15

分散式系統的理論與實務

15.1 分散式運算系統的基本概念
15.2 事件的排序問題
15.3 互斥 (mutual exclusion)
15.4 分散式的單元性 (atomicity)
15.5 分散式的並行控制 (concurrency control)
15.6 分散式的死結處理 (deadlock handling)
15.7 分散式演算法的常見問題
15.8 以 CORBA 為例來認識分散式系統
15.9 雲端運算的發展與分散式運算

要合作就必須先溝通與協調，合作的力量大，但是溝通與協調可是一門大學問。這就是分散式運算問題的來源，分散式運算環境中合作的對象是分散各地的電腦，這些電腦透過網路連接在一起，每一台電腦都獨立運作，但是也能藉由分散式系統來進行合作。由於各種電腦系統存在著異質性 (heterogeneity)，合作之前必須先協調解決差異，規範出合作的方法來。

碁峰 www.gotop.com.tw

15.1 分散式運算系統的基本規格

前面提到的作業系統主要以單一的電腦為作業的環境，由於電腦硬體的成本降低，而且網路已經相當地普及，軟體系統執行時環境已經逐漸變成所謂的分散式運算 (distributed computing) 的架構，圖 15-1 顯示兩大類型的分散式運算環境，要使資源的分配與管理得到妥善的安排，同樣需要複雜的系統軟體。

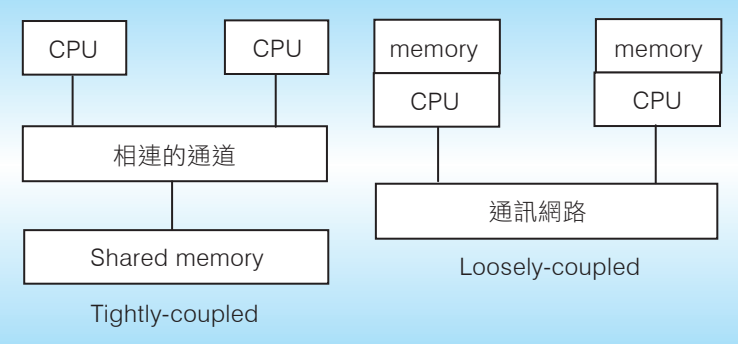

圖15-1 分散式運算的架構

1. 緊密結合的 (tightly-couple) 分散式運算架構 ： 多個 CPU 共享記憶體，可做為彼此之間溝通的基礎，這一類的系統也常稱為平行處理系統 (parallel processing systems)，相連的通道有極快的傳輸速率，適合支援大型的複雜運算。
2. 鬆散結合的 (loosely-coupled) 分散式運算架構 ： 多台電腦經由網路連接起來，因此主要的溝通方式是訊息的交換 (message passing)。常聽到網路作業系統 (Network operating system) 就用在這一類的環境中。

緊密結合的分散式系統受限於共享的記憶體容量與頻寬，不太容易擴充，而且有區域性範圍的限制。鬆散結合的分散式系統則有極大的擴充彈性，涵蓋的區域大，因此，所謂的「分散式系統」(distributed system)，所指的主要是鬆散式結合的分散式運算環境。由於運算的架構上有這些差異，在軟體系統的層次上自然會有一些異質性。

1. 平行系統 (Parallel systems)： 由於平行處理的架構種類很多，軟體系統的設計和架構的特徵關係密切，比較難建立一個通用性的系統。
2. 網路作業系統 (NOS, Network Operating System)： 如圖 15-2 的情況，每台電腦都有很高的自主性 (autonomy)，但可透過 NOS 來共用資源與交換訊息。
3. 分散式作業系統 (DOS, Distributed Operating System)： 如圖 15-3 的情況，DOS 試著降低各電腦的自主性，讓使用者感覺到所有的資源都在單一作業系統的控管下，有點廣域透明化 (wide-area transparency) 的味道，也就是說底下

碁峰 www.gotop.com.tw

分散的細節對於使用者來說是透明的，不用去了解。像 OSF DCE(Distributed Computing Environment) 就可以算是一種 DOS。

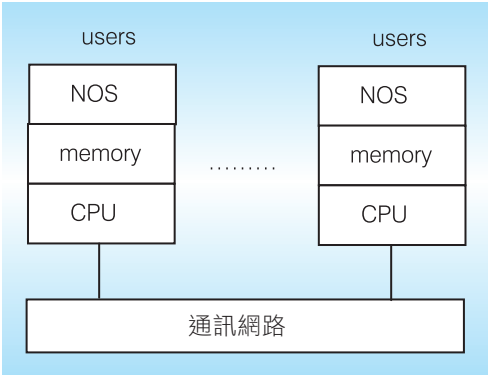

圖15-2 NOS的架構

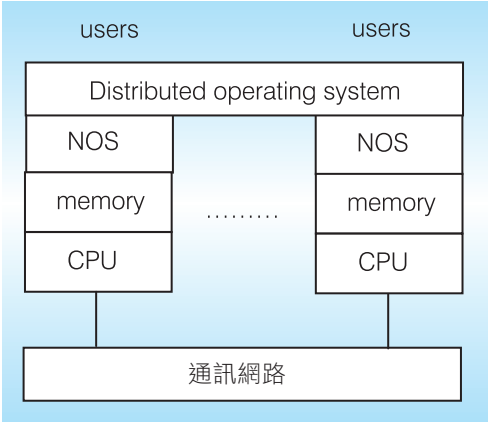

圖15-3 DOS的架構

15.1.1 分散式運算的涵義

分散式檔案系統只是分散式系統的一個例子，事實上，在分散的前題之下，前面所探討的各種作業系統的問題，在分散式系統中已經變成了另一大類的主題，所以很多都是作業系統延伸出來的問題，不過由於分散式系統的硬體架構比較複雜，所以問題也變難了，下面列舉一些分散式系統中探討的問題：

1. 分散式的處理元管理 (Distributed process management)。
2. 訊息傳遞的方法 (Message passing)。
3. 分散式的記憶體管理 (Distributed memory management)。
4. 分散式的處理元管理。

碁峰 www.gotop.com.tw

我們前面曾經介紹過分散式運算的架構，由於具有處理能力的電腦很多，改善運算效能的關鍵就在於如何讓運算能同時在多台電腦上執行，這牽涉到把一件循序執行的工作分割成可並行的工作，我們下面介紹兩種分割的方式 ：

1. 基本的二分法 ： 最直覺的分割方式是將位址空間 (address space) 分成功能 (function) 與資料兩大部分，如圖 15-4 中的第 1 個階段 。
2. 功能主導的分割 (Functional partitioning)： 讓資料在相同的位址空間中，運算本身分割開來，如圖 15-4 中的第 2 個階段 。
3. 資料主導的分割 (Data partitioning)： 讓資料分割開來，交給相同的程式去處理，當然各程式使用個別的位址空間 。 如圖 15-4 中的第 3 個階段 。

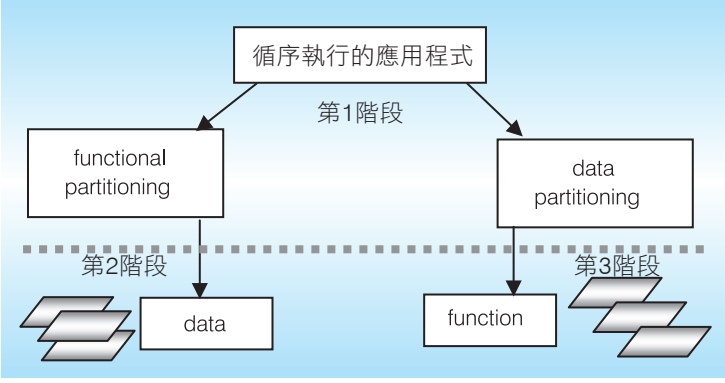

圖15-4 工作分割的方式

我們可以再簡單地說明一下圖 15-4 顯示的運算分割方式，首先，位址空間 (address space) 可以分成程式碼和資料兩部分，至於運算 (computation) 的部分，通常有兩種分割方式，一種是讓資料在相同的位址空間中，運算本身分割開來，在個別獨立的電腦上執行 。 另外一種是讓資料分割開來，交給相同的程式去處理，當然各程式使用個別的位址空間 。 要支援上述的這些分割方法，作業系統本身得具備一些基本的功能 ：

1. 分散式環境中處理元的產生與結束 。
2. 處理元的排程 (scheduling) 與定位 。
3. 同步 (synchronization) 的問題 。
4. 死結 (deadlock) 的管理 。

在分散式的處理元管理中，為了能提昇系統整體的效能，處理元可能要在處理器之間移轉，我們把這種程序稱為處理元的轉置 (process migration) ，目的是要讓空閒一點的處理器多做一點工作，達到所謂負載均衡 (load balancing) 的目標，由於處理元的轉置會損耗一些資源，所以整體效能如何改善仍有待深入的演算法來解決 。

15.1.2 訊息傳遞的方法(Message passing)

分散式系統中訊息傳遞 (message passing) 是主要的溝通方法，為什麼要傳遞訊息呢？這跟應用系統的需求有關，分散式應用系統之間會依照某種協定來溝通，協定本身通常會規定各種訊息的格式與用途。訊息本身通常具有兩種功能：

1. 處理元之間分享資料的媒介。
2. 支援同步 (synchronization) 的機制。

圖 15-5 畫出分散式系統中訊息傳遞的方法，可能大家會覺得很奇怪，為什麼要那麼麻煩呢？其實這跟網路的層次化模型 (layered model) 有關，ISO 的 OSI 模型就是一個有名的例子，作業系統必須負責下層的資料的處理，應用層次則需要有處理訊息的介面。我們在本章後面介紹的 socket interface、RPC、CORBA 與 messaging service 都是分散式系統中訊息傳遞的方法。

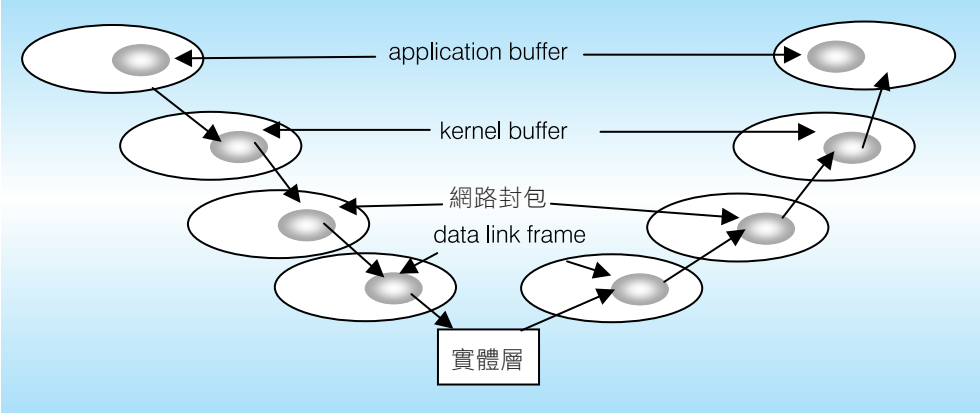

圖15-5 訊息複製的作業

15.1.3 分散式的記憶體管理

分散式的記憶體管理有不少理論上的發展，透過虛擬記憶體 (virtual memory) 的方式，或是記憶體管理介面，都可以進行遠端記憶體空間的存取。圖 15-6 畫出記憶體的一般介面，遠端的儲存空間還是可以透過檔案系統來進行。圖 15-7 畫出遠端記憶體的模型，注意這時候處理元可以經由遠端記憶體介面 (remote memory interface) 來使用遠端的記憶體。當然，分散式記憶體的管理方法不只一種，以 CORBA 來說，可以完全以物件的方式來處理，將分散式物件 (distributed objects) 看成是一種分散式記憶體共享的特殊情況。另外，分散式虛擬記憶體 (distributed virtual memory) 也可以用來處理遠端的位址空間 (remote address space)。

www.gotop.com.tw

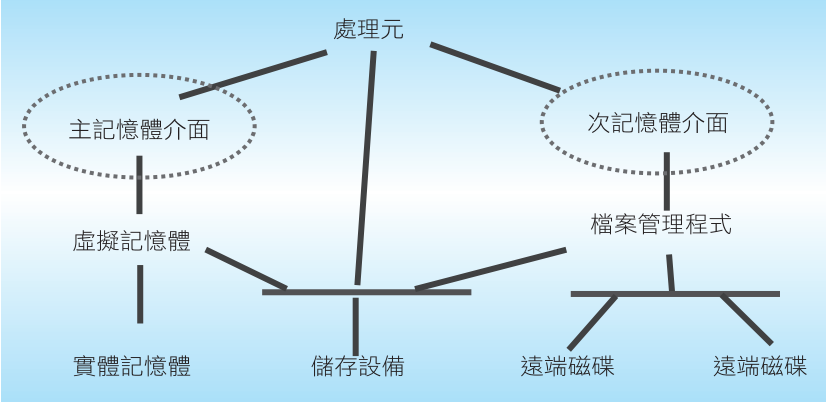

圖15-6 記憶體的一般介面

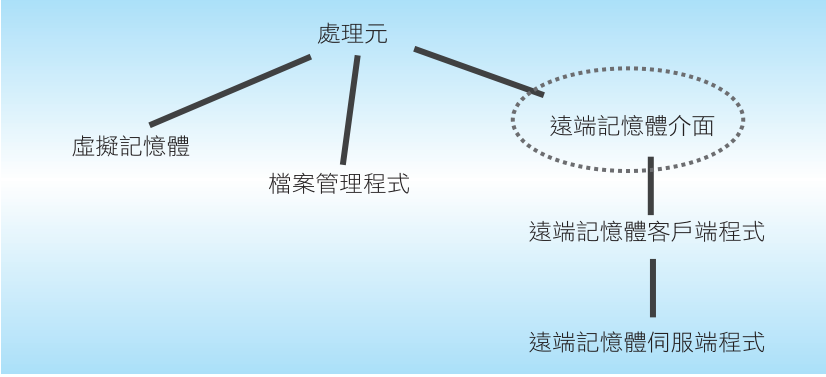

圖15-7 遠端記憶體的模型

15.2 事件的排序問題

我們下面以事件的排序問題為例，來看分散式系統在理論上的一些發展。在單一的電腦系統上，我們可以由系統時鐘 (clock) 來決定兩個事件 (event) 發生的先後順序，例如處理元需要使用某個資源之前，必須確定已經擁有使用權，所以使用資源的事件一定得發生在擁有資源使用權的事件之後。在分散式系統中沒有單一的時鐘與共用的記憶體，但是資源還是一樣要共享，在這種情況下就要另外想辦法來決定兩個事件發生的先後順序。

15.2.1 之前發生的關係(happened-before relation)

在理論上必須對於基本的觀念與假設有精確的定義，後續的探討才會有嚴謹的邏輯可循，所以大家不要小看事件排序的問題。首先定義所謂的之前發生的關係 (happened-before relation)：

碁峰 www.gotop.com.tw

1. 假設 A 與 B 是同一個處理元中的事件，而且 A 在 B 之前執行，則 A 與 B 有 A 在 B 之前發生的關係，可用 A→B 來表示。
2. 假設 A 是某個處理元送出訊息的事件，而 B 是另一個處理元收到訊息的事件，則 A 與 B 有 A 在 B 之前發生的關係，可用 A→B 來表示。
3. 假如 A→B 與 B→C 成立，則 A→C 也成立。

假如兩個事件之間沒有存在任何之前發生的關係，則我們可以認定這兩個事件是同時執行的 (executed concurrently)，當然實際上任兩個事件還是有執行的先後，但是只要從之前發生的關係中無法引申出發生先後的關係，我們可以不用去管實際發生的先後。

15.2.2 決定事件發生先後的方法

為了要在沒有 clock 的情況下能決定事件發生先後，我們讓每一個事件都保持一個時間戳記 (timestamp)，然後規定當事件 A 與事件 B 有之前發生的關係時，則事件 A 的時間戳記小於事件 B 的時間戳記。演算法本身有兩個重點：

1. 每個處理元 Pi 有一個邏輯時鐘 (logical clock) LCi，通常可以用計數 (counter) 的方式來表示邏輯時鐘。
2. 當處理元 Pj 收到 Pi 送來的訊息時，若是發現 LCi(A)>LCj(B)，A 代表 Pi 送出訊息的事件，B 代表 Pj 收到訊息的事件，則令 LCj(B) = LCi(A)+1。

上面的方式可以達到分散式系統中事件的完全排序 (total ordering)，假如任意兩個事件的時間戳記相同，可以用處理元的辨識號碼來決定發生的先後次序。類似的演算法都可以經由證明來確認其正確性。

15.3 互斥(mutual exclusion)

互斥的觀念在單機的作業系統中是很重要的，分散式系統也有類似的問題。假設分散式系統中有 n 個處理元，每個處理元都位於一台電腦上執行，我們下面介紹 3 種達到這些處理元之間互斥的演算法。

1. 集中式的演算法。
2. 完全分散式的演算法。
3. 註記傳遞 (token-passing) 的演算法。

碁峰 www.gotop.com.tw

15.3.1 集中式的演算法

由其中一個處理元來協調進入臨界區域 (critical section) 的權力，希望啟動互斥的處理元送出請求 (request) 給負責協調的處理元，等收到協調處理元的回應 (reply) 之後，才可以進入臨界區域。要出臨界區域時，送釋出 (release) 的訊息給協調處理元。

協調處理元這邊在收到請求訊息時，會檢視是否已經有處理元進入臨界區域，假如沒有則立即送出回應訊息，假如有則將請求先放到佇列 (queue) 中。協調處理元決定排程時可以用像先來先做 (first-come, first-served) 的政策，避免發生飢餓 (starvation) 的情況。在集中式的演算法中，假如協調處理元的執行失敗 (fail)，則可透過選擇的演算法 (election algorithm) 來選出一個新的協調處理元，重建請求佇列 (request queue)。

15.3.2 完全分散式的演算法

當處理元 Pi 要進入臨界區域時，先產生一個時間戳記 TS，然後把請求訊息 (Pi, TS) 送給包括自己在內的所有處理元，當處理元收到該請求訊息時，若是自己不在臨界區域內，可立即送回一個回應 (reply)，否則就暫緩 (defer) 回應。

假如一個處理元收到了所有處理元送出的回應訊息，該處理元可以馬上進入臨界區域，等完成後要出臨界區域時，再送出回應訊息給原先暫緩請求的處理元。處理元在決定收到請求訊息 (Pi, TS) 後是否送出回應時有 3 個考量的因素：

1. 假如處理元正在臨界區域中，則暫緩 (defer) 送出回應。
2. 假如處理元並沒有要進入臨界區域，則馬上送出回應。
3. 假如處理元也想要進入臨界區域但還未進入，則比較自己的請求的時間戳記 TSj 與外來請求的時間戳記 TSi，若 TSj>TSi 則馬上送出回應。否則暫緩 (defer) 送出回應。

這個演算法有以下幾個優點：

1. 可以達成互斥 (mutual exclusion)。
2. 可避免死結 (deadlock) 的發生。
3. 可避免飢餓 (starvation) 的發生，因為使用了時間戳記，可採取先來先做 (first-come, first-served) 的政策。
4. 需要傳送的溝通訊息不算多。

碁峰 www.gotop.com.tw

當然這個演算法本身也有一些缺點：

1. 處理元必須知道所有其他處理元的辨識碼，假如有其他的處理元加入，則新處理元的資料要送給現有的處理元，新處理元自己也要建立有關於其他處理元的資料。
2. 假如有處理元執行失敗 (fail)，演算法就失控了。
3. 處理元太多的時候，會造成處理元的執行經常中斷。

15.3.3 註記傳遞(token-passing) 的演算法

在處理元之間傳遞註記 (token)，擁有註記的處理元就可以進入臨界區域，同一時間只會有一個處理元擁有註記，因此可以達到互斥。我們可以在邏輯上把所有的處理元看成形成一個環狀 (ring) 結構，註記在結構中輪流傳送，假如註記遺失或是處理元執行失敗，必須有演算法重建。很多分散式系統中的演算法就像上面介紹的一樣，有訊息的傳送與各處理元之間的互動，大家有興趣的話，不妨試著證明看看上述的這些演算法的確可以得到正確的結果。

15.4 分散式的單元性(atomicity)

在分散式系統中，所有執行一個交易 T 的節點必須對於 T 的執行結果有一致的看法。假如 T 確認 (commit) 的話，必須在所有執行 T 的節點上都 commit。假如 T 取消 (abort) 的話，必須在所有執行 T 的節點上都 abort。我們需要確認的協定 (commit protocol) 來達到這樣的要求。常見的有兩階段確認的協定 (two-phase commit protocol)：

第1階段(Phase 1)

1. 將 <prepare T> 記錄寫入 log 中，而且讓 log 寫入磁碟。
2. 將 (prepare T) 的訊息送往所有執行 T 的節點。
3. 當節點收到 (prepare T) 的訊息時，先檢驗是否要確認 (commit) 自己所執行的部分，假如要確認的話，先把 <ready T> 的記錄寫入 log，而且讓 log 寫入磁碟，然後回覆 (ready T) 的訊息。假如不要確認，將 <no T> 的記錄寫入 log，然後回覆 (abort T) 的訊息。

碁峰 www.gotop.com.tw

第2階段(Phase 2)

1. 送出 (prepare T) 的節點從所有的節點收到回覆以後，或是等待超過一段時間以後，該節點就可以決定是否要 commit 或 abort。
2. 假如從所有的節點收到的回覆是 (ready T)，則 T 可以 commit，否則 T 就要被 abort。
3. 將 <commit T> 或 <abort T> 的記錄寫到 log，而且讓 log 寫入磁碟。
4. 把 (commit T) 或 (abort T) 的訊息送往所有的節點，收到訊息的節點將訊息寫入 log。

two-phase commit protocol 是相當有名的協定，雖然交易處理有很多理論性的發展，仍然有很多複雜的問題不完全是理論上能提供很好的解決辦法的，在分散式系統的領域中，同樣的問題會變得更複雜！

15.5 分散式的並行控制(concurrency control)

程序在分散式系統中會相互溝通，當這些程序要一起完成一項工作時，必須有更進一步的協調合作，所謂「同步的控制」(Synchronization control) 就是指協調合作的機制。在單機系統中，很多協調合作可經由共享記憶體 (Shared memory) 的方式來達成，在分散式系統中，問題就變得相當複雜了，我們下面就從幾個不同的角度來看分散式系統的同步控制。做到同步以後就能支援並行的作業 (concurrency)。

15.5.1 時鐘的同步(Clock synchronization)

每台電腦上都有一個系統時鐘，在分散式系統中，每個系統時鐘之間都會有一些微小的差異，為了讓分散式系統的功能不受這些差異的影響，有所謂的「時鐘同步演算法」(Clock Synchronization algorithm) 來使分散式系統中各電腦上的時鐘讀數一致。事實上，所謂的時鐘可分成邏輯時鐘 (Logical clock) 與實體時鐘 (Physical clock)，邏輯時鐘強調發生的時間順序，是相對的時差，實體時鐘指示的就是一般的時間觀念。分散式系統需要那一種時鐘決定於各種程式本身的要求。

15.5.2 互斥(Mutual exclusion)

當多個程序同時使用相同的資料時，必須確定不會產生和程序邏輯相異的結果，在單機系統中最常見的方法是使用一些共用的資料結構，例如 Semaphore，來做為控制的基礎，確定當某個程序在使用該資料時，其他程序無法使用相同的資料。類

www.gotop.com.tw

似的方法雖然也可以應用在分散式系統中，控制資源的共用，但是控制的程式在某一台電腦上執行，萬一該電腦故障，即使資源可用，其他程序仍舊無法使用。因此，互斥 (Mutual exclusion) 在分散式系統中也可以寫成分散式的演算法 (Distributed algorithm)，使用電腦故障而停頓的機率降低。

15.5.3 交易異動模型(Transaction model)

交易處理要求產生的結果是正確的，以圖 15-8 中的兩個程序為例，P1 與 P2 個別執行時，可以得到符合程式邏輯的正確結果，在分散式系統中，P1 與 P2 也有可能同時執行，在不做任何管制下，有可能得到像圖 15-8 右邊的不正確結果。為了防止這種情況，在分散式系統中必須要求任何交易異動具有 ACID 的特性：

1. A 代表 Atomic：任何交易異動的結果，必須和其依序執行的結果一樣，否則就算失敗，所以在分散式系統中仍然可以平行執行多個交易異動，只是系統必須保證其結果與個別依序執行的結果相同。
2. C 代表 Consistent：交易異動的結果不違反系統的一些法則，例如銀行系統有收支平衡的法則，交易異動完成後就不應該造成不平衡。
3. I 代表 Isolated：多個交易異動同時處理，即時彼此間有共用資料，得到的結果與分開個別執行的結果一樣。
4. D 代表 Durable：一旦交易異動成功結束，其結果造成的改變是永久的。

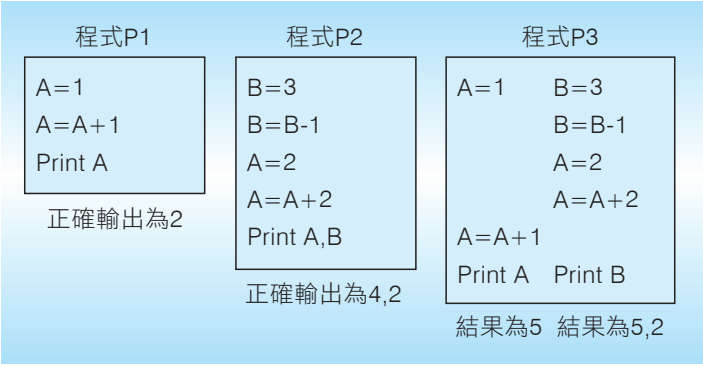

圖 15-8 交易異動處理的問題

碁峰 www.gotop.com.tw

15.6 分散式的死結處理(deadlock handling)

分散式系統中資源共享，程序可同時執行，也因而造成了死結的問題，圖 15-6 的第 I 種情況表示程序 P1 送訊息給 P2 並等待回應，P2 送訊息給 P3 並等待回應，P3 送訊息給 P1 並等待回應，由於 P1、 P2、 P3 都在等待回應，也就不會先回應所收到的訊息，因而造成死結。圖 15-9 中的第 II 種情況是由於共享資源而造成死結，程序 P1 請求使用 R2 的資源，此資源正被 P2 占用，P2 所請求使用 R1 的資源，則正被 P1 使用中，由於 P1 與 P2 都得不到所要的資源，無法繼續執行，也無法釋放所占有的資源。

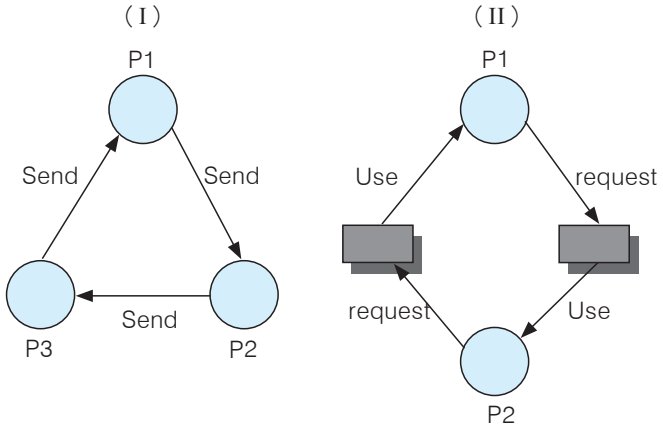

圖 15-9 死結的產生

分散式系統中有各種解決死結問題的方法，包括死結的偵測 (Deadlock detection) 與死結的預防 (Deadlock prevention)、死結的避免 (Deadlock avoidance) 等。死結的偵測是在死結發生後，試著以演算法偵測出來，再加以復原；死結的預防則是事先在系統設計上就想辦法不讓死結發生；死結的避免則試著在分配資源時仔細處理，避免死結的發生。

15.7 分散式演算法的常見問題

有許多分散式的演算法都運用所謂的協調處理元 (coordinator process)，假如協調處理元無法繼續執行，則必須選出新的協調處理元，這時候要透過選舉演算法 (election algorithm) 來找出這樣的處理元來。

另外一個常見的問題是合作的處理元之間如何對於特定值尋求一致的看法 (reaching agreement)，主要是因為分散式系統中處理元要透過通訊來溝通，可能碰到

碁峰 www.gotop.com.tw

通訊不良或是處理元發生錯誤的情況，造成無法確認正確的值。這個問題跟拜占庭將軍的問題 (Byzantine generals problem) 類似，也就是在拜占庭陸軍的各部隊指標官之間需要透過傳令兵的傳訊來達到一起發動攻擊的命令。

15.8 以CORBA為例來認識分散式系統

CORBA 是分散式應用的一種架構與規格，由 OMG(Object Management Group) 所主導發展出來的。完整的 CORBA 規程文件可以參考 www.omg.org 網站上的資料，在深入介紹 CORBA 以前，我們先來導正幾個有關於 CORBA 的觀念。

1. CORBA 是一種規格，CORBA 不是程式語言，CORBA 所定義的是一種應用的架構，可以簡化分散式物件的開發。
2. CORBA 算是一種標準化的規格，但是支援 CORBA 的軟體不只一種。
3. CORBA 與 RMI 可以適當地結合使用，並不見得一定要二者擇一來使用。

15.8.1 CORBA的基本觀念

在 n-tier 的應用中，通常都會把 clients 與 objects 分開，讓他們分別屬於不同的 tier，在溝通上必須以物件的 interface 來規範，由於要容許 clients 與 objects 使用不同的程式語言，所以 CORBA 使用一種與任何程式語言都無關的中介語言 (IDL, interface definition language) 來定義 clients 與 objects 之間的關係，然後自動產生與 clients 或 objects 結合的程式碼，圖 15-10 畫出 CORBA 的基本原理，IDL 檔案在 client 端產生 stub 檔案，在 object 端產生 skeleton 檔案，看起來 client 與 stub 結合成通訊的一端，object 與 skeleton 結合成另一端，stub 與 skeleton 處理了一些通訊與資料轉換上的細節，所以 client 端中的 client 程式與 stub 的溝通看起來好像只是依照 IDL 檔案的定義來進行的，其實很多細節由 stub 的程式碼解決了，object 端也是一樣，由 skeleton 負責處理了一些額外的工作，結果是 client 與 object 的設計者只管把 IDL 檔案定義清楚就好了。

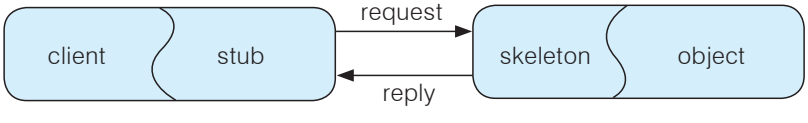

圖15-10 stub與skeleton的觀念

碁峰 www.gotop.com.tw

```
命令提示字元
C:\flag\ch12\RMI-IIOP-CORBA\CORBA\client>javac *.java

C:\flag\ch12\RMI-IIOP-CORBA\CORBA\client>java HelloWorldCORBAClient HelloWorldCO
RBA -ORBInitialPort 900
CORBA Client : The response is 'Hello there from CORBA!'
```

圖15-21 client端的處理

```
命令提示字元
C:\flag\ch12\RMI-IIOP-CORBA\RMI>java -Djava.naming.factory.initial=com.sun.jndi.
cosnaming.CNCtxFactory -Djava.naming.provider.url=iiop://localhost:900 HelloWorl
dRMIClient2 HelloWorldCORBA
RMI-IIOP Client : 得到的回應為  'Hello there from CORBA!'


C:\flag\ch12\RMI-IIOP-CORBA\RMI>
```

圖15-22 RMI-IIOP client和HelloWorldCORBAServer程式溝通

本章一路介紹的範例包括 CORBA client 應用、RMI client 應用、CORBA object server 與 RMI object server，從圖 15-23 可以看到這些 client 與 server 程式的互通性，有興趣的話可以用程式配對做實際的測試，當然原理在前面已經說明了，也就是 IIOP 的功用。

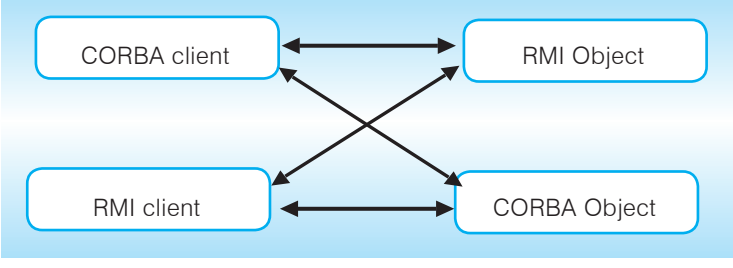

圖15-23 RMI-IIOP與CORBA的互通

15.9 雲端運算的發展與分散式運算

「雲端運算 (cloud computing)」是大家經常耳聞的科技名詞，但是對於它的涵義好像始終都無法得到一個確切的答案。古人曾說：「只緣身在此山中，雲深不知處。」只要讓我們的生活更方便，不管在山裡還是雲裡，大概也沒有那麼重要了。

15.9.1運算平台(computing paradigm)的變革

碁峰 www.gotop.com.tw

平常把網路線插入連接網路的插座就可以上網，或是把電源插頭插入插座就能供給電源，但是我們並不需要知道這些插座連到哪裡去，或是供電與網路裝置是如何安

裝與作用的。雲端服務也運用了類似的概念，提供方便穩定的服務，但是使用者不需要知道是怎麼辦到的。

開放原始碼軟體可以讓雲端運算的技術加速進展，包括雲端運算的業者運用開放原始碼軟體來建立雲端架構，或是在雲端服務之上建立與提供開放原始碼軟體，也有人透過開放原始碼軟體來提供雲端服務。開放原始碼軟體可以免費自由下載使用，增加了大家使用的意願，假如能跟雲端技術結合，勢必會對未來的運算平台產生重大的影響。

我們可以從兩個角度來看雲端的開放資源，一種是跟雲端運算的建置及技術有關的一些開放原始碼的軟體，另外一種則是雲端業者所提供的一些開放資源，「開放」就是指這些資源是免費的。

從所謂的運算平台 (computing paradigm) 的變革可以看到人類運用運算資源方式的改變，圖 15-24 顯示以下 3 個階段：

1. 大型主機運算 (mainframe computing)：使用者透過功能簡單與價格便宜的終端機 (terminal) 來分享大型主機 (mainframe) 的運算資源。
2. 個人電腦運算 (PC computing)：個人電腦的效能提昇，足以應付個人的運算需求。
3. 網路運算 (network computing)：網路普及使得個人電腦與伺服器能透過網路相連，分享資源。

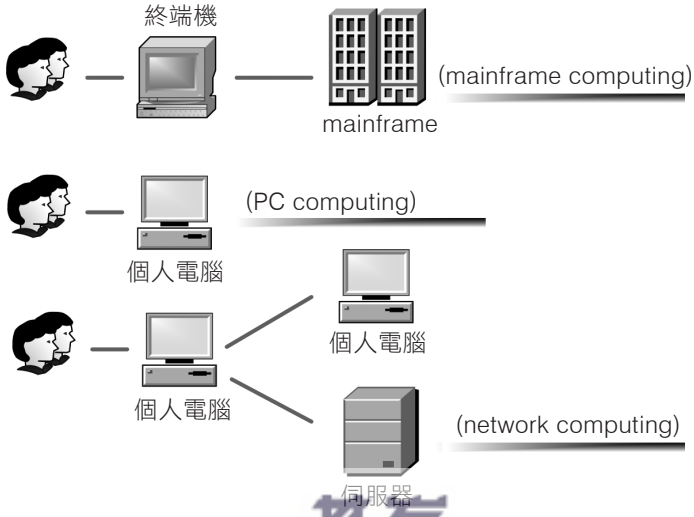

圖15-24 運算架構的變遷 (I)

碁峰 www.gotop.com.tw

2. 網際網路運算 (Internet computing)： 圖 15-25 顯示的是網際網路運算，讓使用者透過個人電腦上網，連接並使用其他伺服器上的運算資源，TCP/IP 的網路協定是促成網際網路運算普及的關鍵 。

3. 網格運算 (grid computing)： 網格運算也集合了眾多電腦的資源，但是跟傳統的平行運算架構不太一樣，電腦之間不是靠高速的線材相連，而是直接透過網路連接，反而比較像是經由中介軟體協調合作的分散式系統 (distributed systems)，所以建置比較費工夫，使用者比較缺乏對於運算資源或是環境調整與分配的彈性 。

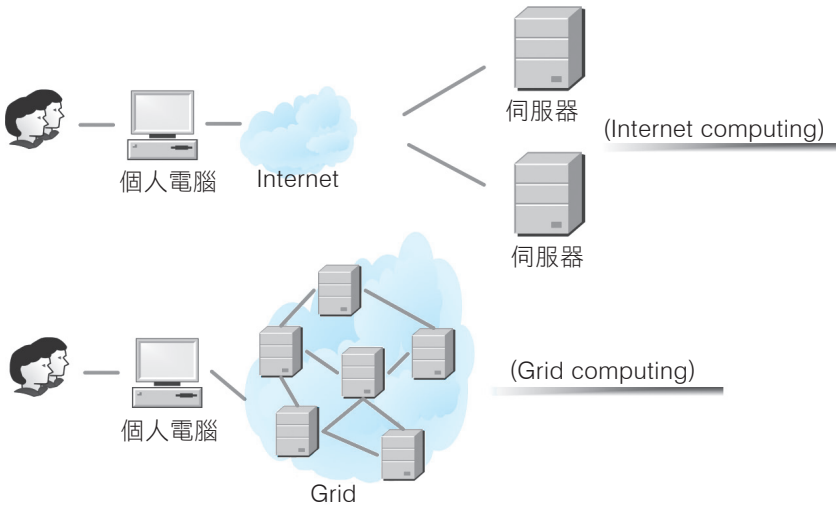

圖15-25 運算架構的變遷 (II)

4. 雲端運算 (cloud computing)： 圖 15-26 顯示雲端運算的架構，看起來好像跟網格運算沒有差多少，但是兩者之間有一些明顯的差異，雲端運算可以更有彈性地分享與分配資源，迎合更多元化的需求 。 雲端運算以使用者的需求為中心，讓用戶能彈性調整所需要的資源與運算的環境 。

碁峰 www.gotop.com.tw

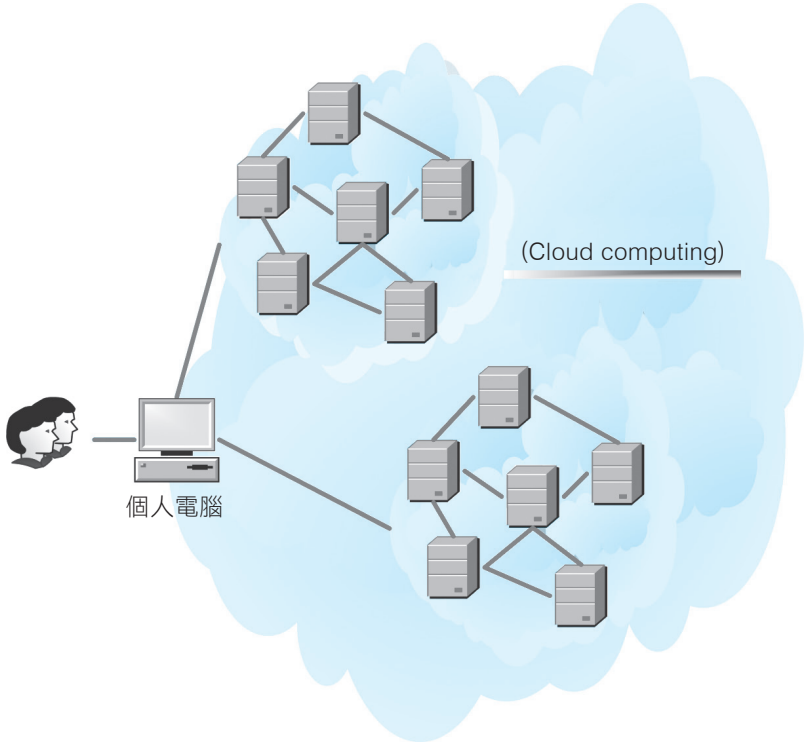

圖15-26 運算架構的變遷 (III)

雲端運算可以看成是由一群服務所組成的，如圖 15-27 所顯示的雲端運算架構，最上層的雲端應用由軟體即服務 (SaaS) 的概念來支援，讓使用者從遠端透過網路來執行資訊應用。平台即服務 (PaaS) 包括了作業系統跟相關的服務，換句話說，使用者還可以指定電腦的作業系統。架構即服務 (IaaS) 讓使用者指定電腦硬體、處理的效能與網路的頻寬。最下層的 dSaaS(data-Storage-as-a-Service) 代表硬體的儲存空間，提供穩定安全的資料保存空間。

對於雲端業者來說，要管理那麼多的硬體以及資訊資源，所需要的軟體就是一筆不小的投資，假如能夠使用開放原始碼的軟體，可以省下不少花費，尤其是以整個資訊環境的規模來說，開放原始碼與雲端技術的結合將能促成更可觀的經濟效益。

碁峰 www.gotop.com.tw

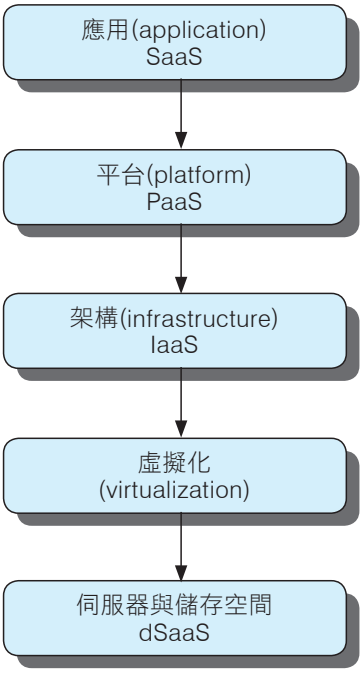

圖15-27 雲端運算的層次化架構

15.9.2 雲端運算的特性

雲端運算所指的是一種平行與分散的運算系統，靠的是網路連接的電腦，以一致的方式來提供運算的資源，服務提供業者與用戶之間有服務層級的協議 (SLA，Service-Level Agreement)，雲端運算具備下列的特性，是其他的運算架構比較欠缺或是沒有那麼完備的部分 ：

1. 彈性規模 (scalability) 與隨需求調整的服務 (on-demand service)： 雲端運算業者能夠隨著需求的差異調整所提供的服務，所以彈性很大，使用者可以依自己的需求來取得資源或是服務 。
2. 以使用者為中心的介面 (user-centric interface)： 不受地點的限制，使用者可以透過瀏覽器或是網際網路的服務 (Web service) 來取得雲端服務或是資源 。
3. 確保服務品質 (guaranteed quality of service)： 雲端運算可以確保運算效能、儲存空間或是網路頻寬等服務的品質 。
4. 自治的系統 (autonomous system)： 雲端運算的系統是自治的 (autonomous)，使用者不需要負責管理的工作 。
5. 合理的價格 ： 雲端服務不需要付出建置的初始費用，使用者可以依照自己的需求付出取得以及使用的服務費用。

碁峰 www.gotop.com.tw

15.9.3 雲端運算的技術架構

有人把雲端運算的技術架構分成兩大類：分散式的架構與虛擬化的架構。表 15-2 整理出這兩種架構特性的比較，事實上這些架構的形成跟業者在發展雲端運算技術時採用的策略有關，以 Google 為代表的業者一開始就傾向於尋求整合運用大量運算資源的方式，概念跟傳統的分散式運算類似。

表15-2 雲端運算技術架構的比較

比較的特性	分散式的架構	虛擬化的架構
運算的架構	採用新的程式設計架構	不改變傳統的運算架構
應用的範圍	網際網路上大量資料的處理與應用	整合資訊資源
資源的整合方式	聚集運算資源做彈性的分配	分享效能強大的運算資源並且做彈性的分配
工作的分派方式	透過運算工作的指定	透過虛擬機器或是虛擬化的概念

傳統的分散式運算或是平行處理利用多台的電腦來參與運算，將一個工作分割成多個小工作，交由合作執行的電腦進行平行的處理，縮短得到結果的時間。網格運算也是運用這樣的概念，Google 公司在技術上做了一些簡化，讓同樣的概念在網際網路中一樣可以落實，例如假設運算節點是不可靠的，這樣一來就要透過軟體來解決硬體不可靠的問題。指令與資料分流是另外一個關鍵，克服特定節點形成瓶頸的問題。

虛擬化的架構建立在資訊資源分享的概念上，一開始是在效能極高的硬體資源上透過虛擬機器的概念來進行分享，例如一台電腦上建立多個虛擬機器，安裝多種作業系統。後來進一步地擴充虛擬化的概念，凡是硬體的資源，不管數量多寡、種類或是特性，都可以經由虛擬化的技術來彈性地分享。事實上，對於雲端業者來說，這兩種技術架構可以並存、同時採用的，不過在了解雲端技術的發展時，這樣的分類可以幫助我們建立基礎的概念。

碁峰 www.gotop.com.tw

重點整理

分散式系統 (distributed system) 在資訊專業領域中是一門獨立完整的課程，事實上，資訊相關研究所開的作業系統課程有很多時間是在探討這個領域的主題。分散性 (distribution) 為作業系統帶來了很多複雜的問題，網路中介軟體的領域還在一直變化中，本章介紹的 RMI 與 CORBA 雖然已經有實際的軟體環境可以使用，但是在技術的演進上仍然有改善的空間，例如 ICE(The Internet Communications Engine) 就是比較近期的發展，大家可以尋找相關的文獻，順便比較一下 ICE 與 CORBA 的相似與相異之處。分散式物件的發展領域中有很多不同的技術存在，未來會如何變化仍然是有待觀察的。

自我評量

1. 試說明分散式系統中有那些理論上的發展是由作業系統延伸出來的？
2. 試比較 socket、 RPC 與 CORBA 在功能上的相似與相異之處，同時說明作業系統與這 3 者的關係。
3. 為什麼分散式系統中需要處理事件的排序問題？
4. 分散式系統中的死結處理跟一般單一電腦系統的死結問題有什麼不一樣的地方？
5. 為什麼在分散式系統中必須要求任何交易異動具有 ACID 的特性？
6. 談到分散式系統中時鐘同步的問題，試想一下一般人使用的股市看盤與下單系統中都會有一個時間，這個時間的資料從哪裡來的？來自我們的電腦的時鐘還是來自伺服器？為什麼？

碁峰 www.gotop.com.tw