**國立臺灣師範大學**

**資訊工程研究所碩士論文**

指導教授：　黃冠寰　　博士

（論文題目）

研究生：　　簡偉智　 撰

中華民國　　105年　　月

**摘 要**

（論文題目）

簡偉智

這篇論文中我們所研究的是如何開發一個有效率的即時稽核方法以及雲端儲存系統使用的違反證明方法 (Proof of Violation, 簡稱 POV)。POV 方法可以讓使用者或是服務提供者做出密碼學的證據，可以用來證明服務提供者違反特性 (Properties)，或讓服務提供者證明自己的清白。POV 方法是讓雲端的使用者和服務提供者擁有互相不可否認性的方法。即時的稽核會在每一次的檔案操作之後執行，以確保可以即時發現服務提供者的違反行為。在目前所知的解決方法中，他必須要在客戶端的裝置中保留檔案的雜湊值，客戶端儲存及同步這些檔案的雜湊值會照成極大的負擔，而且當一個客戶端一段時間未做檔案操作，下一次檔案操作前的同步動作會花上非常長的時間。我們提出一個投票的方法讓客戶端的設備不需要保留任何的檔案的雜湊值。利用多個互相無關聯的服務提供者，客戶端不僅能即時的稽核、支援 POV 方法，又能同時擁有多份的備份。[實驗結果顯示......]。雲端儲存的服務提供者可以利用我們所提出的方法，將互相不可否認性的保證加入他們的服務層級協議。

**ABSTRACT**

（Thesis Title）

by

Wei-Chih Chien

目 錄

[第一章　緒論 1](#_Toc451007181)

[第二章　雲端儲存即時稽核協定 6](#_Toc451007182)

[第一節 證明違約協定 6](#_Toc451007183)

[第二節 雲端儲存即時稽核協定 6](#_Toc451007184)

[第三節 改良的雲端儲存即時稽核協定 8](#_Toc451007185)

[第三章　問題討論 13](#_Toc451007186)

[第四章　實驗結果 15](#_Toc451007187)

[第五章　相關研究 27](#_Toc451007188)

[第六章　結論 31](#_Toc451007189)

[參考著作 32](#_Toc451007190)

**附表目錄**

**附圖目錄**

# 第一章　緒論

雲端儲存 (Cloud Storage) 是一種藉由網路來存取的線上儲存系統，資料皆儲存於雲端的服務提供者所管理的虛擬化存放集區 (Virtualized Storage Pools)。服務提供者操控巨大的資料中心，使用者若想將資料放在這些資料中心，就需要向服務提供者購買或租賃儲存的容量。一些著名的雲端儲存系統有：Google Drive 、 Dropbox 、 OneDrive 、 iCloud 、 SugerSync 以及 Box。

安全性 (Security) 和可靠性 (Reliability) 是雲端儲存的兩大主要考量。客戶端不太可能會信任地將他們的資料交給一個沒有保障的公司。最基礎的安全性的功能有：身分驗證 (Authentication) 、資訊的私密性 (Confidentiality)、完整性 (Integrity) 和不可否認性 (Nonrepudiation)，其中前三項由使用密碼學的雲端儲存系統就可以簡單的實作出來。因為密碼學演算法的保護，使用密碼學的雲端儲存系統擁有私密性；每一個儲存在雲端儲存系統的檔案都可以對應到一個由使用者的私鑰建立的數位簽章，這樣就能確認資料的完整性。然而，使用者仍然無法保證他的檔案不會遺失或是從雲端儲存系統收到的檔案為最新的版本。試想下列的情況：因為某種內部的錯誤或是惡意的安全性攻擊，使用者儲存於雲端儲存系統的一些資料遭到損毀或刪除，服務提供者很有可能會利用之前備份的舊版本的檔案以及對應的數位簽章回復檔案，便可以否認使用者最新版本的資料遺失。這樣的情況便稱為回捲攻擊 (Roll-back Attack)，可以明顯的看出否認性這個問題存在於使用者和服務提供者之間。

有些系統考量雲端儲存系統本身就是不可信任的，因此提供一些方法來偵測有沒有違反完整性、寫入的順序性 (Write Serializability) 以及讀取的新鮮性 (Read Freshness)，且不需要備份資料或是依賴信任的第三方。他們的方法讓客戶端可以偵測出是否有來自任何有害的伺服器操作或惡意的使用者，企圖做未經驗證的檔案存取。他們需要保存檔案的雜湊樹或是檔案的版本編號以偵測是否有發生違反的動作。一般來說，這些方法只能拿來偵測違反動作的發生，他們無法提供正式的證據給第三方來證明服務提供者曾發生違反的動作。

一個雲端儲存系統應有責任提供一些特性，如：完整性、寫入的順序性以及讀取的新鮮性。然而，現今沒有任何的雲端儲存系統提供者在他們的服務層級協議(SLAs)中保證那些特性。譬如 Amazon S3 和 Azure 的 SLAs 中保證他們每月可提供 99.9% 的可用性 (Availability)。如果可用性降至 99.9% 以下，消費者就會有資格獲得賠償。然而，消費者無法在可用性以外的特性違反時獲得賠償。譬如說，如果檔案遭到損毀或遺失，使用者通常沒有任何明確的證據可以證明。因為消費者無法對遺失資料的風險做出明智的決定，他們能做的就是希望這樣的服務可以減少。

如果我們要在一個 SLA 中對更多的特性提供保證，我們需要一個實用的方法：當使用者做出錯誤的指控時，服務提供者可以證明自己是無辜的；當服務提供者違反了一些指定的特性，使用者可以拿來證明服務提供者的錯誤。一旦雙方的不可否認性構築於這個方法上，使用者和服務提供者可以制定一個金融合約，讓服務的價格由不同等級的安全需求決定。服務提供者保證若使用者能證明資料的安全性喪失或者被竄改時，將提供雙方約定的賠償金。

POV 這個方法能夠讓使用者或服務提供者提供一個精確的證據，來證明違反特性的情況發生；或是讓服務提供者證明自己是無辜的。POV 這個方法由三個元組 (Tuples) 組成：第一，他需要定義一組特性，服務提供者提供服務給使用者時必須不能違反這組特性。第二，證據必須是基於密碼學的證據，使用簽章將使用者和他們發出的請求和服務提供者全部結合在一起。第三，稽核能夠藉由收集密碼學的證據來實現，證明特性是否違反。稽核可以偵測出服務提供者是否違反了特性，或是服務提供者可以證明自己是否受到使用者錯誤的指控，也就是說，使用具有 POV 服務的使用者無法栽贓給服務提供者。假若一個違反的事件發生，客戶端和服務提供者可以利用密碼學的證據決定誰要來負責。並且，雙方若有爭議發生，他們可以呈現證據給一個第三方 (如仲裁人或法官) 。POV 這個方法完全滿足一個具負責任的雲端系統的需要。

研究將 POV 方法使用在雲端系統的是從基於 epoch 的方法開始的。當一個 epoch 開始的時候，使用者和服務提供者之間交換的密碼學的證據就會累積起來。接下來，隔了一段時間之後，累積的密碼學的證據就拿來執行稽核。在這篇論文中[16(之後要改)]，他採用文件目錄的雜湊樹和累積的密碼學的證據來實行稽核的方法。通過稽核就可以證明在這個 epoch 期間內，沒有任何的檔案操作動作違反了服務的特性。在稽核結束後，累積的證據便可以捨棄，並開始一個新的 epoch。一個 epoch 可能是幾個小時或者是幾天，藉由設備的效能或是通訊的頻寬等等原因所決定。

既然基於 epoch 的方法是可行的已經被論證，目前的問題就是只能在一個 epoch 結束時才偵測出特性的違反。考量以下情況：若儲存於雲端儲存系統中的是一個公司要簽署的重要文件，使用到錯誤的版本可能會造成巨大的損失。基於 epoch 的 POV 方法只能在 epoch 結束的時候才能偵測出特性的違反，而不能在每次下載檔案的時候偵測。因為有這樣的情況，驅使我們開發一種即時的 POV 方法，在每次的檔案操作當下都可以執行稽核。

在這之前我們的一份研究提出了一個可以在每次檔案操作時執行稽核的即時的 POV 方法，他提出了一個密碼學負責的協定，利用四步驟的握手協議來收集密碼學的證據。為了滿足即時的稽核的需求，下載的檔案的雜湊值應該暫存於客戶端的設備中。暫存於客戶端設備中的檔案的雜湊值會藉由儲存於服務提供者那裡累積的證據來同步，客戶端就可以捨去交換檔案的雜湊值的麻煩。比起儲存所有檔案的雜湊值，我們設計了只要儲存部份的雜湊樹於客戶端的設備，如此便能節省記憶體的需求以及初始化的時間。然而，收集密碼學的證據的負擔、同步雜湊值的負擔、以及儲存雜湊樹於多個客戶端設備的負擔仍然無法避免。相較於沒有使用即時的 POV 方法的檔案操作，小檔案(1~100kB)和大檔案(5~10MB)平均分別需要70~100倍以及8~10倍的時間來完成一個檔案寫入的操作。這驅使我們開發一個更有效率的方法。

[在這篇論文中我們提出了一個……]

[這篇論文的組織如下：……]

# 第二章　雲端儲存即時稽核協定

## 第一節 證明違約協定

前一章提到為了讓服務提供者可以證明自己的清白或是讓使用者證明服務提供者的確有出錯，過去的研究提出了 POV 方法，由三個元組組成：需要定義一組特性、證據必須是基於密碼學的證據、以及具有稽核的機制。藉由POV方法，可以建立更可信任的雲端儲存空間。

## 第二節 雲端儲存即時稽核協定

一篇先前發表的論文提出了一個以部分雜湊樹達成有效率的雲端儲存系統即時稽核方法，在此提到這篇論文的原因，是因為本論文後面提出的方法能夠有效的解決目前碰到的問題，因此先在此簡單介紹之前這篇論文的架構與做法。此論文考慮一個用戶有多個裝置，可能會同時有許多的裝置向服務提供者存取的情況。為了達到多個裝置的即時稽核，以及解決使用者與服務提供者之間的不可否認性問題，此論文提出了一個機制，如圖所示。

C:\Users\Chienweichih\Desktop\偉賢.png

首先，使用 Merkle tree，使用者將整個資料夾裡的檔案外取並以 buttom up 方式由最底下的檔案算到最頂端後的根雜湊(root hash)的值，使用者存著整個 Merkle tree 以及最後存取操作所留下的 root hash，在每一次和服務提供者操作不同的動作後，都需要更新 Merkle tree 以及 root hash，以確保使用者以及服務提供者彼此目前的狀態是一致的。

儲存伺服器儲存使用者的資料夾及檔案；稽核伺服器與使用者的裝置交換證據以及Merkle tree，以達到與服務提供者同步的步驟；使用者必須儲存 Merkle tree，以及最後留下來的密碼學的證據，用來和稽核伺服器更新目前最新的狀態。最後同步伺服器儲存最後一個使用者存取後的密碼學的證據以及 root hash，以防止服務提供者更改檔案或是稽核時需要的資料。

以上的架構雖然能夠運作，但我們發現在一個特別的情況下，使用以上架構的系統會遭遇到非常嚴重的延遲，使用者會因為這個問題，在未開始操作檔案的上傳下載動作前要等待一段明顯的延遲時間。此問題發生在使用者的一個裝置上線前，有其他的裝置操作了非常多次的動作之後，通常是因為此裝置一段時間未始用。因為後來上線的裝置需要先將自己儲存的Merkle tree 更新，前面累積了數百次、數千次的動作，Merkle tree 需要重新計算大量的節點的 hash 值，因此在還沒正式做指定的工作前，使用者必須等待一段很長的時間才能開始，此為以上架構之最壞情況(worst-case)。

## 第三節 改良的雲端儲存即時稽核協定

為了處理前面所提到的 worse-case，我們提出了一種全新的、更快速的架構，能夠解決客戶端在真正檔案操作的動作前同步時間太長的問題，如圖所示。

**C:\Users\Chienweichih\Desktop\我的.png**

在這個新的架構中，一樣使用同步伺服器來儲存最後操作所留下的證據，即服務提供者的回覆。但是在這個新的架構中，客戶端的設備完全不需要儲存任何的Merkle tree 或證據資料，藉由利用多個各自獨立的服務提供者，收集他們回傳的結果以即時的確認資料的版本是否正確。各個服務提供者的伺服器儲存使用者的資料夾的Merkle tree，還有備份這次操作修改的檔案的前一個版本，為的是後來稽核演算法的需要，利用備份的前一個版本的檔案記算出前一個版本的 Merkle tree。在後面的章節會解釋稽核演算法的步驟。

我們的系統的安全性基於一個假設：「同時有 k 個伺服器上，同一個檔案出問題的機率是趨近於0。」在此說明，因為我們可以選擇全世界不同地區的服務提供者，不容易因為天然或人為的災害遺失所有伺服器上的資料；而不同的服務提供者使用不同的演算法來備份使用者的資料，有心人士無法使用同一個方法來攻擊各式各樣的系統，因此我們這個假設是十分合理的。有了這樣的假設，使用者可以相信多數一樣結果的服務提供者是對的。每次的檔案操作請求後，收集各個服務提供者的回覆並比較，若有不同的結果出現時，只要稽核與多數不同的服務提供者就可以了。

即時稽核溝通架構

當每次設備P要向伺服器提出請求，都會遵循以下步驟：

步驟一 : Lock 同步伺服器。

步驟二 : 當設備P要存取他帳號(用戶U)裡的檔案，他會發送一個請求訊息 (request message) Qi, Qi=(OPi,[OPi]pri(U))給各個伺服器。其中OPi為請求的格式，例如讀取檔案、寫入檔案或者稽核要求。

步驟三 : 當一個伺服器收到Qi後，藉由驗證 [OPi]pri(U)電子簽章，以及檢查OPi裡面的檔案的hash值，確認是否為正確的請求。

步驟四 : 如果Qi為寫入檔案f，伺服器備份檔案f的hash值，使用Qi的OPi中的檔案hash值更新Merkle tree。

步驟五 : 伺服器將檔案f的hash值和Merkle tree的roothash值加入執行結果Li。

步驟六 : 伺服器回傳一個回應訊息(acknoledgement message) ACKi給設備P，其中ACKi=(Li,Qi,[Li,Qi]pri(Server))。

步驟七 : 設備P收集所有伺服器回傳的ACKi，確認所有的ACKi結果皆相同，更新同步伺服器的ACKi。

步驟八 : Unlock 同步伺服器。

步驟九 : 如果Qi為讀取檔案，向最近的伺服器下載檔案，確認ACKi中的hash值是否相同。

步驟十 : 如果Qi為寫入檔案，將檔案上傳到每一個伺服器。

在步驟七中，使用者藉由檢查每一個伺服器回傳的結果是否相同，馬上可以知道系統是否發生錯誤，服務提供者是否給予不正確的檔案或證據，即時的啟動稽核的機制。因此在這樣多個裝置的操作環境中，是可以防止回捲攻擊的。

因為伺服器回傳的ACK中有包含 root hash，最後儲存在同步伺服器中，因此可以確定在使用者未進行下一個請求前，服務提供者所儲存的資料是被驗證過沒有問題的。接著使用者發出下一個請求，若在這個請求後伺服器回傳的結果是錯誤的，就可以確定是在這一次的請求發生問題，使用稽核的演算法也就只要檢查這一次的過程中哪裡出錯，就可以確定發生錯誤的是使用者或是服務提供者。使用者要啟動稽核，首先需要向伺服器索取前一個版本的 Merkle tree (MTi-1)，接下來稽核演算法檢查完以下兩點，就能證明這次操作的過程是否有出錯：

使用者檢查 MTi-1的 root hash，應該要和儲存於同步伺服器中，前一版本的證據 (ACKi-1) 裡面的 root hash一樣，如此一來才能確定服務提供者沒有給錯誤的 Merkle tree 給使用者。

使用者以這次請求中的 OPi 裡面的 hash 值更新 MTi-1 ，也就是找到對應的檔案位置，將 hash 值修改後，在由下往上重新計算到 root hash。於是更新後 MTi-1 的 root hash 應該要和服務提供者現在的 root hash 是一樣的，若不同則表示這一次的請求是服務提供者出了問題，因此才會和使用者重新驗算出的 root hash 值不相同。如此一來，使用者就有足夠的證據能夠提供法院或是信賴的第三方，證明服務提供者的錯誤，以獲得雙方約定的賠償。

# 第三章　問題討論

在此研究中，有一項問題值得讓我們提出來討論。本研究的架構是設定能夠提供給多個設備同時進行存取的操作，藉由同步伺服器的控制，讓設備之間動作的紀錄不會有錯誤的順序，也不會讓多個設備同時讀寫同一個檔案時使系統發生死結 (Deadlock)的狀況。但是使用者和服務提供者之間需要交換證據以及傳輸檔案，考量可能會有各種不同效能、不同距離範圍、不同網路傳輸速度等種種原因，傳輸檔案會耗費相當長的時間。若使用之前那篇以部分雜湊樹達成有效率的雲端儲存系統即時稽核方法中的做法，使用者必須將檔案上傳給服務提供者或從服務提供者處下載檔案之後，雙方才能確認這一次的操作完全正確，才能驗證對方提供的證據正確，最後才可以 Unlock 同步伺服器。若傳送的檔案極大或是前面提到的種種原因，照成一個請求的操作時間非常長久，又或者是有惡意的使用者了解此架構如此的缺陷，故意地將檔案傳輸的速度減慢，一直重複這樣的動作使得服務提供者的服務被霸佔，使其他真正想要使用服務的使用者無法使用，這樣的攻擊稱之為 Slow Running Attack。

為了處理可能會發生 Slow Running Attack 的問題，本論文提出了和之前的論文不同的方法。首先，為了讓雙方都能有一份互不可否認之證據，使用者先傳送有自己簽章的請求，請求內包含這次操作的動作與選擇的檔案的雜湊值。服務提供者收到請求和簽章後，先驗證簽章沒有問題，再紀錄本次使用者將要操作的檔案的雜湊值，以確保之後的動作不會發生錯誤，接下來回傳回應訊息給使用者，就可以 Unlock 為其他設備服務了。能夠不用立即傳資料的原因是，使用者檢查了服務提供者回傳的回應訊息的簽章和資訊後，雙方就都擁有對方對這次操作的簽章，只要任何時間發生問題，都可以拿出這個證據來澄清自己的清白，或對方的錯誤。因此經過交換證據後，使用者就可以慢慢的將資料傳到服務提供者的伺服器上，服務提供者可以同時處理後面其他設備的請求，不會因為要處理一個非常大的檔案而占用非常久的時間，每一個檔案的處理時間都是固定的，因此 Slow Running Attack 便不會發生了。

# 第四章　實驗結果

在這個章節中我們實作了一系列的實驗來證明本論文提出架構的可行性以及詳細的執行時間，包含建立 Merkle tree、不同距離、不同數量的伺服器的各種步驟所需要花費的時間。首先先介紹我們實驗所使用的三個不同的帳戶，如[表]所示分別是(1) 帳戶 A，屬於檔案數目小以及容量小的範例，(2) 帳戶 B為檔案數目眾多，但是每個檔案的容量都不大，(3) 帳戶 C為實際使用了20年的一個資料夾，因此具有參考價值。

|  |  |  |  |
| --- | --- | --- | --- |
|  | 總容量 | 檔案數量 | 資料夾數量 |
| 帳戶 A | 777 MB | 48 | 6 |
| 帳戶 B | 145 MB | 54198 | 188 |
| 帳戶 C | 5.95 GB | 45089 | 1459 |

接下來的實驗我們在兩個平台上面實作：(1) Linux Mint 17.3 系統，作為客戶端的設備，(2) CentOS 6.6 系統，作為服務提供者端的伺服器。其中(1) Linux Mint 17.3 系統規格 CPU 為 2.13GHz Intel Core i3-330M 雙核心，8GB 的記憶體，120GB 的固態硬碟空間。(2) CentOS 6.6 系統規格 CPU 為 3.30GHz Intel Xeon E5-2643 四核心，8GB 的記憶體，53GB 的硬碟空間以及 Java 版本 1.8.0\_40。

建立 Merkle tree 的時間 (單位 秒)：

|  |  |  |  |
| --- | --- | --- | --- |
| 使用環境 | A | B | C |
| Linux Mint 17.3 | 14.876 | 61.176 | 198.405 |
| CentOS 6.6 | 6.821 | 144.267 | 620.151 |

[表]為第一次建立帳號時，所有的檔案都還沒有算過雜湊值，所以必須要計算所有檔案的雜湊值並彙整出 root hash 的時間。

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 使用環境 |  | A | B | C |
| Linux Mint  17.3 | 序列化時間 | 0.040 | 0.756 | 0.670 |
| 反序列化時間 | 0.009 | 0.299 | 0.295 |
| CentOS  6.6 | 序列化時間 | 0.011 | 0.343 | 0.299 |
| 反序列化時間 | 0.015 | 1.016 | 0.860 |

若之前已經計算過所有檔案的雜湊值、彙整出 root hash 後，使用者就可以將 Merkle tree 的 java 物件序列化（Serialize）輸出成檔案，將此檔案儲存，當未來要再一次使用 Merkle tree 時就能直接從檔案反序列化（Deserialize）回 Java 物件，快速的得到 Merkle tree，時間如[表]所示，可以看到在像帳戶 C 這麼多個檔案的資料夾中，產生與回復 Merkle tree 的檔案的時間並不長。

Merkle tree 占用容量：

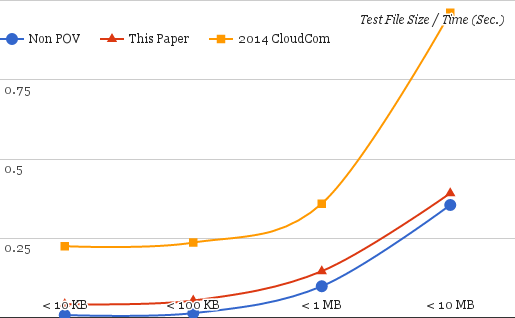
|  |  |  |  |
| --- | --- | --- | --- |
|  | A | B | C |
| Merkle tree 占用容量 | 5.4 KB | 5.08 MB | 4.37 MB |

接著[表]為產生完整 Merkle tree 的容量大小，Merkle tree 的容量大小由檔案數量所決定，可以看到就算是 54198 個檔案數量的帳戶 B 也只需要 5.08 MB 的Merkle tree 容量，這樣的容量並不會對客戶端的設備造成太大的負擔，而且在我們提出的方法中客戶端的設備不需要一直保留 Merkle tree ，只有在稽核計算時才會使用到，其他的情況下不會占用客戶端設備的容量。

客戶端設備和服務提供者在同一個網路區段

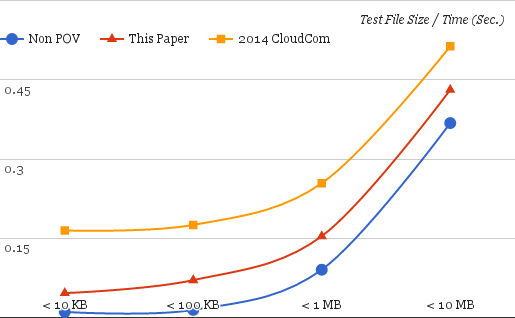
三種協定執行 Download 動作所花費時間 (單位 秒)：

|  |  |  |  |
| --- | --- | --- | --- |
| 檔案大小 | Non POV | 本論文提出方法 | 2014 CloudCom |
| 小於 10 KB | 0.007845 | 0.042295 | 0.224947 |
| 小於100 KB | 0.013691 | 0.053583 | 0.236347 |
| 小於1 MB | 0.098570 | 0.146021 | 0.359045 |
| 小於10 MB | 0.354916 | 0.392072 | 0.961740 |



三種協定執行 Upload 動作所花費時間 (單位 秒)：

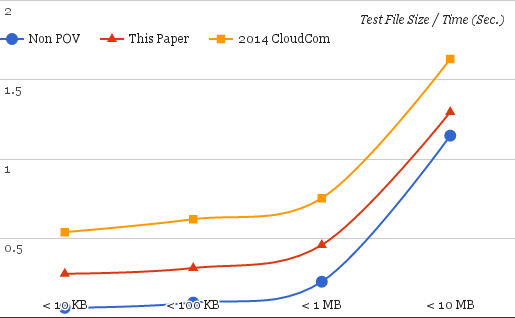
|  |  |  |  |
| --- | --- | --- | --- |
| 檔案大小 | Non POV | 本論文提出方法 | 2014 CloudCom |
| 小於 10 KB | 0.010608 | 0.046139 | 0.164744 |
| 小於100 KB | 0.014393 | 0.070739 | 0.175226 |
| 小於1 MB | 0.090440 | 0.153822 | 0.253963 |
| 小於10 MB | 0.367989 | 0.430937 | 0.513308 |



客戶端設備和服務提供者在不同的網路區段

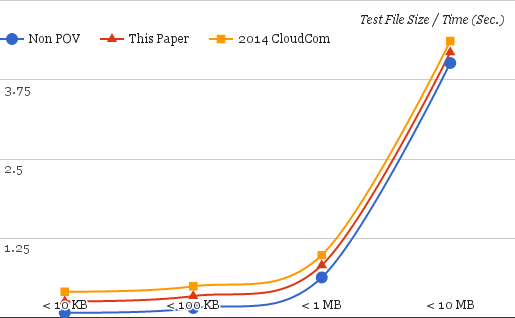
三種協定執行 Download 動作所花費時間 (單位 秒)：

|  |  |  |  |
| --- | --- | --- | --- |
| 檔案大小 | Non POV | 本論文提出方法 | 2014 CloudCom |
| 小於 10 KB | 0.061063 | 0.275808 | 0.538531 |
| 小於100 KB | 0.093941 | 0.312340 | 0.620296 |
| 小於1 MB | 0.225640 | 0.457329 | 0.752591 |
| 小於10 MB | 1.147272 | 1.296215 | 1.631534 |



三種協定執行 Upload 動作所花費時間 (單位 秒)：

|  |  |  |  |
| --- | --- | --- | --- |
| 檔案大小 | Non POV | 本論文提出方法 | 2014 CloudCom |
| 小於 10 KB | 0.077653 | 0.254801 | 0.407407 |
| 小於100 KB | 0.149493 | 0.338238 | 0.492000 |
| 小於1 MB | 0.631626 | 0.825261 | 0.983832 |
| 小於10 MB | 4.014217 | 4.182142 | 4.359997 |



接下來為讀取及寫入檔案的相關實驗數據，分成四個部分討論，(1)客戶端設備和服務提供者在同一個網路區段的上傳檔案時間、(2)客戶端設備和服務提供者在同一個網路區段的下載檔案時間、(3)客戶端設備和服務提供者在不同的網路區段檔案上傳時間、(4)客戶端設備和服務提供者在不同的網路區段下載檔案時間。

首先第一個部分，客戶端設備和服務提供者在同一個網路區段的上傳檔案動作，使用者向服務提供者們提出請求，服務提供者收到後回傳有簽章和結果的密碼學證據給使用者，使用者檢查回傳的證據中的簽章是否正確，收集這些證據之後互相比對，確定沒問題了再上傳檔案到所有的伺服器，完成動作，以上是本論文所提出的方法。其實驗數據如[表]所示，其中將檔案分為4組，分別是小於 10 KB 的檔案、小於 100 KB 但大於等於10 KB 的檔案、小於 1 MB 但大於等於100 KB 的檔案、小於 10 MB 但大於等於1 MB 的檔案，每組隨機挑出 100 個 Merkle tree 中所記錄的檔案做上傳到服務提供者的動作，最後再算出平均一個動作所花費的時間。然後將(1)沒有使用 POV，直接上傳檔案給服務提供者，然後只檢查上傳檔案的雜湊值是否和請求相同、(2)本論文提出的方法、(3)前面協定的章節提到的一篇先前提出的方法 (2014 CloudCom)，三個方法的數據比較。

從[表]中可以看到小檔案傳輸的速度，使用本論文的方法可以比2014 CloudCom的方法快4倍以上，最少也有2倍以上的速度，時間都可以在0.5秒以內完成。

第二個部分是客戶端設備和服務提供者在同一個網路區段的下載檔案動作，和上傳檔案一樣使用者和服務提供者互相交換證據，使用者收集這些證據之後互相比對，確定沒問題了再向一個伺服器下載檔案，完成動作。其實驗數據如[表]所示，一樣將檔案分為4組不同大小，每組隨機挑出 100 個 Merkle tree 中所記錄的檔案做下載的動作，最後再算出平均一個動作所花費的時間。然後一樣是三個方法，沒有使用 POV、本論文提出的方法以及 2014 CloudCom 三個方法的數據比較。由於下載動作服務提供者不需要更新 Merkle tree，直接提供檔案的雜湊值給使用者就可以了，因此所花費的時間少於上傳動作的時間。

從[表]中可以看到最大檔案傳輸的速度，本論文的方法比2014 CloudCom的方法快16倍以上，與沒有使用 POV 的時間非常接近，以此可見使用本論文提出的方法對使用者不會有太大的負擔。

第三個部分是客戶端設備和服務提供者在不同的網路區段檔案上傳時間，比較的方法、測試的檔案以及紀錄的方式都和同一個網路區段的做法一樣，不同之處就是客戶端設備和服務提供者的伺服器之間隔了9個路由器的距離，比較接近於真正使用者所使用的環境。這樣的實驗時間主要分布於檔案傳輸的部分，因此由實驗數據([表])可見和沒有使用 POV 的方法時間差距極小，可見本論文提出的方法在真正的雲端環境中是可行的。

從[表]中可以看到檔案傳輸的速度，不論測試檔案的大小如何，本論文的方法比2014 CloudCom的方法快2倍左右的速度，大大改良了雲端儲存空間的及時稽核方法的速度。

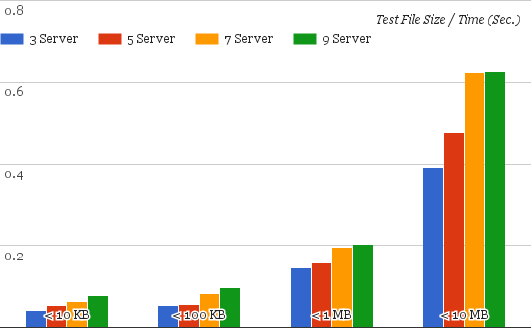
第四個部分是客戶端設備和服務提供者在不同的網路區段檔案下載時間，比較的方法、測試的檔案以及紀錄的方式等等也都和剛剛所說的一樣。由實驗數據([表])可見和沒有使用 POV 的方法時間一樣差距極小。

從[表]中可以看到檔案傳輸的速度，不論測試檔案的大小如何，本論文的方法比2014 CloudCom的方法快2到3倍左右的速度。

服務提供者使用不同數量的伺服器

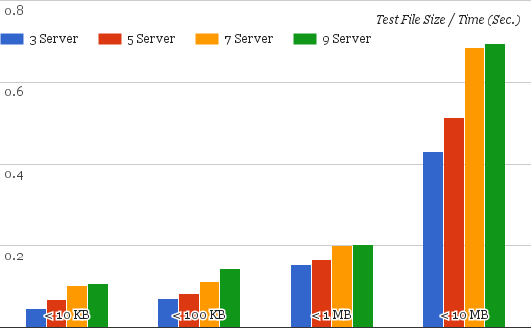
Download 動作所花費時間 (單位 秒)：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 檔案大小 | 3個伺服器 | 5個伺服器 | 7個伺服器 | 9個伺服器 |
| 小於 10 KB | 0.042295 | 0.054263 | 0.064370 | 0.078872 |
| 小於100 KB | 0.053583 | 0.055442 | 0.083961 | 0.097507 |
| 小於1 MB | 0.146021 | 0.159869 | 0.195817 | 0.202213 |
| 小於10 MB | 0.392072 | 0.476251 | 0.622665 | 0.625499 |



Upload 動作所花費時間 (單位 秒)：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 檔案大小 | 3個伺服器 | 5個伺服器 | 7個伺服器 | 9個伺服器 |
| 小於 10 KB | 0.046139 | 0.067923 | 0.101676 | 0.108696 |
| 小於100 KB | 0.070739 | 0.083563 | 0.112895 | 0.145049 |
| 小於1 MB | 0.153822 | 0.166289 | 0.200053 | 0.203870 |
| 小於10 MB | 0.430937 | 0.513879 | 0.684666 | 0.694259 |



接下來我們比較服務提供者使用不同數量的伺服器，所花費的時間各是如何。[表]的實驗數據分別是使用3個伺服器、5個伺服器、7個伺服器以及9個伺服器分別的時間，和剛剛的實驗一樣客戶端的設備將測試檔案分為小於 10 KB 的檔案、小於 100 KB 但大於等於10 KB 的檔案、小於 1 MB 但大於等於100 KB 的檔案、小於 10 MB 但大於等於1 MB 的檔案，每組隨機挑出 100 個 Merkle tree 中所記錄的檔案做上傳到服務提供者的動作。

[表]顯示上傳檔案所花費的時間隨著服務提供者使用伺服器的數量增加而上升，但是每增加兩個伺服器所增加的時間少於0.1秒，造成的影響並沒有明顯的提升，因此使用者和服務提供者可以依雙方的環境、需要備份的數量和希望的最低傳輸時間等等考量調整所使用的伺服器的數量，將他約定在服務層級協議之中。

[表]的實驗和前面所述相同，不同之處是將上傳動作改為下載的動作，一樣比較使用3個伺服器、5個伺服器、7個伺服器以及9個伺服器分別的時間。

2014 CloudCom 提出方法之 Worst-case (單位 秒):

|  |  |  |
| --- | --- | --- |
| Hash Chain 的長度 | Upload | Download |
| 100 | 0.595 | 0.827 |
| 1000 | 5.54 | 7.871 |

# 第五章　相關研究

目前常見的一些雲端儲存系統，像是 Google Drive， Dropbox， iColud， SugarSync， OneDrive 以及 Box，他們沒有提供擁有雙方不可否認性的SLAs。欲將伺服器上被錯誤更改的檔案回復，有一種做法是基於將資料備份到多個伺服器上。例如 Plutus 是一個加密的儲存系統，使用者不用依賴檔案伺服器是否能被信任，就可以分享機密的檔案。所有的資料都被加密的形式儲存，而且所有金鑰的分配是以分散的方式管理。客戶端的設備每一次讀寫都更改所有的備份檔，只要有足夠數量的備份檔，客戶端的設備就可以將損毀的檔案回復。然而，這些系統沒辦法提供任何形式的 POV。

挑戰-回應協定(challenge-response protocol)和可檢索的證明(proofs of retrievability, POR)被提出來實行檔案系統的遠端完整性證明。在這些方法中，驗證者會定期地傳送一個請求給伺服器，伺服器需要計算指定的檔案的驗證碼(checksum)，然後將結果回傳給驗證者。驗證者接者會拿自己儲存的驗證碼來比較回傳的結果是否正確。最近提出的一些基於 POR 的遠端完整性證明提供讓第三方的稽核者可以在遠端的伺服器上檢查資料的完整性。稽核者必須是一個被信任的組織(如政府)，才能在資料擁有者以及雲端伺服器之間提供沒有偏頗的稽核結果。由於稽核是由稽核者來執行，當使用者的數量愈多時就需要提供更多的伺服器。稽核者通常只會隨機的挑選各別的檔案或是檔案區塊，因此他們沒有辦法證明所有的檔案都有被稽核過。此外，第三方的稽核者沒辦法偵測是否有違反寫入的順序性和讀取的新鮮性。

有些系統考量雲端儲存系統是無法信任的，因此他們提供不需要將資料備份或使用信任的第三方的方法來偵測檔案的完整性、寫入的順序性以及讀取的新鮮性是否被違反。SiRiUS 是一個安全的檔案系統，他是實作在分層設計的不安全的網路之中。儲存在檔案伺服器上的檔案保留兩個部份：第一個部份包含檔案的後設資料 (metadata, md-file)，另一個部份是加密後的資料 (d-file)。SiRiUS 利用一個雜湊樹來保證 md-file 的新鮮性。使用者客戶端的設備會建立一個包含他所有 md-file 的雜湊樹。但是 SiRiUS 所保證的新鮮性只有在 md-file 上，因此是有可能被發動回捲攻擊，將最新版本的 d-file 替換成較舊的版本。SUNDR 是一個設計成可將資料安全地儲存在不被信任的伺服器上的網路檔案系統。SUNDR 讓客戶端設備可以偵測任何企圖以未經認証的方式修改檔案的惡意伺服器操作或是惡意使用者。SUNDR 使用的協定是實作了一種稱為 fork consistency 的特性，他能夠保證客戶端的設備只要可以看到每個檔案的修改，就可以偵測完整性和一致性發生的錯誤。客戶端的設備保留一個清單，其包含儲存伺服器上的檔案五個版本的紀錄。當使用者 U 執行一個檔案操作，他的客戶端的設備取得一個全域的 Lock，然後為每個使用者和群組下載最新版本的清單。這些版本清單可以用來偵測是否違反。SUNDR 利用一個稱為 forking semantics 的方法處理違反偵測的問題，這個方法也被其他的研究所使用。上述的解決方法保證完整性，並且在客戶端設備中增加一些額外的頻外 (out-of-band) 通訊來實現一致性的相關概念。Venus 系統解決的這個通訊的問題，透過延遲建立操作的一致性直到他們結束後的一段時間。總的而言，上述的這些方法只能支援偵測違反，他們沒有辦法提供一個方法來向第三方證明違反的發生。

Iris 是一個雲端檔案系統，他提供即時的完整性檢查。他保證消費者可以在任何時候驗證檢索的檔案是否違最新的版本，因此就可以預防回捲攻擊將檔案回復成之前的狀態。Iris 被設計為可以讓企業所使用。一個集中且被信任的入口網站置於企業信任的邊界，扮演著企業客戶端設備與雲端之間所有通訊的中介的角色。入口網站保留最近被企業用戶所處理的檔案以及使用者檔案的雜湊樹。此入口網站佈署了一個 semaphore 以確保共同處理暫留的雜湊樹，這可能會是這個系統的瓶頸處。雖然他可以提供動態的 POR，Iris 仍無法提供 POV。

Yumerefendi 和 Chase提出了一個框架，CATS，他能夠提供課責性(accountability) 於網路儲存系統。CATS 伺服器從有效的寫入操作讀取回復的結果來提供密碼學的證據。每一份 CATS 的請求和回復都包含了一個數位簽章，用以證明發送者的唯一的身份驗證以及證明訊息的完整性。為了支援寫入的順序性和讀取的新鮮性，當寫入者得到的寫入物件的版本戳章與目前完全一樣，如此的寫入請求伺服器才會允許。CloudProof 能夠提供基於 epoch 的 POV。使用者不只能夠偵測完整性的違反、寫入的順序性、讀取的新鮮性，他們還能夠向第三方提出以上違反事件發生的證明。使用簽章以及鏈雜湊值(chain hashing)，CloudProof 的架構能夠提供不可否認以及寫入的順序性的特性。新鮮性是由定期的稽核資料所保證。然而客戶端的設備如果對同一個帳戶做操作，就必須和設備們交換取得最新的使用者端的證據，否則客戶端的設備就必須保留全部使用者端的證據。

在我們先前的一篇研究，我們提出了一個新的方法，客戶端的設備不需要儲存任何檔案的雜湊值。使用者帳號的檔案，將其雜湊值儲存成 FBH 樹的格式，存放在服務提供者的伺服器。使用者帳號的原始資料夾和檔案結構利用索引函式對應成 FBH 樹。一個 FBH 樹是設計成儲存於一維的矩陣，因此我們能夠非常快速的讀取或寫入裡面的內容。並且設計了多個步驟的握手協定，使得服務提供者可以傳送一個 FBH 樹的片段給客戶端設備來提供即時的稽核以及POV。實驗結果顯示此方法速度優於之前的研究，效能提升了一至兩個數量級。這個方法在任何的情況下，都能比正常、沒有POV的檔案傳輸所需的兩倍時間內完成檔案的操作傳輸。雲端儲存的服務提供者能夠使用這篇論文中提出的方法，在他們的服務層級協議中提供一個互相不可否認的保證。但是本論文提出的方法能夠同時將檔案備份數份，更增加了檔案的安全性，這是之前這篇論文無法做到的優點。

# 第六章　結論

在這篇論文中，我們提出了一個新的方法，使雲端儲存系統能夠及時地發動稽核以及POV。我們考量一個使用者帳號的檔案可能由多個客戶端設備同時操作存取等等動作的情況。客戶端的設備不需要暫存任何的檔案雜湊值或是檔案狀態的資訊。

[架構]

[實驗結果]

# 參考著作