分散式系統容錯之時間同步化策略及其在 Web 環境之應用

葉耀明、藍文正
國立臺灣師範大學資訊教育研究所
ymyeh@ice.ntnu.edu.tw 、 wclan@ice.ntnu.edu.tw

摘要

「時間同步化」（time synchronization）一向是分散式系統容錯中相當重要的課題。分散式系統的時間同步化上，有其技術上的困難與限制，如每個節點的計時器頻率不一致，外部參考時間的正確與否，系統及網路傳輸時間延遲的影響等等。

由於 Web 環境將成為未來分散式系統主要的應用發展平臺；但是 Web 環境在做時間同步化工作時，有其先天上的限制。本文探討將時間同步化策略應用於 Web 環境時所遭遇的困難與限制，並提出我們的解決方法。我們利用一種容錯效能佳之時間同步化的策略－滑動視窗演算法，應用於 Web 環境的發展出一套具容錯能力的時間同步化系統雛型－Web Ticker 系統。

關鍵字

分散式系統(Distributed System)，容錯計算(Fault-Tolerant Computing)，時間同步化 (Time Synchronization)，全球資訊網(World-Wide Web)

1 簡介

自從一九七0年代分散式系統發展以來，有相當多的學者發展出各種容錯之時間同步化的策略與演算法。根據 Parameswaran Ramanathan，Kang G. Shin 與 Ricky W. Butler 的分類【2】，基本上可以分為以下三大類型：（一）主僕式（master-slave）及時間伺服器式（clock server）時間同步化策略。（二）拜占庭協議式（Byzantine agreement）時間同步化策略。（三）收斂函數式（convergence function）時間同步化策略。

主僕式與時間伺服器式時間同步化策略是最廣泛被應用於分散式系統中達成時間同步化的方法。它最主要的策略，就是在系統中建置一個或若干個，具高可信度與有效度的時間伺服器。而系統中其它的節點則透過通訊網路與伺服器直接地擷取正確的時間，來修正各自的時間，進而達成整個系統中各節點之間時間上的同步。主僕式時間同步化策略最主要的應用，就是 NTP（Network Time Protocol)【3】及其相關的 SNTP(Simple Network Time Protocol）。主僕式與時間伺服器式時間同步化策略主要好處是原理簡單，設計容易，但缺點是容錯能力低，集中式管理易受瓶頸效應（bottle-neck effect）影響等等。

拜占庭協議式時間同步化策略主要是以分散式系統容錯中一項很重要的技術「拜占庭協議」【9】為基礎。它的優點是容錯能力強，而且適用於任何型態的系統錯誤，但是它需要相當大量訊息交換（message exchange）來達成，這會加重整個網路負載；而且它適用於特定訊息的交換，對於時間同步化中需交換非特定之參考時間訊息，有其應用上的限制。

收斂函數式時間同步化策略最主要的精神，就是蒐集系統中各個節點的參考時間，透過一個有效的收斂函數從所蒐集的參考時間裡決定同步化時間，再根據其結果來調整各自的時間。收斂函數式時間同步化策略的特性，就是每個節點地位平等的，不同於主僕式與時間伺服器式時間同步化策略。各節點僅需要交換彼此的參考訊息，訊息交換量小，但其容錯能力不如拜占庭協議式時間同步化策略來得好。

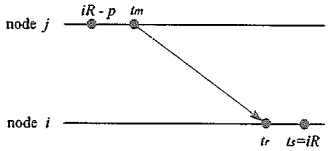
2 滑動視窗演算法

滑動視窗演算法（Sliding Window Algorithm，簡稱 SWA）是由 Manfred J. Pfluegl 和 Douglas M. Blough 兩位學者在 1995 年 5 月發表於 IEEE Journal of Parallel and Distributed Computing 期刊。它是歸類於收斂函數式時間同步化策略的一種，經過驗證，它比其它同屬於收斂函數式的時間同步化演算法，不論在精確度與容錯能力上，都有較佳的表現，這也是本論文經過比較後採用其演算法來應用於 Web 環境的原因。【4】

滑動視窗演算法與一般收斂函數式時間同步化策略相同，它是一個週期性時間同步化的機制，R 表示它的週期。在下次時間同步化週期來臨前，即 $iR-p$（i 表示進行第 i 次時間同步化，p 表示確保參考時間訊息可送達的時間），先從其它節點獲取參考時間值 t_m，再經估算其於時間同步化週期到達時的時間估計值

$$\tilde{X}_j = t_m + t_s - t_r + \varepsilon_{mean}$$

j 表示從節點 j 獲取的參考時間值，t_s 表示下次時間同步化時間（即 $t_s = iR$），t_r 表示收到該參考時間值的時間，ε_{mean} 表示平均誤差。其示意如圖一所示。



圖一　時間估計值示意圖

在計算完各個參考時間估計值後，接著就要採用一個收斂函數計算其合理的同步時間，也就是使用滑動視窗演算法來獲取我們需要的同步化時間結果，該演算法步驟如下：

（一）執行滑動視窗程序，獲得 n 個視窗樣本。（n 表參考時間估計值個數）

（二）選取包含時間估計值個數最多的視窗樣本做為標準視窗樣本。

（三）若包含最多時間估計值個數的視窗樣本非唯一，則選取當中標準差最小的一組視窗樣本為標準視窗樣本。

（四）若包含最多時間估計值個數且標準差最小的視窗樣本非唯一，則選取第一個視窗樣本為標準視窗樣本。

（五）選定標準視窗樣本後，則 SWA_{median} 表示該標準視窗樣本包含時間。

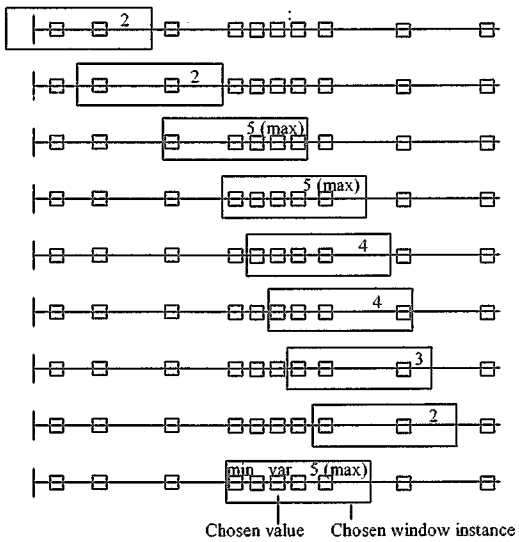
滑動視窗演算法之概念如圖二所示，其橫軸代表時間估計值之時間軸。滑動視窗演算法中的視窗長度 $\gamma = \Delta_{max} + \varepsilon_{max}$，其所獲的同步化時間結果之精確度為

$$\Pi = \varepsilon_{max} + \frac{k(\Delta_{max} + \varepsilon_{max})}{n-k} + \frac{k(2\Delta_{max} + \varepsilon_{max})}{n-k+1}$$

而其必要條件為

$$\Delta_{max} \geq \frac{(n^2+n-k^2)\varepsilon_{max} + 2\Re\bar{\rho}_{max}(n-k)(n-k+1)}{n^2-5kn+n+4k^2-2k}$$

如此可以保證容錯能力可達 $n/4$，即 $k < n/4$，其中 n 為時間估計值個數，k 為發生錯誤的時間估計值個數，R 為時間同步化週期，Δ_{max} 為時間估計值間隔中最大者，ε_{max} 為訊息延遲時間最大者，$\bar{\rho}_{max}$ 為參考時間偏移值之標準差。以上結果皆在 Manfred J. Pfluegl 和 Douglas M. Blough 論文中有所驗證。此外他們並將 SWA 與其它收斂函數式之時間同步化演算法，如快速收斂演算法（FCA）、中點容錯演算法（FTMA）及平均容錯演算法（FTAA）〖1，4，5，6〗，針對不同錯誤型態進行模擬比較。發現 SWA 的容錯能力遠優於其它演算法，尤其在短期錯誤的容錯能力更可以高達 50%。另外在精確度上也比其它演算法平均提升高達 38%左右。因此證明滑動視窗演算法的效能是相當不錯的，這也正是我們採用 SWA 做為分散式系統容錯之時間同步化策略最主要的因素。〖3〗



圖二　滑動視窗演算法概念圖

3　Web Ticker 系統

3.1　Web 環境時間同步化的限制

Java Applet 是我們用來實現 Web 環境時間同步化的主要工具，然而 Java 語言需要在 Java 虛擬機器（Java Virtual Machine）上執行，為了確保 Java 語言「跨平台」這項重要的特性，Java 虛擬機器在系統安全性加上了種種驗證與保護的機制，所以一些傳統的程式語言能夠達成的低階動作，如檔案的寫入與修改，Java 虛擬機器都會有相當嚴格的限制，特別是與網頁結合的 Java Applet，它的安全限制更多。就時間同步化的過程來說，唯一無法通過 Java 虛擬機器安全標準的，就是更動系統時間（system time）。偏偏更動系統時間又是整個時間同步化過程中最重要的一環，無法更動系統時間，也就失去了時間同步化的意義。

針對這方面，我們找到了可行且符合目標的解決之道－CORBA。CORBA（Common Object Request Broker Architecture）是由 OMG（Object Management Group）電腦工業公會所研發的產品。它是一種中介體（middleware），主要是讓網路上任何節點可以透過共同物件匯流排（object bus）找尋其它遠端的服務物件，進而與其通訊合作或使用。使用任何程式語言寫成的應用程式可以視為分散式系統中的獨立物件，只要編譯其搭配定義 CORBA 物件之 IDL（Interface Definition Language）寫成的.idl 檔，即可透過 CORBA 的共同物件匯流排－ORB（Object Request Broker）進行溝通合作。我們可以將更改系統時間的部份以其它語言寫成另一個獨立執行的應用程式，透過 CORBA 的 ORB 使得 Java Applet 與更改時間的應用程式溝通，解決先前 Web 環境中有關時間同步化的限制。

Java 虛擬機器多重的安全防護措施，也使得每個 Java 程式執行時需經過重重的檢查與認證。其所需的執行時間也因而被拉長，造成程式執行效能低落。在我們進行分散式系統時間同步化時，程式執行效能不佳，會影響到我們時間同步化結果的精確性。因此時間延遲（delay）的估計與掌握，也是我們在考慮時間同步化精確度時相當重要的因素。

3.2　系統需求分析

我們根據滑動視窗演算法，發展了一套 Web 環境之時間同步化應用系統雛型，我們取名為 Web Ticker 系統。整個系統的功能與需求分析如下：

伺服器端需要隨時接受新的客戶端連線要求，顯示系統時間，顯示經同步化後修正的系統時間，修改伺服器端系統時間，管理所有客戶端與伺服器端的連線狀況，查詢任何連線上之客戶端與伺服器端的時間差（offset），查詢任何連線上之客戶端與伺服器端的網路傳輸時間延遲，顯示時間同步化結果的精確度。

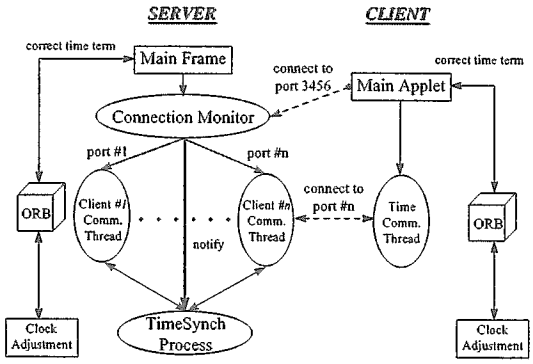
客戶端的需求為建立客戶端與伺服器端之連線，顯示系統時間，顯示經同步化後修正的系統時間，修改客戶端系統時間，顯示客戶端與伺服器端的時間差，顯示客戶端與伺服器端的網路傳輸時間延遲，顯示時間同步化結果的精確度。

3.3　系統架構

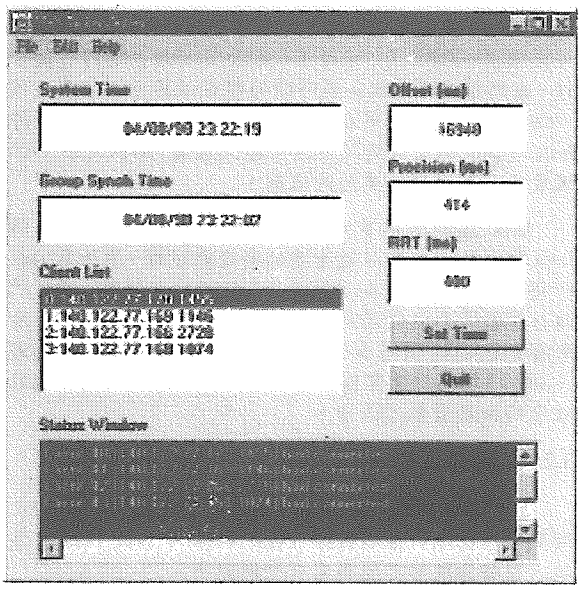
整個系統架構如圖三所示。主要可分為三個部份，除了主程式中基本初始設定與使用者介面處理外，需要一個專門負責網路通訊的處理程序，另外還有時間同步化的處理程序。在客戶端的 Java Applet 方面，主要分為兩個部份，主程式負責基本初始設定與使用者介面處理，以及負責與伺服器端通訊溝通的處理程序。此外，不論伺服器端或客戶端的程式，另外都需要更改系統時間的程式，與負責協調更動系統程式與主程式溝通的 CORBA 之 ORB。

Web Ticker 系統的伺服器端 Java Application 介面

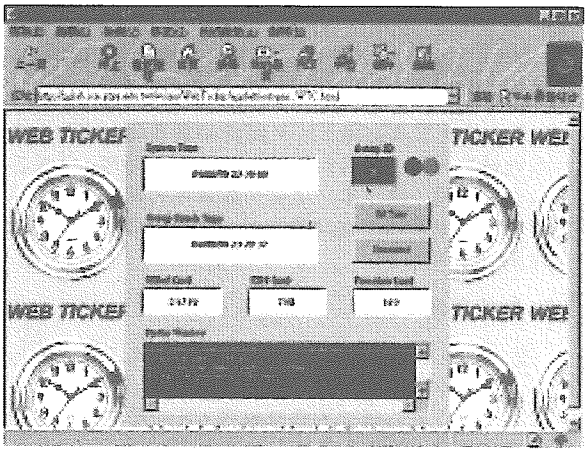
如圖四所示；客戶端的 Java Applet 介面如圖五所示。



圖三　系統之程式架構



圖四　Web Ticker 伺服器端的 Java Application 介面

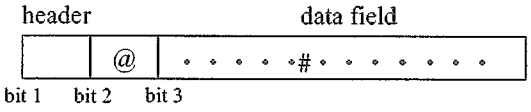


圖五　Web Ticker 客戶端的 Java Applet 介面

3.4 系統通訊協定

由於 Web 環境之時間同步化需要其它節點的參考時間，因此在我們的系統伺服器端與客戶端之間需要頻繁地交換訊息，進行溝通。為了規範客戶端和伺服器間的訊息傳輸方式，我們針對整個系統中可能的交換訊息機制，制定了一套簡單的通訊協定。

在此通訊協定中，關於交換訊息的資料格式（data format），我們是以字串（string）物件型態來處理，每個交換訊息都是一個字串物件。而在每個訊息字串中，我們將它區分為兩個欄位，中間以'@'字元區隔，前面第一個欄位是訊息標頭（header）的字元，代表各種不同訊息型態；第二個欄位是從第三個字元起，稱為訊息內容欄（data field），儲存傳達的訊息資料內容。訊息資料內容多為單一資料，如多於兩筆以上資料則以'#'字元加以區隔辨示，如圖六所示。



圖六　交換訊息之資料格式

4 系統評估

評估分散式系統容錯之時間同步化的結果，最重要的兩項基本元素就是可靠度（reliability）與精確度（precision）。可靠度主要就是指系統的容錯能力，以拜占庭協議式時間同步化的策略為例，它的容錯能力就高達三分之一。時間同步化精確度就是對於任何參與時間同步化的兩個非錯誤節點所形成之配對中，該配對中兩個節點時間差的最大值。以 SNTP 來說，它的精確度大致介於 10－100ms 之間。

分散式系統容錯之時間同步化的容錯能力，取決於其容錯策略的選擇。我們的 Web Ticker 系統係採用滑動視窗演算法，根據 Manfred J. Pfluegl 與 Douglas M. Blough 兩位學者在其論文中針對滑動視窗演算法進行的模擬分析模式，將主要的錯誤種類區分為四種：

（一）時間訊息的遺失

（二）過度的時間偏差

（三）隨機性時間值改變

（四）固定性時間值改變

我們的 Web Ticker 系統是建構在 TCP/IP 通訊協定的基礎，而 TCP/IP 本身是連線導向（connection-oriented），是可靠性的網路通訊協定，可以保證資料傳輸時的不遺失及封包的次序性。因此對 Web Ticker 系統中，訊息遺失的錯誤可以消除，增加另外三種錯誤類型的容錯能力。不過也因為採用 TCP/IP 的通訊協定，增加了網路傳輸的時間延遲，也降低了 Web Ticker 系統的精確度。

根據前面滑動視窗演算法所推導精確度的公式，我們可以推論影響整個時間同步化結果最主要的因素為 ε_{max} 與 Δ_{max}，其中 ε_{max} 為時間訊息延遲時間最大值，Δ_{max} 為時間估計值中間隔最大者。在整個系統啟動或新的客戶端加入時，可能會因為 Δ_{max} 值太大影響精確度，但經過兩次時間同步化的週期後，就可收斂至一定的數值。

另一個影響時間同步化精確度的因素，就是網路傳輸之時間延遲 ε_{max}。事實上，不只是滑動視窗演算法，對所有時間同步化的策略來說，時間延遲是個非常重要的考慮因素。就我們的 Web Ticker 系統來說，假設以節

點總數 n=100 個，Δ_{max} 為 150ms，容錯能力為四分之一（$k = n/4$），若要達到至少 1 秒鐘（=1000ms）的精確度，ε_{max} 值至少要小於 512ms。就一般乙太網路（Ethernet）的區域網路來說，在網路負載正常情況下，經由我們系統實驗得到ε_{max}值約為 200±50ms，精確度約為 418±20ms。雖然說精確度小於 1 秒鐘尚可以接受，但相較於其它軟體之時間同步化 10-100ms 的精確度來說，仍然相去甚遠。其中最主要的原因，還是因為網路傳輸的時間延遲太大，影響了整個時間同步化結果的精確度。針對造成網路傳輸之時間延遲過大的原因，我們歸納出以下兩點：（一）Web 環境與 Java 語言特性的限制，（二）TCP/IP 網路通訊協定的影響。其中第一項的是受限於 Java 平臺先天的特性，很難加以改進。但對於 TCP/TP 的影響，是可以有限度地加以改善。

由於我們的 Web Ticker 系統是採用 TCP/IP 通訊協定來做為網路通訊的基礎，雖然可保證時間訊息不會遺失，但因為 TCP/IP 裡頭為保證訊息不遺失的重送（retransmission）機制，增加了網路傳輸的時間延遲，造成ε_{max}過大，影響時間同步化結果的精確度。而且根據滑動視窗演算法的必要條件，ε_{max}過大會導致時間同步化的條件不易符合，無法進行時間同步化。

根據 ISO 所訂定 ISO 網路參考模式七層架構，Web Ticker 系統係屬於應用層（application layer）間的網路通訊，而 TCP/IP 是屬於傳輸層（transport layer）的網路通訊協定。根據我們的實驗結果，以我們的訊息資料量最大量（約 22 Bytes）為例，一般乙太網路的區域網路真正的傳輸時間延遲均小於 10ms，卻與我們測得動輒上百毫秒的時間延遲，有蠻大的差距。造成網路傳輸時間延遲過大的最主要原因，乃在於 TCP/IP 通訊協定的通訊協定延遲（Protocol Delay）造成ε_{max}估計過大。如同在傳輸速度很慢的通訊網路上進行時間同步化一般，互相傳遞交換的時間訊息延遲自然變大，間接影響時間同步化的精確與否。

有關以上問題的解決之道在於消除通訊協定延遲的影響因素，我們可以將整個時間同步化的工作拉至傳輸層（transport layer）上進行，以縮小訊息傳輸的時間延遲，讓整個網路的傳輸延遲接近真實的網路時間延遲。其效果如同改善網路傳輸的速度一般，將整個時間同步化的過程拉至較快速的通訊網路上進行，即能改善時間同步化的精確度，

然而消除通訊協定延遲的影響並非那麼容易，因為 TCP/IP 屬可靠性傳輸通訊協定，它的傳輸時間是無法預測的（unpredictable）。同樣地，它的通訊協定延遲亦是無法預測：此一時刻的通訊協定延遲無法代表或推測下一時刻的通訊協定延遲。因此很難直接量測真正的通訊協定延遲。我們可以使用替代方式，在進行時間同步化大量交換蒐集時間訊息時，伺服器端與客戶端各自送個訊息至本地主機（Local Host），訊息至傳輸層會被傳回來，如此可以量測當時伺服器端與客戶端的通訊協定延遲總和 Qd。

通訊協定延遲的大小可以顯示當時該機器的網路傳輸狀態，影響其值的因素非常多，包括網路負載（network traffic load）與該機器處理程序的忙碌狀況等等。雖然我們對當時的通訊協定延遲進行量測，但無法代表在交換訊息時所有訊息的通訊協定延遲。既然無法準確量測每次訊息傳輸時的通訊協定延遲，我們可以求最小參考值，以期盡量接近當時的通訊協定延遲。

由於我們 Web Ticker 系統主要是針對區域網路（LAN）應用，根據先前進行程式量測的結果，區域網路的真正傳輸時間延遲小於 10ms，假設真正網路傳輸時間延遲最大值為 d，第 i 個客戶端的通訊協定延遲為 Qc_i，伺服器端的通訊協定延遲為 Qs，則我們可得等式：

$$\varepsilon_i = \left(Q_{c_i} + Q_s\right) + d$$

因為 d <<（Qs + Qc_i），我們亦可以取$\varepsilon_{min} \equiv$（Qs + Qc_i）來做為消除通訊協定延遲的最小參考值。我們可以從直接測得的通訊協定延遲 Qd 與網路傳輸時間延遲的最小值ε_{min} 兩者之中取較小者，當做通訊協定延遲最小參考值。因此我們取 min（Qd，ε_{min}）做為消除通訊協定延遲的參考值。

從另一方面來看，其實消除通訊協定延遲最重要的意義，對滑動視窗演算法而言，是在調整滑動視窗的視窗大小。在進行消除通訊協定延遲的調整前，產生較低的精確度的主因是因為選擇視窗樣本的視窗大小或太大，導致選擇的視窗樣本不夠準確所致。經調整後的視窗大小較接近真實的訊息延遲，所求得的時間同步化結果精確度較高。

5 結論

在我們探討分散式系統容錯之時間同步化策略中，我們發現「滑動視窗演算法」是該類時間同步化策略裡效能頗佳的演算法，而且容錯能力均較其它同類演算法為強，對拜占庭錯誤的容錯能力可達 $n/4$（n 表節點總數）。因此我們以滑動視窗演算法為基礎，以目前分散式系統主要應用平台之一－Web 環境為發展平台，利用 Java Applet 跨平臺的特性，開發出 Web 環境上時間同步化的應用程式－Web Ticker 系統，以瀏覽網頁的方式達成伺服器端與客戶端的時間同步化。

在發展我們的系統過程中，我們發現了 Web 環境裡進行時間同步化有其先天的限制：第一個限制是 Web 環境為保有跨平台的特性，建構了層層驗證與保護的關卡，特別是無法允許作業系統核心層級的管理與操作。對時間同步化而言最大的困難就是無法改變系統時間。雖然我們找到了 CORBA 中介體來解決這個問題，不過無法完全地脫離平臺的限制仍是美中不足的地方。第二個限制是 Java Applet 執行的效能不彰，再加上 TCP/IP 通訊協定的不可預測性，導致傳送時間訊息延遲過大，影響了整個時間同步化結果的精確度，我們探究了其中可能的原因，並以降低 TCP/IP 的通訊協定延遲因素來調整滑動視窗演算法的滑動視窗大小，改善了整個系統時間同步化的精確度，但比起一般獨立執行的應用程式，仍無法達到相同程度的精確度。

因此在發展 Web 環境上之時間同步化軟體時，不可避免地要去面對以上所提先天的兩大限制。未來我們要努力的方向，首先是改採不同的分散式系統容錯之時間同步化策略進行比較，更換傳輸層的通訊協定，加入適當的容錯機制，正確量測通訊協定延遲，擴展廣域網路之應用。其次我們要與時間伺服器資源進行整合，統整所有分散式系統之時間同步化機制。

國科會計畫編號 NSC 88-2213-E-003-004

6 参考文献

【1】 Daly, W. M., A. L.Hopkins, Jr., and J. F. McKenna. "A Fault-Tolerant Clocking Systems." Proc. 3rd Symp. Fault-tolerant Computing Systems, IEEE, Los Alamitos, CA, 1973.

【2】 Parameswaran Ramanathan, Kang G. Shin, and Ricky W. Butler. "Fault-Tolerant Clock Synchronization in Distributed Systems." IEEE Computers, Vol. 23, No. 10, pp. 33-42. , Oct. 1990

【3】 David L. Mills. "Internet Time Synchronization: the Network Time Protocol." IEEE Transactions on Communication, Vol. 39, No. 10, pp. 1482-1493, October 1991.

【4】 Daly, W. M., A. L. Hopkins, Jr., and J. F. Mckenna: "A Fault-Tolerant Clocking System," Proc. 3rd Symp. Fault-Tolerant Cumputing systems, IEEE, Los Alamitos, CA, 1973.

【5】 Kopetz, H., and Ochsenreiter W. "Clock Synchronization in Distributed Real-Time Systems", IEEE Trans Comput. C-36, 8,pp933-940, Aug. 1987.

【6】 Lundelius, J., and Lynch, N. " A New Fault-Tolerant Algorithm for Clock Synchronization", Inform. And Comput. 77, 1, pp1-36, Apr. 1988.

【7】 Thambiduai, P., Finn, A., Kieckhafer, R., and Walter, C. "Clock Synchronization in MAFT", Digest, 19th International Fault-Tolerant Computing Synposium, pp142-149, 1989.

【8】 F. Cristian, H. Aghili, and R. strong. "Atomic Broadcast: From Simple Message Diffusion to Byzantine Agreement." 14th International Symposium on Fault Tolerant Computing Systems, pp. 200-206, 1985.

【9】 Jalote, Pankaj, "Fault Tolerance in Distributed Systems", Prentice-Hall, 1994.

【10】 R. Cole and Clare Foxcroft. "A Experiment in Clock Synchronisation." The Computer Journal, Vol. 31, No. 6, pp. 496-502, 1988.

【11】 L. Lamport, and P. M. Melliar-Smith. "Synchronizing Clocks in the Presence of Faults." Journal of the ACM, 32, January 1985.

【12】 Alessandro Braccini, Alberto Del Bimbo and Enrico Vicario. "Interprocess Communication Dependency on Network Load." IEEE Transactions on Software Engineering, Vol. 17, No. 4, pp.357-368, April 1991.