以輕量比對之資料重複刪除機制 提昇雲儲存環境之效能

王順生	嚴國慶*	王淑卿*	陳柏煒
朝陽科技大學	朝陽科技大學	朝陽科技大學	朝陽科技大學
工業工程與管理系	企業管理系	資訊管理系	資訊管理系
副教授	教授	教授	研究生
sswang@cyut.edu.tw	kqyan@cyut.edu.tw	scwang@cyut.edu.tw	s10114603@ cyut.edu.tw

*: 聯絡人

摘要

在網際網路快速發展的今日，雲儲存(Cloud Storage)的服務模式已漸漸被使用者所接受。現今有越來越多的使用者選擇將檔案存放至雲儲存環境中，因此，雲儲存環境各儲存節點需承受的負載量也越來越大。而透過資料重複刪除(Data Deduplication)技術，能夠在現有的硬體環境下釋放更多的儲存空間。在雲端運算環境中有大量的服務請求產生，若使用傳統資料重複刪除技術進行比對刪除，將會消耗過多的運算資源，進而影響到雲端服務的效能。因此，在本研究中，提出輕量比對之資料重複刪除機制(Lightweight Deduplication Mechanism；LDM)，以布隆過濾器為基底，修改傳統布隆過濾器的機制，能夠解決在進行資料重複刪除時，因比對重複資料時所帶來過多的資源消耗的問題。

關鍵詞：雲儲存、資料重複刪除、布隆過濾器、資料索引。

Abstract

Network bandwidth and hardware technology is developing rapidly, more and more users choose to use cloud storage, therefore the loading of storage node is heavier than before. Data deduplication techniques can greatly reduce the amount of data, but the traditional data deduplication techniques still have some defects such as consume a lot of computing resources. Therefore, the traditional techniques are not suitable in cloud computing environment. In this paper, Lightweight Deduplication Mechanism (LDM) is proposed. In LDM, the traditional Bloom Filter mechanism is modified that can solve the defects of traditional data deduplication.

Keywords: Cloud storage, Data deduplication Bloom filter, Data index.

1. 前言

由於電腦硬體設備的快速演進，網際網路及網路頻寬的蓬勃發展，分散式系統(Distributed System)的概念逐漸取代傳統集中式處理的運作模式。在 2007 年末，由 Google 提出雲端運算(Cloud Computing)以使用者為導向(User Oriented)的概念，掀起網路供應商在網路服務領域上另一種新格局的出現[12,14]。雲端運算是個概念而不是技術，是由分散式系統衍生出的概念，透過網際網路將大量的運算處理程序分解成無數個小的子程序，再交由多部伺服器組成的巨大系統，經由搜尋、運算分析，將處理的結果回傳給用戶端[2]。

雲端運算的特性包含了規模性、可靠性、高延展性、虛擬化及按需使用等特性[12]，使用者可以依照自己的需求向大廠租用運算資源或儲存資源，企業亦可選擇自行建立私有雲，針對較重要的商業資料進行備份[6]。雲儲存環境雖然能夠提供大量的儲存空間讓使用者使用，然而雲儲存的底層是由許多能力較低的節點所組成，藉以提供與伺服器相同能力的服務，因此當越來越多使用者將檔案存放於雲儲存環境時，各儲存節點所需負擔的成本將越來越大。而透過資料重複刪除(Data Deduplication)技術[6,8,9]，能夠將雲儲存各節點中冗餘、重複的資料刪除，進而釋放出更多的雲儲存空間進行使用。

使用傳統的資料重複刪除技術進行重複的資料刪除時，必須負擔一定程度的額外成本，例如需要耗費運算資源進行重複資料比對。由於在雲端運算環境中有大量的服務請求產生，若使用傳統的資料重複刪除技術進行比

對及刪除，將會消耗過多的運算資源，進而影響雲端服務的效能。

在王淑卿等人的研究中[1]，提出了「動態索引式布隆過濾器」改善了資料重複刪除技術資源消耗的缺點。但是該方法仍然存在著缺點，例如可能會產生稀疏矩陣，造成空間使用率過低的問題。因此，在本研究中提出了輕量比對之資料重複刪除機制(Lightweight Deduplication Mechanism；LDM)。LDM 除了可以解決「動態索引式布隆過濾器」的缺點外，並且能夠以更少的運算資源完成檔案比對之動作，使資料重複刪除技術在雲儲存環境中更為適用。

本文第 2 節為文獻探討，將介紹資料重複刪除與布隆過濾器的基本概念、及這兩個技術的相關研究；第 3 節將說明本研究所提出的方法；第 4 節為實例探討；最後一節為結論與未來研究。

2. 文獻探討

本節中將介紹資料重複刪除與布隆過濾器的基本概念、及這兩個技術的相關研究。

2.1 資料重複刪除(Data Deduplication)

資料重複刪除是一種檔案的壓縮方法，此壓縮方法與一般傳統檔案壓縮、增量備份等技術不同[10,11]，其最主要的目的就是刪除冗餘、重複之實體檔案。在資料重複刪除技術中，一般會將檔案利用特定演算法(如 MD5、SHA-1 等)轉換成數百個位元(bit)大小之唯一識別碼(Unique Identifier)。當檔案上傳時，將檔案之唯一識別碼與後端檔案系統進行比對，比對時若出現相同唯一識別碼，表示欲上傳之檔案已存在檔案系統中，因此將欲上傳之實體檔案進行刪除，直接於後端系統製作索引值，該索引值指向所比對之唯一識別碼的實體資料位置。若比對完成後沒有發現相同的唯一識別碼，表示檔案系統中無該檔之實體檔案，因此必須上傳完整檔案，並將此檔案之唯一識別碼存入檔案系統資料庫中，其概念圖如圖 1 所示。

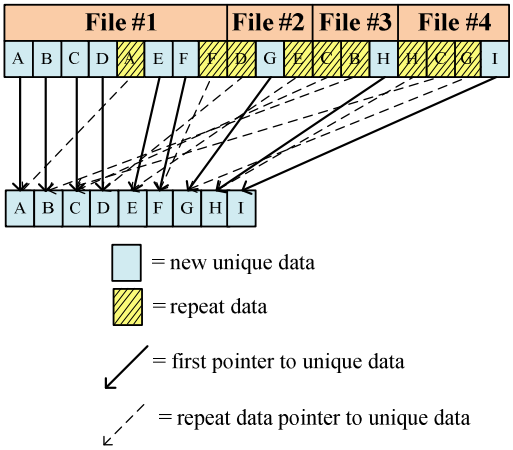

圖 1 資料重複刪除概念

資料重複刪除技術在定義重複資料之層級時主要分成兩個層級[7]，分別是檔案層級(File Level)及塊層級(Block Level)。兩個層級之優缺點比較如表 1 所示。資料重複刪除技術雖然能夠釋放出許多儲存空間，但該技術仍須負擔一些額外成本，包括：

(a) 比對重複資料消耗大量運算資源：當檔案系統中的檔案數量越多時，消耗之運算資源將會更可觀。

(b) 檔案之可靠性：透過資料重複刪除之運作後，所有實體檔案皆只剩下一份，檔案的可靠性將較為不足。

表 1 資料層級之優缺點比較

比較因子	檔案層級	塊層級
檔案數量	較少 (檔案不須切割)	較多 (檔案需切割)
比對刪除速度	較快	較慢
重複刪除率	較低	較高

2.2 布隆過濾器(Bloom Filter)

布隆過濾器是由 Burton Bloom 學者[4]在 1970 年時提出，它最大的特色就是能夠快速的辨識某個資料是否存在於資料集中。一般布隆過濾器會有一個 m 大小的陣列及 k 個雜湊函數(Hash Function)，陣列的初始皆為 0。

當有檔案進行儲存時，會透過 k 個雜湊函數進行運算，並將運算完之結果映射(Map)至陣列的相關位置上，被指到的陣列位置其值將轉換成 1，若重複映射至同一位置，則數值保持為 1 即可。進行資料判斷是否在該資料集中，亦是透過 k 個雜湊函數進行計算，若 k 個

雜湊函數映射到的位置皆為 1，即表示該資料存在於資料集中，若是有一個位置為 0，即表示該資料不存在於資料集中，其概念圖如圖 2 所示，其中在進行查詢時 File C 映射位置有一值為 0，透過快速過濾後可判定 File C 不存在於資料集中。

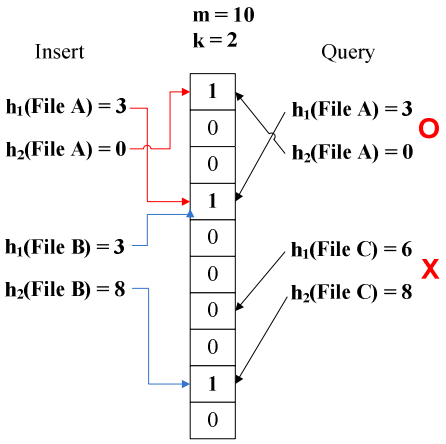

圖 2 布隆過濾器概念

布隆過濾器在極大的效率中有著“存在誤判” (Positive False)的可能性，亦即過濾器雖然判定該資料存在於資料集中，但事實上卻不存在。發生“存在誤判”情形時並不會導致系統錯誤或崩潰，但系統會浪費資源來進行檔案搜尋。而布隆過濾器可以保證不會發生“不存在誤判” (Negative False)的情形，即當布隆過濾器判定該資料不存在於資料集中，就表示該資料絕對不存在於資料集中。“存在誤判”的機率如公式(1)所示[3,4]，其中 m 為陣列大小、k 為雜湊函數數量、n 為存在資料集中的總檔案數量。

$$f' = \left(1 - \left(1 - \frac{1}{m}\right)^{kn}\right)^k \quad \ldots (1)$$

“存在誤判”的機率大小受到以上三個因子影響，表 2 為分析各因子與錯誤率之關係表。

表 2 各因子影響錯誤率

名稱	變數	減少	增加
陣列大小	m	增加錯誤率	減少錯誤率
雜湊函數數量	k	增加錯誤率	減少錯誤率
元素數	n	降低錯誤率	增加錯誤率

當 m 與 k 較大時，其錯誤率會比較小，但在實際應用上，建置布隆過濾器時會採用固定的陣列大小(m)，當採用固定大小的陣列後，k 值會有一門檻值，當 k 超過門檻值後，其錯誤率反而會上升。在設定 k 個雜湊函數時，若等於門檻值，將可得到最小的錯誤率，k 值門檻值的公式如(2)所示[4]。

$$k = \frac{m}{n}\ln 2 \approx \frac{m}{n} * 0.69314 \quad \ldots (2)$$

布隆過濾能夠透過極小的錯誤率換取極大的效率，因此在能夠接受些許“存在誤判”的應用下，布隆過濾器能夠完全發揮其優勢。然而，傳統的布隆過濾器有著許多不足的地方，包括：

(a) 陣列內容更新問題：當資料刪除時，陣列相關位置上的 1 無法變回 0，因為這個位置可能同時有好幾筆資料指向這個位置。

(b) 應用在檔案系統中之不足：傳統布隆過濾器只能做到過濾的效果，也就是說它只能夠初步過濾出此檔案是否存在於系統中，若判斷為是，將無法繼續判定該檔案儲存在哪裡，缺少了索引性的特性，在實用上還是有些不足。

因此近年來，針對布隆過濾器改良的議題，也是一個頗為熱門的研究主題[3,5,7,8,11,13][9]。

2.3 相關研究

資料重複刪除技術除了能夠有效降低儲存的容量外，也有一些研究在進行整體效能的提升。Thwel 等學者[9]，認為進行資料重複刪除機制時，需特別注意的是當進行比對時遇到完全區塊索引(Full-chunk Index)的情形發生。為了避免資料重複刪除進行比對時間過長，因此 Thwel 等學者採取 B+ tree 的資料結構建置系統，以較快速的索引提升資料重複刪除機制的整體效能。

在布隆過濾器的相關研究中，針對不同的應用也有許多學者提出改良的策略。Antichi 等學者[3]透過改良布隆過濾器的機制，能夠將其結合索引值，雖然過濾器判斷存在後能夠快速的找到索引位置，且時間複雜度為 O(log(n))。但作者假設必須在完美雜湊函數(Perfect Hash)下才能夠實現，在實際應用上，完美雜湊函數的設計式非常不容易的，且 Antichi 等學者的研究也沒有考慮到刪除檔案時，布隆過濾器該如何處理。

Papadopoulos 等學者[8]，利用 R-tree 的資料結構與布隆過濾器的資料結構進行範圍查詢(Range Query)與單點查詢(Point Query)的應用，在其實驗中可發現，布隆過濾器在實現單點查詢有非常優異的表現。

Wei 等學者針對布隆過濾器擴充之問題，提出動態布隆過濾器架構[11]。在 Wei 等學者提出的架構中，並未修改舊有陣列之映射位置，而是新增新的陣列來進行映射放置，藉此能夠有效降低錯誤率，以滿足其動態性。但 Wei 等學者所提出的方法需耗費較多的資源進行查詢作業，原因是因為必須查詢多個陣列才能判定檔案是否存在。

雖然目前有學者提出布隆過濾器與索引值結合的方法，但卻必須在完美雜湊函數下才能進行。然而在一般大型的檔案系統中，勢必經常需要進行檔案的刪除，陣列內容更新問題也是必須去解決的。因此在本研究中提出一個較佳的解決方案，以較佳的時間複雜度進行索引查詢，並且該機制亦能夠滿足陣列內容的更新。

3.研究方法

在本研究中，採用塊層級之資料重複刪除機制。由於塊層級之資料重複刪除機制於比對過程中需耗費大量運算資源，因此在本研究中提出輕量比對之資料重複刪除機制(Lightweight Deduplication Mechanism；LDM)來解決此問題。系統架構將於 3.1 節中說明，LDM 執行策略則在 3.2 節中說明。

3.1 系統架構

在本研究中所提出的架構中，數個儲存節點集結成一個群組(Cluster)，群組的組成為一台 Namenode 及數台 Datanode，其中 Namenode 為群組頭(Cluster Head)，負責進行機制處理與放置檔案至 Datanode 中，Namenode 本身也會提供儲存空間，而 Datanode 的功用為提供儲存空間的儲存節點，每一個群組的 Namenode 皆擁有兩個元件，其中 LDM 為機制執行之元件、Metadata Table 為儲存檔案之元數據(Metadata)，系統架構圖如圖 3 所示。

當使用者上傳檔案前，首先透過傳輸代理人系統(Transfer Agent System；TAS)進行前處理，將檔案以每 b MB 之大小切割成一檔案區塊，並利用 SHA-1 演算法將檔案區塊轉換成唯一識別碼，最後選擇任一儲存群組進行上傳。檔案上傳時會執行 LDM 元件進行重複刪除比對，確認檔案是否重複。若是重複，則在本地端進行刪除動作，並製作索引檔指向實體檔案之位置，Metadata Table 將會記錄所有索引檔之詳細資料。若是不重複，表示檔案系統還沒有此檔案之實體檔案，因此必須進行上傳動作。

每一個儲存群組都包含兩種類型之檔案，分別是實體檔案(Source File)及索引檔案(Index File)，在同一儲存群組裡，實體檔案不會重複，亦即只有一份實體檔案，所有使用的檔案都是索引檔案，索引檔案會指向實體檔案之位置，圖 4 為實體檔案與索引檔案之關係圖。

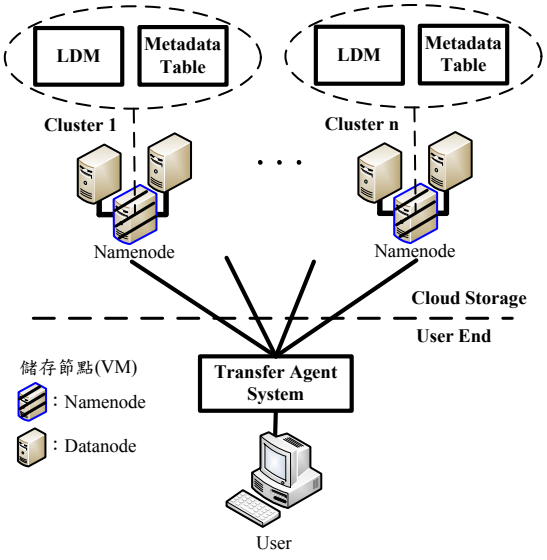

圖 3 系統架構圖

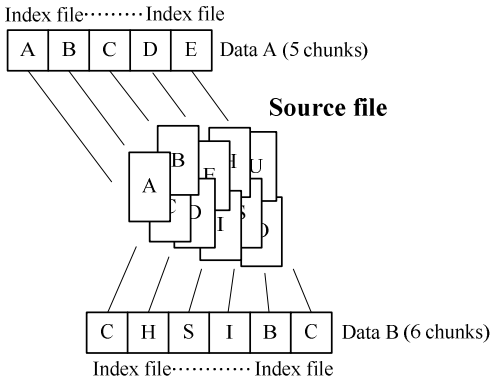

圖 4 實體檔案與索引檔案之關係

3.2 LDM 執行策略

在本研究的系統架構中，每一個儲存群組都擁有 LDM 機制，LDM 機制中會使用布隆過濾器，首先須進行布隆過濾器初始化，設定布隆過濾器之參數，初始化之流程如圖 5 所示。

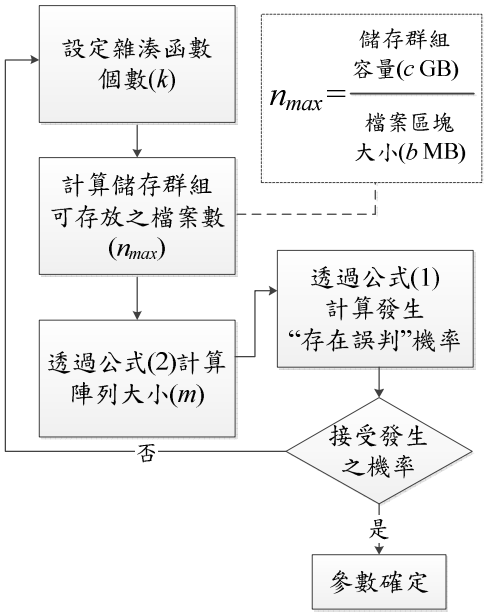

圖 5 布隆過濾器初始化

步驟與實例說明如下：

1. 設定雜湊函數個數(k)：假設服務供應商共設計 7 個雜湊函數。
2. 計算該儲存群組內可以存放的檔案數量(n_{max})：假設該儲存群組之容量為 40 GB，檔案區塊大小為 64 MB，因此該群組可存放 640 個檔案(40 GB/64 MB)。
3. 透過文獻探討中的公式(2)計算可得出布隆過濾器之陣列大小(m)為 6463。
4. 透過文獻探討中的公式(1)計算可得出"存在誤判"之機率為 0.0078，亦即平均查詢 1000 次會發生 8 次"存在誤判"之情形。
5. 服務提供商選擇是否接受該"存在誤判"機率，若接受則完成布隆過濾器參數設定；若不接受則重新回到第一步，設計較大的 k 值，即可得到更小的"存在誤判"機率。

由於在資料重複刪除系統中，實體檔案不會重複，因此 n_{max} 可視為該儲存群組之實體檔案最大數量。而為了解決傳統布隆過濾器之陣列內容無法更新問題，因此本研究採用整數陣列型態進行設計，以實數表示還有幾個檔案指向該位置，在進行查詢時，k 個映射指向之位置若不為 0，即判斷該檔案存在，如圖 5 所示，File A 判定為存在，File B 判定為不存在。

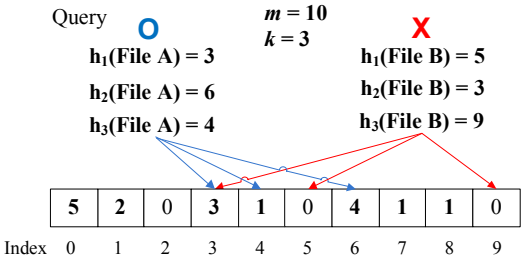

圖 6 整數型態之布隆過濾器

圖 6 之布隆過濾器陣列中，陣列索引編號 3 的內容值為整數 3，表示有 3 個檔案指向該位置，若使用者後續使用時將 File A 刪除，則其所映射位置之值將會減 1，如圖 7 所示。

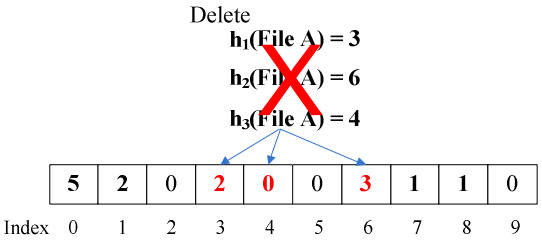

圖 7 File A 刪除之結果

在王淑卿等人的研究中[1]，提出的「動態索引式布隆過濾器」改善了資料重複刪除系統資源消耗之問題，該研究之方法架構如圖 8 所示。

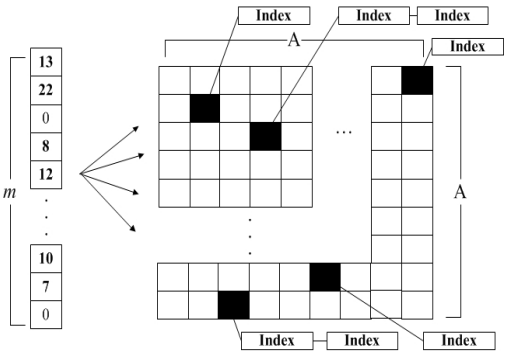

圖 8 「動態索引式布隆過濾器」概念

在「動態索引式布隆過濾器」方法中[1]，當檔案要存放至系統中時，首先透過 k 個雜湊函數運算並映射至布隆過濾器上。接著，透過公式分別計算二維陣列中第一維之位置及第二維之位置，並映射至方法所設計的二維陣列上。當檔案進行查詢比對時亦是透過相同方法取出，透過該方法能夠降低大量的比對運算資源。

「動態索引式布隆過濾器」仍然存在著一些缺點，如二維陣列之大小該如何決定，並且檔案之唯一識別碼透過雜湊函數運算後之結果視為不可預期之結果，可能會發生不同檔案映射至同一個二維陣列位置中，導致某些陣列位置存放許多值，而某些陣列位置卻是空值的稀疏矩陣問題，造成一定程度上的空間浪費。

因此本研究提出 LDM 機制，除了改善「動態索引式布隆過濾器」的問題，使空間的使用率較佳化，並且能夠以更少的運算資源達到比對之動作，LDM 機制概念圖如圖 9 所示。

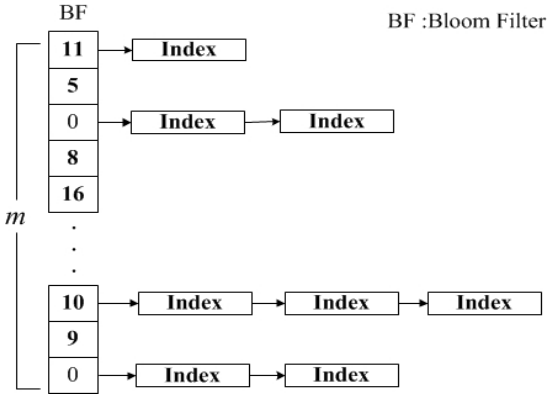

圖 9 LDM 機制概念

在 LDM 機制中，採用鏈結串列的方式進行存放檔案。當進行新增檔案時，首先透過 k 個雜湊函數運算並映射至布隆過濾器上，接著使用 k 個運算完的雜湊值進行運算，決定鏈結串列之起始位置。k 個雜湊值轉換成鏈結串列之起始位置如公式(3)所示，其中 $val(h_i)$為第 i 個雜湊函數所計算出的雜湊值、m 為布隆過濾器之長度、%符號表示餘數函式(mod)。

$$Link(position) = (\sum_{i=1}^{k} val(h_i)) \% m \ldots (3)$$

透過上述公式計算後，將得到布隆過濾器上的某個位置，接著該位置將產生鏈結串列指標，檔案之索引值將會儲存在該鏈結串列上。舉例來說，假設有 4 個雜湊函數 $val(h_1)=6069$、$val(h_2)=36$、$val(h_3)=5$、$val(h_4)=642$，$m=6463$，透過(3)計算(6069+36+5+642) % 6463 = 289，因此會在布隆過濾器之陣列索引值 289 產生鏈結串列，存放該檔案之索引值。

在 LDM 機制中，若不同檔案計算完畢後之位置相同(如 File X 計算完畢後亦為 289)，則會由最後一個鏈結串列產生新的鏈結，根據需求動態產生鏈結空間進行存取，可有效將空間使用率提升，整體概念圖如圖 9 所示。

在 LDM 機制中，檔案比對的動作和存入之動作相同。當有比對需求進入後，首先透過布隆過濾器判斷檔案是否存在，若布隆過濾器判斷不存在，則不需進行比對刪除動作，系統可直接上傳檔案；若布隆過濾器判斷存在，則透過公式(3)進行計算，並至相關位置尋找鏈結串列，透過本研究之方法能夠快速進行比對動作。若是搜尋完所有鏈結串列仍未發現相同檔案，表示發生“存在誤判”情形，而 LDM 透過快速的判斷，亦能將“存在誤判”所付出的額外成本降低。

4. 實例探討

在本研究所提出之方法中，若是某一陣列位置所指向的鏈結長度過長，整體的搜尋時間將會過久，因此在本節中將以實例進行探討並與先前所提出的「動態索引式布隆過濾器」(Dynamic Index Bloom Filter；DIBF)方法進行比較。透過研究方法中，布隆過濾器初始化流程，定義出模擬環境如下：

群組一：

- 雜湊函數個數(k)：7 個
- 儲存群組容量：40 GB
- 檔案區塊大小：64 MB
- 群組可存放之檔案數(n_{max})：640 個
- 布隆過濾器陣列大小(m)：6463

群組二：

- 雜湊函數個數(k)：7 個
- 儲存群組容量：100 GB
- 檔案區塊大小：64 MB
- 群組可存放之檔案數(n_{max})：1600 個
- 布隆過濾器陣列大小(m)：16158

本研究使用 Microsoft visual studio 2008 C#撰寫 LDM 與 DIBF 兩個機制，並以群組一及群組二兩個腳本分別進行模擬。在模擬測試

環境中，將針對各群組存放不同數量之檔案進行測試，並觀察其最大鏈結長度。由於每次存進檔案系統的檔案之型態無法事先預測，因此雜湊函數皆以隨機值輸入至系統進行位置計算。圖 10 為群組一測試 10 次取平均之結果，圖 11 為群組二測試 10 次取平均之結果。

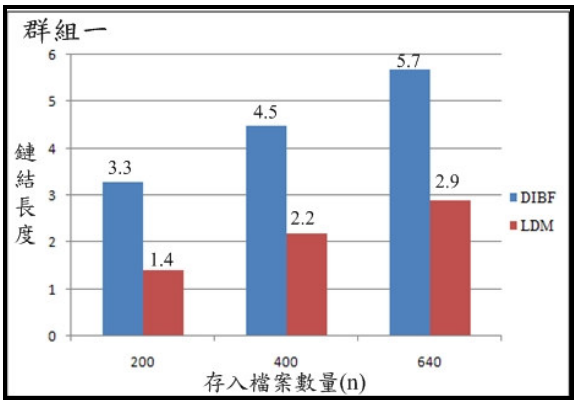

圖 10 群組一之模擬結果

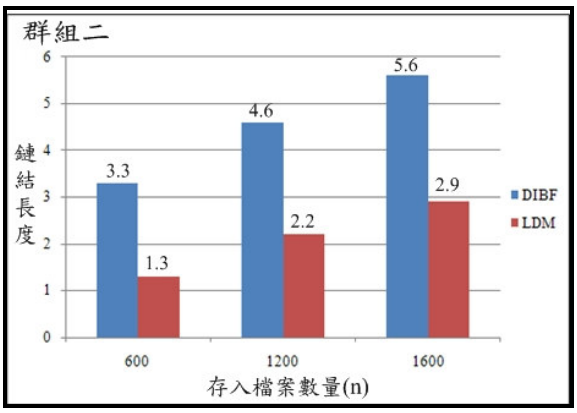

圖 11 群組二之模擬結果

由模擬實驗中可以發現，群組一與群組二之最大鏈結長度幾乎相同，原因是因為在 DIBF 與 LDM 兩個機制中，布隆過濾器之初始化參數皆是以等比方式進行設計的。亦即，雖然群組一與群組二的儲存群組容量不相同，但經過相同公式進行設計後，所得到的布隆過濾器以比例來說是相同的，因此最大鏈結長度會相同。

從圖 10 及圖 11 結果可以得知 LDM 機制之最大鏈結長度比 DIBF 機制還要短，亦即 LDM 機制能夠以較快的速度進行索引比對之動作。LDM 方法會比 DIBF 好的原因是因為，DIBF 機制中採用二維陣列進行存放，二維陣列必須設計剛好大於等於該群組可容忍之檔案數量，以群組二來說，可容忍 1600 個檔案，因此 DIBF 機制之二維陣列將設計為 40*40 之陣列大小，而在 LDM 機制則是將鏈結串列新增在現有的布隆過濾器後方，一般而言布隆過濾器之長度會大於 DIBF 機制所設計之二維陣列大小。以群組二來說，DIBF 機制要存放 1600 個檔案，會將 1600 個檔案放到長度 1600 的陣列(二維陣列 40*40)中；LDM 機制則是將 1600 個檔案放到長度 16158 的陣列(布隆過濾器大小 m)中。以機率來說，以 DIBF 機制檔案存放到重複的陣列位置會比 LDM 機制還要來得高，從模擬實驗結果中也能夠證實。並且 LDM 機制採取動態產生鏈結串列方式進行存放索引值，整體的空間使用率亦比 DIBF 機制來得更好。

5. 結論

資料重複刪除技術能夠降低雲儲存節點的負載量，在雲端環境中有大量的服務請求產生，若使用傳統資料重複刪除技術進行比對刪除，將會消耗過多的運算資源，進而影響到當前的服務效能。

在本研究中，我們提出了輕量比對之資料重複刪除機制 (Lightweight Deduplication Mechanism；LDM)，LDM 透過改良布隆過濾器之機制，能夠快速進行資料重複刪除之比對動作，改善傳統資料重複刪除技術消耗過多運算成本之缺點。LDM 之空間使用率比「動態索引式布隆過濾器」還要好，並且從模擬實驗結果來看，LDM 亦有較佳的查詢索引速度。

本研究所提出的方法適用於靜態穩定的儲存容量下，當容量增大時，則無法改變布隆過濾器陣列(m)的大小，原因是因為若是改變了 m 的大小，所有的檔案都必須重新計算存放位置，當 m 的大小不變，存放於布隆過濾器的檔案數量越來越多時，“存在誤判”之機率將變得越來越高。在未來工作上，將朝著以最小的成本消耗，來達成能夠適應在動態的儲存容量。

致謝

這篇論文是國科會計畫(NSC101-2221-E-324-032 與 101-2221-E-324-034)研究成果的一部份，在此我們感謝國科會經費支持這個計畫的研究。

參考文獻

[1] 王淑卿、王順生、嚴國慶、陳柏煒，「在雲儲存環境下 以動態索引式布隆過濾器增進資料重複刪除系統之比對效率」，***2012 年民生電子研討會***，2012。

[2] Aymerich, F.M., Fenu, G. and Surcis, S., "An Approach to A Cloud Computing Network," ***Proceedings of the First International Conference on the Applications of Digital Information and Web Technologies***, pp. 113-118, 2008.

[3] Antichi, G., Pietro, A.D., Ficara, D., Giordano, S., Russo, F. and Vitucci, F., "Achieving Perfect Hashing through an Improved Construction of Bloom Filters," ***Proceedings of the IEEE International Conference on Communications***, pp. 1-5, May 2010.

[4] Bloom, B.H., "Space/time Trade-offs in Hash Coding with Allowable Errors," ***Communication of the ACM***, vol. 13, no. 7, pp. 422–426, July 1970.

[5] Brunette, and G., Mogull, R., "Security Guidance for Critical Areas of Focus in Cloud Computing," V2.1, ***Cloud Security Alliance***, 2009.

[6] He, Q., Li, Z. and Zhang, X., "Data Deduplication Techniques," ***Proceedings of the International Conference on Future Information Technology and Management Engineering***, pp. 430-433, 2010.

[7] Hua, N., Zhao, H., Lin, B. and Xu, J., "Rank-Indexed Hashing A Compact Construction of Bloom Filters and Variants," ***Proceedings of the IEEE International Conference on Network Protocols***, pp. 73-82, October 2008.

[8] Papadopoulos, A. and Katsaros, D., "A-Tree: Distributed Indexing of Multidimensional Data for Cloud Computing Environments," ***Proceedings of the Third International Conference on Cloud Computing Technology and Science***, pp. 407-414, 2011.

[9] Thwel, T.T. and Thein, N.L., "An Efficient Indexing Mechanism for Data Deduplication," ***Proceedings of the International Conference on the Current Trends in Information Technology*** (CTIT), pp. 1-5, 2009.

[10] Tsuchiya, Y. and Watanabe, T., "DBLK: Deduplication for Primary Block Storage," ***Proceedings of the IEEE 27th Symposium on Mass Storage Systems and Technologies***, pp. 1-5, 2011.

[11] Wei, J., Jiang, H., Zhou, K. and Feng, D., "DBA: A Dynamic Bloom Filter Array for Scalable Membership Representation of Variable Large Data Sets," ***Proceedings of the 19th International Symposium on Modeling, Analysis & Simulation of Computer and Telecommunication Systems***, pp. 466-468, 2011.

[12] Zhang, S., Zhang, S., Chen, X. and Huo, X., "Cloud Computing Research and Development Trend," ***Proceedings of the Second International Conference on Future Networks***, pp. 93-97, 2010.

[13] Dutch, M., Data Management Forum Data Deduplication & Space Reduction SIG Co-Chair EMC Senior Technologist, "Understanding data deduplication ratios," June 2008. URL: http://storage.ctocio. com.cn/imagelist/2009/ 222/l3pm284d8r1s. pdf.

[14] More Google Product, October 2012, http:// www.google.com/options