代而產生。

若交易新增或更新某一物件（版本），在交易完成後，會對該物件產生一永久物件，儲存於資料庫中。後來的交易若欲更新此一物件，需將此永久物件由資料庫中讀至系統工作區中，在交易執行期間，每一個更新運算均會在系統工作區中對物件產生一暫時版本，此暫時版本則稱之爲暫時物件(temporary object)。當交易完成後(committed)，最終的暫時版本將被儲存到資料庫中，成爲永久物件(permanent object)，而原先被更新物件的永久物件則因被取代而又轉換爲暫時物件，暫存在系統工作區中。所以，一個物件版本被更新時，在工作區中存在唯一的永久物件與多個（或無）暫時物件，這樣，交易可以讀取存在於系統工作區中的暫時及永久物件，提供較佳的並行能力。

下一小節，我們將以版本衍生圖爲基礎，描述版本模型中所擴展的邏輯版本與實體版本的概念。

4.2 版本衍生圖、邏輯版本與實體版本

版本衍生圖(version derivation graph)[7]是傳統版本模型描述一個版本物件衍生不同版本的有向非循環圖(directed acyclic graph, DAG)。版本衍生圖的節點由不同的物件版本所組成，而物件版本與其衍生的物件版本間關係即爲此圖的邊，由應用層的觀點而言，每一個節點代表儲存在資料庫中的一個物件版本。

爲了運用儲存在系統工作區中的暫時與永久物件，我們將物件版本的觀念擴展爲邏輯版本與實體版本，將版本衍生圖中的節點視爲邏輯版本，而節點的內容爲該邏輯版本的所有實體版本所構成的集合，如圖 1 所示。

圖 1 中的邏輯版本(logical version)指的是版本衍生圖中的一個節點，代表某一個物件版本。版本控制機制利用衍生圖中的邏輯版本管理版本物件。而物件版本的實體版本(physical version)有兩類：一類是儲存在資料庫中的永久物件，對於某一物件版本而言，它是唯一的；另一

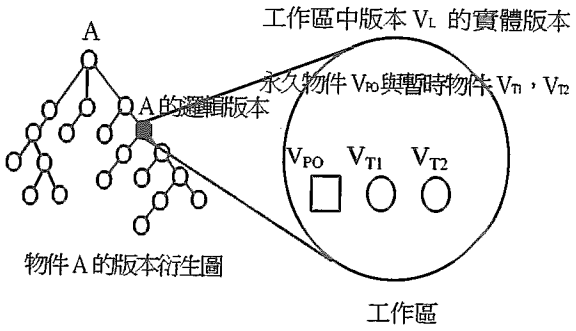

圖 1：邏輯版本與實體版本

類是系統工作區中因交易的更新運算所產生的暫時物件。換言之，某一個物件版本的邏輯版本是版本衍生圖中的節點；而它的實體版本（包含永久物件與暫時物件）是交易實際存取的物件版本。例如，圖 1 版本衍生圖中的節點 V_L 爲物件 A 的邏輯版本 V_L，而 V_L 目前有 V_{PO}、V_{T1}、V_{T2} 等實體版本，其中 V_{PO} 表示儲存在資料庫中 V_L 的永久物件，而 V_{T1}、V_{T2} 則爲在工作區中因交易的更新（update）運算而產生的暫時物件。邏輯版本、實體版本與版本衍生圖的正式定義如下：

定義 3□實體版本□: 物件版本 V 的某一實體版本，以 V_P 表示，是該物件版本 V 的永久物件或是其在系統工作區的任一暫時物件。

定義 4□邏輯版本□: 物件版本 V 的邏輯版本，以 V_L 表示，是物件版本 V 的所有實體版本所構成的集合。

定義 5□版本衍生連結□: 版本衍生連結(Version Derivation Link) ϑ，以→符號表示，代表某兩個邏輯版本間的衍生關係（透過新增運算而產生）。對二個邏輯版本 V_{L1} 及 V_{L2}，$V_{L1}\rightarrow V_{L2}$ 代表 V_{L2} 是由 V_{L1} 經新增運算而產生。

定義 6□版本衍生圖□: 版本衍生圖(Version Derivation Graph)，以 G 表示，是一個有向非循環圖(directed acyclic graph, DAG)，而 G=({V_L}, {ϑ})，其中{V_L}爲 G 中的所有邏輯版本，而ϑ（如定義 5 所示）爲 G 中所有 V_L 間的衍生關係。

版本衍生圖中的每個物件版本皆有版本狀態，依據這些狀態，系統允許使用者得以執行某些特定的運算。在下面二小節中，我們將描述版本模型中的邏輯版本與實體版本可能存在的狀態。

4.3 邏輯版本的狀態

傳統的版本模型中，物件版本分爲三種狀態：暫存狀態(transient state)、正運作狀態(working state)和永久狀態(released state)[12]。我們的版本模型以這三種狀態表示邏輯版本的狀態，暫存狀態表示此物件版本可以 更新，但無法衍生新的版本(邏輯版本)；正運作狀態表示此物件版本已不允許再被修改，但可以用來衍生新的版本(邏輯版本)；永久狀態表示此物件版本已經穩定，不允許更新或衍生新版本。圖 2 爲邏輯版本狀態轉換圖。

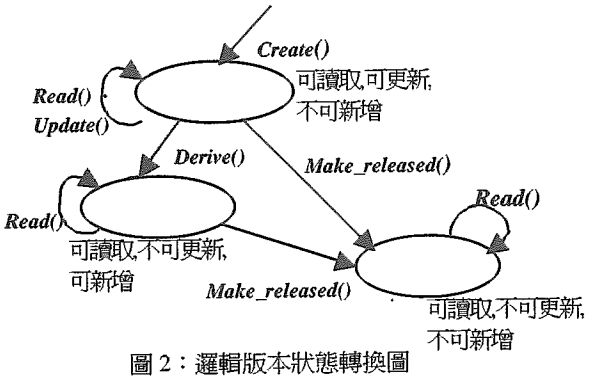

圖 2：邏輯版本狀態轉換圖

4.4 實體版本的狀態

我們的版本模型中，實體版本的 狀態有三種，分別爲 TT，RT 及 PO。交易中的更新運算在系統工作區中產生暫時的實體版本，此時的暫時物件的狀態爲 Transient Temporary (TT 狀態)，在交易完成前，不允許其他交易使用 TT 狀態下的暫時物件。當交易完成後，最終的暫時物件成爲 Released Temporary (RT 狀態)，可供其他交易讀取，藉以提高交易並行程度，而處於 TT 狀態的所有暫時物件則隨著交易結束而被清除。若系統將此 RT 狀態的暫時物件存回資料庫，則此暫時物件變成爲永久物件(Permanent Object)，其狀態成爲 PO 狀態，由於每一物件版本僅存在唯一的永久物件，所以，原 PO 狀態的永久物件退化爲 RT 狀態的暫時物件，可以被其他交易讀取。實體版本之狀態轉換如圖 3 所示。

另一方面，實體版本中的暫時物件是爲了增加讀取運算的並行能力而設計，隨著時間的延續，暫時物件也會隨之增多，因此，必須有一套暫時物件清理規則（purging rules），適時的清理系統工作區中的不必要的暫時物件，此清理規則將在第 5.2 節中討論。

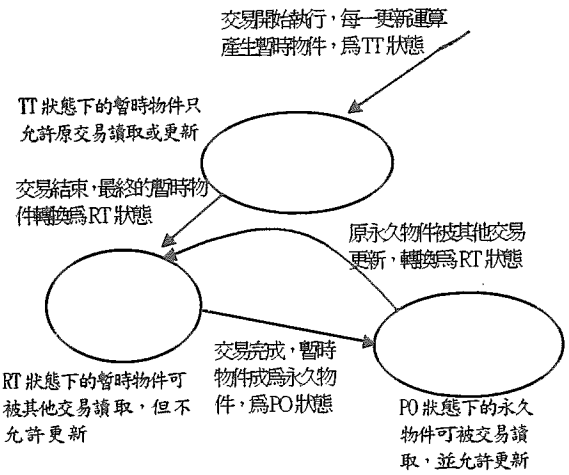

圖 3：實體版本的狀態轉換

4.5 版本運算

本節將討論版本模型所提供的讀取、更新、新增、刪除等運算。在描述這些運算之前，我們先利用數對(V_P,S_P)的概念來描述實體版本 V_P 的內容與目前狀態，利用集合 $H_P(V)$來描述邏輯版本 V 的內容與目前狀態。

定義 7□實體狀態數對□: 實體狀態數對(V_P,S_P)爲一數對，表示實體版本 V_P 目前處於狀態 S_P 中，$S_P \in$ { TT, RT, PO}，其中 TT、RT、PO 分別代表 Transient Temporary 狀態、Released Temporary 狀態與 Permanent Object 狀態。

由於只有 RT 或 PO 狀態的實體版本可供原交易外的其它交易使用，因此，我們用定義 8 描述邏輯版本中可共享的實體版本內容與狀態。

定義 8□邏輯版本的實體歷史□ 邏輯版本 V 的實體歷史 $H_P(V)$爲一個由實體狀態數對所構成的集合，描述目前邏輯版本 V 的組成元素（實體版本）及元素目前狀態，$H_P(V)$={ (V_P,S_P) | V_P 是邏輯版本 V 中的實體版本，$S_P \in$ { RT, PO}}。

例如，圖 1 中的(V_{PO}, PO)、(V_{T1}, TT)、(V_{T2},RT)爲版本 V_L 目前的實體狀態數對，而{(V_{PO}, PO)、(V_{T2},RT)}爲爲版本 V_L 的實體歷史。

以下，我們利用上述定義來描述讀取、更新、新增、刪除等運算。

若不違反並行規則，系統允許交易讀取物件在任何狀態下的邏輯版本。讀取（read）運算會在使用者指定的版本 V 中讀取適當的實體版本 V_{Pi}，讀取運算並不會影響邏輯版本或實體版本的狀態，對整個版本物件的內容不會有任何改變。亦即，若交易讀取的版 V，其讀取前

與讀取後的實體歷史分別爲 $H_P(V)$及 $H_P'(V)$，則 $H_P'(V)=H_P(V)$。然而，工作區中的版本，除了實體歷史中包含的實體版本（處於 RT 或 PO 狀態）外，尚有處於 TT 狀態下的暫時物件，這些物件只允許由產生這些物件的交易讀取或更新，不與其它交易共享。

若不違反並行規則，系統允許交易更新 Transient 狀態下的邏輯版本。更新（update）運算於交易結束後在實體版本集合中產生 RT 狀態的暫時物件供其他交易讀取，當系統允許此暫時物件儲存至資料庫時，此暫時物件會變成爲永久物件，而原來的永久物件則因爲被取代而成爲 RT 狀態的暫時物件。因此，更新運算對邏輯版本增加一個新的實體版本並更動相關實體版本的狀態，但對邏輯版本本身的狀態不會有影響。亦即，交易若將物件 A 呈 Transient 狀態的版本 V 的永久物件由 V_{PO} 更新至暫時物件 V_{TOn}，其更新前與更新後的實體歷史分別爲 $H_P(V)$ 及 $H_P'(V)$，則 $H_P'(V)=H_P(V)-\{(V_{PO},PO)\}\cup\{(V_{PO},RT)\}\cup\{(V_{TOn},PO)\}$。

至於新增運算只對處於 Working 狀態的邏輯版本有意義，它會先在版本物件中產生一新的邏輯版本，並將工作區中交易產生的暫時物件變成爲新的邏輯版本中的永久物件。新增運算會牽涉到兩個邏輯版本，它在原邏輯版本新增一暫時物件，並在新的邏輯版本中產生一永久物件。在交易完成後，原邏輯版本 V 的內容與狀態都不變，因此 $H_P'(V) = H_P(V)$，而新增的邏輯版本 V′ 其內容爲 V_{TOn}，V_{Ton} 變成爲永久物件，因此新增運算後之實體歷史 $H_P'(V') = \{(V_{TOn}, PO)\}$。

版本模型允許交易中的刪除運算刪除在 Transient 或 Working 狀態下的邏輯版本，工作區中其所對應的實體版本也一併被刪除。因此，邏輯版本與實體版本都不復存在，因此 $H_P'(V)=\phi$。

五、並行控制機制

爲結合上述的版本模型，使用邏輯版本管理版本物件，並使用實體版本之永久物件與暫時物件增加交易並行程度，我們提出一個新的並行控制機制。我們結合 Timestamp Ordering Protocol 及版本控制機制，提出我們的並行控制機制，稱之爲 Timestamp Ordering with Logical and Physical Versions（TOLPV）。首先，定義並行控制機制中的交易時戳及邏輯版本與實體版本寫入與讀取時戳。

5.1 時戳

資料庫系統賦予每一個進入系統的交易 X 唯一的交易時戳 T(X)，代表交易進入系統的順序，我們可以依據交易的時戳，定義版本模型中實體版本及邏輯版本的讀取與寫入時戳，藉由時戳順序的意義，讓交易得以並行執行。

定義 9□實體版本的寫入時戳□ 實體版本 V_P，其寫入時戳 $WT(V_P)$代表目前更新此實體版本 V_P 的所有交易中最大的交易時戳值。

定義 10□實體版本的讀取時戳□ 實體版本 V_P，其讀取時戳 $RT(V_P)$代表讀取此實體版本 V_P 的所有交易中最大的交易時戳值。

定義 11□邏輯版本的寫入時戳□ 邏輯版本 V_L，其寫入時戳 $WT(V_L)$爲 V_L 之實體版本集合中的永久物件的寫入時戳。

定義 12□邏輯版本的讀取時戳□ 邏輯版本 V_L，其讀取時戳 $RT(V_L)$爲 V_L 之實體版本集合中的永久物件的讀取時戳。

5.2 暫時物件清理規則

交易在執行更新或新增運算後會產生暫時物件，因此，系統工作區中的暫時物件會隨時間增長而增加，然而，系統中的某個暫時物件其寫入時戳如果小於所有正在執行的交易時戳，則此暫時物件是不可能會再有交易使用，因此，這些暫時物件就可以由系統工作區中清除。爲了適時的清理系統工作區中的不必要的暫時物件，並讓系統中的交易得以使用暫時物件增加並行能力，因此必須找到一個暫時物件 V_{TOr}，且不存在另一暫時物件 V_{TOi}，使得 $WT(V_{TOr})<WT(V_{TOi})<T_{min}(X)$，供系統中具有最小交易時戳 $T_{min}(X)$的交易 X 使用，而寫入時戳小於 $WP(V_{TOr})$的所有暫時物件都可以被清除。

5.3 TOLPV 並行控制規則

我們的版本模型中允許讀取、更新、新增與刪除四種運算，但因爲刪除運算之並行控制處理方式與更新運算相似，因此，我們將只以前三種運算來說明並行控制規則(Concurrency Control Rules)。以下是本文提出的 TOLPV 並行控制規則。

當交易 X 欲讀取(read)邏輯版本 V_L 時，先以其寫入時戳 $WT(V_L)$判斷是否 V_L 的永久物件 V_{PO} 可以供 X 讀取；若無法讀取，則在 V_L 對應的實體版本集合中尋找一個可供使用的暫時版本物件 V_{TOi}；若又不成功，則交易 X 必須回復(rolled back)，重新執行。

當交易 X 執行每一更新(update)運算，在工作區中會產生一暫時版本物件 V_{TOi}(屬於 TT 狀態)，當交易 X 完成時，最終的暫時物件 V_{TOn} 成爲 RT 狀態，再依據以下 TOLPV 規則，判斷此暫時物件是否可寫入資料庫中成爲永久物件。

而對於新增(derive)運算，系統的處理方式與寫入運算相似，但新增運算不會與其他交易有存取衝突的情形發生，所以系統對於交易的新增運算永遠允許其馬上執

行。TOLPV 並行控制規則演算法敘述如下：

(1) 讀取運算：當交易 X 要求讀取物件版本 V_L，有下列四種情形：
   (a) 若 $T(X)>WT(V_L)$，則允許 X 讀取 V_L 之永久物件 V_{PO}，且設定 $RT(V_L) \leftarrow \max(T(X), RT(V_L))$。
   (b) 若 $T(X)=WT(V_L)$，則允許 X 讀取 V_L 之永久物件 V_{PO}，或讀取屬於 TT 狀態且由交易 X 所產生的暫時物件，並設定 $RT(V_L) \leftarrow \max(T(X), RT(V_L))$。
   (c) 若 $T(X)<WT(V_L)$，則不允許 X 讀取 V_L 之永久物件 V_{PO}，但可在 V_L 的實體版本集合中，尋找一暫時物件 V_{TO_i}，其狀態爲 RT，滿足 $T(X)\geq WT(V_{TO_i})$，且不存在另一暫時物件 V_{TO_j}，使得 $WT(V_{TO_i}) < WT(V_{TO_j}) < T(X)$。
   (d) 若上述的暫時物件 V_{TO_i} 不存在，則 X 必須回復，重新執行。

(2) 更新運算：當交易 X 完成，要求更新版本 V_L 中的永久物件 V_{PO} 爲 V_{TO_n}，V_{TO_n} 現爲暫時物件，則有下列三種情形：
   (a) 若 $T(X)<RT(V_L)$，則不允許 X 更新物件版本 V_L，X 必須回復，重新執行。
   (b) 若 $T(X)<WT(V_L)$，則不允許 X 更新物件版本 V_L，X 必須回復，重新執行。
   (c) 若上述條件(a)、(b)都不成立，則允許 T 更新物件版本 V_L 之永久物件 V_{PO} 爲 V_{TO_n}，且設定 $WT(V_L) \leftarrow T(X)$，$RT(V_L) \leftarrow T(X)$，V_{TO_n} 轉換爲永久物件，而原先之 V_{PO} 則轉換爲暫時物件。

(3) 新增運算：當交易 X 要求衍生版本 V_L 成爲版本 V_L'，且以暫時物件 V_{TO_n} 爲 V_L' 之永久物件，則允許交易 X 衍生 V_L 到 V_L'，且設定 $WT(V_L') \leftarrow T(X)$，$RT(V_L') \leftarrow T(X)$，並將 V_{TO_n} 變成爲 V_L' 之永久物件。

5.4 TOLPV 的正確性

以下我們將討論 TOLPV 並行控制規則的正確性。首先，此版本模型中會有以下特性：

Lemma 1：邏輯版本 V_L 的寫入時戳 $WT(V_L) = \max(\{WT(V_P) \mid V_P$ 爲 V_L 的實體版本$\})$

Lemma：任一邏輯版本 V_L 的 $WT(V_L) \leq RT(V_L)$，任一實體版本 V_P 的 $WT(V_P) \leq RT(V_P)$。

Lemma：邏輯版本只有在 transient 狀態下，其實體版本集合中才會有暫時物件。邏輯版本在 Released 或 Working 狀態下，其實體版本集合中僅有唯一的永久物件，而無暫存物件。

定理 1：若 TOLPV 規則 1(c)的條件成立，則必存在一暫時物件 V_{TO_i}，使得 $T(X)\geq WT(V_{TO_i})$。

TOLPV 規則中，交易 X 若引用規則 1(c)後，代表資料庫中已有一交易 Y 更新過此版本，且 $T(X)<T(Y)$，因此，若交易 X 完成後，其更新運算將以更新過的內容取代永久物件，亦即，交易 Y 所更新的內容將被較早交易 X 的所更新的內容取代，造成資料不一致的現象。然而，依據 TOLPV 規則，取得暫時物件的交易 X 欲更新某版本的內容是不可能發生的。定理 2 描述此情形。

定理 2：若交易 X 中的讀取運算根據規則 1(c)取得暫時物件 V_{TO_i}，且交易 X 中另有更新運算，並在交易結束後產生暫時物件 V_{TO_n}，則 TOLPV 不會允許交易 X 將永久物件 V_{PO} 更新爲 V_{TO_n}，亦即，即使 TOLPV 允許交易讀取暫時版本 V_{TO_i}，系統並不會因而導致資料不一致的狀態。

另一方面，交易 X 欲讀取有存取衝突的物件時，可依 TOLPV 規則讀取物件版本 V_L 之某一暫時物件 V_{TO_i}，但依據 lemma ，物件 V_L 此時必處於 transient 狀態，版本模型不允許新增物件，所以不會有不一致的狀態發生。定理 3 爲其描述。

定理 3：若交易 X 根據規則 1(c)讀取暫時物件 V_{TO_i}，則交易 X 中的新增運算必不可能發生，因此不會造成不一致的狀態。

因此，依據定理 2 及定理 3，我們可以得知 TOLPV 的並行控制規則不因暫時物件的加入與使用而 破壞資料的一致性。因此：

定理 4：TOLPV 並行控制規則是正確的。

六、與 Ree 模型比較

AVANC 系統中的 Reed 模型結合系統層版本與 Timestamp Ordering 的方式達成多版並行控制機制。在本節中，我們將以兩交易 T_1 與 T_2 的各種可能的運算相互間的關係，比較 TOLPV 與 Ree 模型，由比較結果可知利用邏輯版本與實體版本的版本模型 除了在版本控制機制較 AVANCE 系統的一維版本管理方式爲佳外，在並行控制能力上，亦提供較佳的效能。

首先，在不失其一般性情況下，我們假設交易時戳 $T(T_1)<T(T_2)$，而 T_1 與 T_2 存取的物件版本爲 V_L（V_L 爲邏輯版本，而交易存取的目標物件爲其對應之實體版本集合中之永久物件 V_{PO}），V_L 之寫入、讀取時戳分別爲 $WT(V_L)$、$RT(V_L)$。

以下依交易之讀取(Read)、寫入(Write)運算發生先後次序之所有可能的八種相互關係，加以探討 TOLPV 之並行性，由於新增(Derive)運算永遠允許被執行，因此不必列入討論。爲了討論的明確性與獨立性，在各種情況中，必須假設 T_1 與 T_2 不會與其他交易有存取衝突發生，因此交易時戳（$T(T_1)$、$T(T_2)$）與寫入、讀取時戳（$WT(V_L)$、$RT(V_L)$）必須有適當的順序限制：

Case 1：T_1 讀取 V_L，T_2 讀取 V_L；（$WT(V_L)\leq T(T_1)$ 且

$WT(V_L) \le T(T_2)$)

因爲 $T(T_1) < T(T_2)$，故交易 T_1 先讀取版本 V_L（實際上讀取 V_L 之永久物件 V_{PO}），T_2 再讀取該版本，由於都是讀取運算，所以不會違反 Serializabilit 理論，因此兩交易都可順利執行其讀取運算。T_1、T_2 都引用規則 1(a)讀取 V_L。

Case 2：T_2 讀取 V_L，T_1 讀取 V_L；($WT(V_L) \le T(T_1)$ 且 $WT(V_L) \le T(T_2)$)

與 Case 1 相同，TOLPV 系統允許兩交易順利執行。T_2、T_1 都引用規則 1(a)讀取 V_L。

Case 3：T_1 讀取 V_L，T_2 更新 V_L；($WT(V_L) \le T(T_1)$ 且 $RT(V_L) \le T(T_2)$)

由於 $T(T_1) < T(T_2)$，所以不會破壞 Serializability，允許並行執行。T_1 引用規則 1(a)讀取 V_L，因此 $RT(V_L) = \max(T(T_1), RT(V_L)) < T(T_2)$，而 $WT(V_L) \le T(T_1) < T(T_2)$，因此，$T_2$ 會引用規則 2(c)更新 V_L。

Case 4：T_1 更新 V_L，T_2 讀取 V_L；($WT(V_L) \le T(T_1)$ 且 $RT(V_L) \le T(T_1)$)

由於 $T(T_1) < T(T_2)$，不會破壞 Serializability，允許並行執行。T_1 引用規則 2(c)更新 V_L，所以，$WT(V_L) \leftarrow T(T_1)$，因此，$T_2$ 引用規則 1(a)讀取 V_L。

Case 5：T_2 讀取 V_L，T_1 更新 V_L；($WT(V_L) \le T(T_2)$)

T_1 必須被回復，因此兩交易無法並行執行。T_2 引用規則 1(a)讀取 V_L，所以 $RT(V_L) \leftarrow \max(T(T_2), RT(V_L))$，因而 $RT(V_L) > T(T_1)$，因此 T_1 引用規則 2(a)回復。

Case 6：T_2 更新 V_L，T_1 讀取 V_L；($WT(V_L) \le T(T_2)$ 且 $RT(V_L) \le T(T_2)$)

T_1 可以讀取對應 V_L 之實體版本集合中的某一暫時物件，達到並行執行，且必可以找到一個暫時物件（定理 1）。T_2 引用規則 2(c)更新 V_L，T_1 引用規則 1(c)讀取 V_L 之暫時物件。此情況在 Ree 模型中，T_1 必須被回復。

Case 7：T_1 更新 V_L，T_2 更新 V_L；($WT(V_L) \le T(T_1)$ 且 $RT(V_L) \le T(T_1)$ 且 $WT(V_L) \le T(T_2)$ 且 $RT(V_L) \le T(T_2)$)

寫入依交易時戳順序執行，不會有寫入衝突，因此允許並行執行。T_1 先引用規則 2(c)更新 V_L，T_2 再引用規則 2(c)更新 V_L。

Case 8：T_2 更新 V_L，T_1 更新 V_L；($WT(V_L) \le T(T_1)$ 且 $RT(V_L) \le T(T_1)$ 且 $WT(V_L) \le T(T_2)$ 且 $RT(V_L) \le T(T_2)$)

T_1 必須被回復，因此兩交易無法並行執行。T_2 引用規則 2(c)更新 V_L，因此 $WT(V_L) \leftarrow T(T_2)$，因而造成 T_1 引用規則 2(b)回復。

由上述討論，TOLPV 之並行能力可以表一來顯示，其中 Y 表示允許並行執行，N 表示不允許並行執行，NA (not applicable)表示由於邏輯版本狀態的限制而不允許該運算執行，並非不允許並行執行；括號後的數字表示對應前述 Case 編號。Ree 模型的並行能力可以表二顯示，斜線或直線區域表示被 TOLPV 改進的部分。對照表一及表二，從 Case 1 到 Case 8，TOLPV 利用實體版本中的暫時物件改進了 Ree 模型中 Case 6 的情形。此外，Reed 模型並未提供新增(Derive)運算，我們利用新增運算，亦可以提高原有 Reed 模型對於寫入(Write)運算的並行能力。

後執行的運算		先執行的運算	T_1的運算			T_2的運算		
			Read	Update (Write)	Derive	Read	Update (Write)	Derive
T_1的運算	Read					Y (2)	Y (6)	Y
	Update (Write)					N (5)	N (8)	NA
	Derive					Y	NA	Y
T_2的運算	Read		Y (1)	Y (4)	Y			
	Update (Write)		Y (3)	Y (7)	NA			
	Derive		Y	NA	Y			

表 1：TOLPV 的並行能力表

後執行的交易		先執行的運算	T_1的運算		T_2的運算	
			Read	Write	Read	Write
T_1的運算	Read				Y	N
	Write				N	N
T_2的運算	Read		Y	Y		
	Write		Y	Y		

表 2：Ree 模型的並行能力表

爲了利用版本控制機制來管理版本物件，並且利用工作區中由交易產生的暫時物件以增加交易執行的並行程度，在本研究提出的版本模型中，版本物件之管理以其版本衍生圖中的邏輯版本爲主，邏輯版本是屬於應用層次版本，可依不同應用之需要做二維延伸，此二維的版本衍生（branch version）較之 AVANCE 系統中 Ree 模型的一維版本衍生（linear version）適合更多的應用，因爲 Reed 模型之版本純爲並行控制考量，就邏輯意義

而言，當新版本產生時，原來的版本就已消失（因一維版本衍生）。對每一邏輯版本，將其實體版本集合中之物件區分爲永久物件與暫時物件，我們的 TOLPV 並行控制機制利用這兩種不同的實體版本，結合 Timestamp Ordering，較 Ree 模型提供更佳的交易並行能力。

七、結論與未來工作方向

本研究利用物件版本與並行控制結合，增強物件導向資料庫系統之效能。首先，我們提出一個適當的版本模型，提供版本控制機制使用的邏輯版本與儲存在工作區的實體版本。其次，利用實體版本中的永久物件與暫時物件的多版特性以增加交易並行執行的能力。我們提出一個結合此版本模型與 Timestamp Ordering 並行控制協定的新並行控制方法 TOLPV，並分析其效能。TOLPV 在版本模型上採用二維管理模式，較 AVANCE 系統之 Ree 模型提供更佳的版本管理方式；而在並行控制程度上，根據第六節分析結果顯示，TOLPV 亦提供更佳的並行能力。

在未來的研究方向上，我們將繼續以其他已存在的並行控制機制，例如 Two-Phase Locking Protocol、Optimistic 並行控制方法與此版本模型結合以分析其可行性、正確性及交易並行能力。

國科會計劃編號□NSC88-2213-E005-002

參考文獻

[1] P. Bernstein and N. Goodman, "Multiversion concurrency control – theory and algorithms," *ACM Transactions on Database Systems*, Vol. 8, No. 4, 1983, pp. 465-483.

[2] M. Carey and W. Muhanna, "The performance of multiversion concurrency control algorithms," *ACM Transactions on Computer Systems*, Vol. 4, No. 4, 1986, pp. 338-378.

[3] K. P. Eswaran, J. N. Gray, R. A. Lorie and I. L. Traiger, "The notion of consistency and predicate locks in a database system," *Communication of the ACM*, Vol. 19, No. 11, 1976, pp. 624-633.

[4] H. T. Kung and J. T. Robinson, "On optimistic methods for concurrency control," *ACM Transactions on Database Systems*, Vol. 6, No. 2, 1981, pp. 312-326.

[5] C. H. Papadimitriou, P. A. Bernstein and J. B. Rothnie, "Some computational problems related to database concurrency control," *Proceedings of the Conference on Theoretical Computer Science*, 1977, pp. 275-282.

[6] C. H. Papadimitriou, "The serializability of concurrent database updates," *Journal of the ACM*, Vol. 26, No. 4, 1979, pp. 631-653.

[7] C. Oussalah and G. Talens and M. Colinas, "Concepts and Methods for Version Modelling", *Proceedings of IEEE Design Automation Conference*, 1993, pp. 332-337.

[8] R. Rastogi, S. Seshadri, P. Bohannon, D. Leinbaugh, A. Sliberschatz and S. Sudarshan, "Logical and physical versioning in main memory databases," *Proceedings of the 23rd VLDB Conference*, 1997, pp. 86-95.

[9] D. Reed, "Implementing atomic actions on decentralized data," *ACM Transactions on Computer Systems*, Vol. 1, No. 1, 1983, pp. 3-23.

[10] D. Reed, *Naming and Synchronization in a Decentralized Computer System*, MIT/LCS/TR-205, MIT Laboratory for Computer Science, Cambridge, MA, 1978.

[11] I. L. Traiger, J. H. Gray, C. A. Galtier and B. G. Lindsay, "Transactions and consistency in distributed databases systems," *ACM Transactions on Database Systems*, Vol. 7, No. 3, 1982, pp. 323-342.

[12] 馮懷君，*一個新的物件導向資料庫版本控制模*碩士論文，國立中興大學，資訊科學研究所，1998.