設計並實現以建構Windows核心驅動方式之分散式資料儲存架構

邱日清　　楊承澔　　高志宏*
國立中山大學電機系
gunny2205@hotmail.com

摘要

分散式檔案系統是目前大資料(Big Data)網路系統中主要的基礎架構之一。在本篇論文中，我們提出了一套新的分散式資料儲存架構，這個架構突破了 Windows 7 作業環境中的系統核心，利用載入核心模式驅動程式的方式，建立起整體網路系統的工作平台。

在這架構中大致上可分為三個部分:客戶端、主控端和儲存端。在客戶端的部份我們除了設計一個可供使用者操作的介面外，同時利用 Event I/O 的觀念，建立了一套核心驅動程式，使其可直接與應用層溝通，並且傳遞資料；而在主控端的部分，則是負責管理所有的儲存端，並且接收客戶端連線要求，同時我們在主控端的核心驅動中，建立了一系列的資料結構，負責記錄客戶端分散檔案的邏輯儲存路徑；最後在儲存端，則是利用多台遠端電腦，透過一致性網路存取命令實作出的邏輯硬碟，提供給客戶端使用。

關鍵詞：分散式檔案系統、視窗核心模式驅動程式、大資料網路系統

Abstract

Distributed file system is one of the basic architecture in the Big Data network system. In this paper, new distributed data storage architecture is proposed, achieving a major breakthrough in the Windows 7 operating system. The kernel-mode driver is used to establish the network working platform.

The research can be divided into three parts: client, master and storage. For the client, users can operate the system with the designed interface, which uses the idea of the event I/O to communicate with a kernel driver. The master is responsible for managing storages, and receiving commands from the client. A series of data structures are also established to record the client's file logical path. For the storage part, consistency commands make all the remote computers transformed into logical disks, which are provided to the client.

Keywords: **Distributed File System、Windows Kernel Mode Driver、Big Data Network**

1. 前言

近年來隨著網路頻寬的提升，使用者透過網路來存取資料變得越來越容易。對於網路檔案的傳輸，最被廣泛運用的策略就是「分散式」的檔案處理，除了可以減少伺服器的負擔，還可以降低網路頻寬的需求。於本論文中我們提出了一套新的分散式資料儲存系統，擁有操作介面的客戶端、負責管理資料流的主控端和負責處理大量檔案存取的儲存端。為了讓系統有更大的兼容性，我們深入到 Windows 7 系統核心中，利用核心的驅動程式去實作分散式的網路資料儲存架構，使得整個系統能更貼近底層。

為了使分散式資料儲存架構能建構在 Windows 7 作業核心中，我們除了去研究 Windows 核心驅動程式在系統中的工作情形之外，還必須要學習核心網路模組 WSK (Winsock kernel)的操作介面，利用其定義的各種函示，如 WskAccept、WskSocketConnect，負責建立整個系統架構間的溝通橋樑；WskSend、WskReceive 等，在架構好的網路空間中完成檔案的接發工作。

最後整個分散式資料儲存的架構，則是參考了一些典型的分散式檔案系統設計[1][2]，提出了一套類似 Google File System，擁有單一 master 和多個 chunk server 的架構，並且擁有客戶端應用程式介面、客戶端核心驅動程式、主控端核心驅動程式以及儲存端核心驅動程式。

2. 背景與相關研究

分散式檔案系統主要的優勢為增加檔案的可靠性，以及降低伺服器的負載與網路頻寬的需求。典型的分散式檔案系統是將檔案儲存於各個伺服器裡，並提供存取介面讓使用者如同面對單一台伺服器。使用者透過操作介面控制客戶端軟體，客戶端軟體則會跟伺服器要求資源，伺服器則透過網路命令去存取後端的伺服器資源，並將相關資源給予客戶端軟體。這邊舉例幾個目前常見的分散式檔案系統:

SUN 的 NFS(Network File System)[3]，其允許 Client 和 Server 透過 Remote Procedure Call 來共享檔案目錄，其中包括檔案搜尋、列出檔案清單等動作，然而它使用單一主機分享資源的方式，並不適於大型的分散式系統。

Andrew File System(AFS)[1]則是適用於大型的

分散式系統，其在使用端提供快取機制加快存取速度，並且使用 Access Control Lists 來管理 User 和 Group。

Alex File System[4]則是將 FTP 的檔案系統轉化成 Unix 的檔案系統，使用者可使用傳統檔案存取方式來使用 FTP 的資源。

Google File System[2]則強調檔案系統的可靠性，整體是由一台 Master 和若干台 Chunk servers 所構成，其中單一 Master 是為了簡化設計並且只負責處理 metadata[5][6]，多個 Chunk servers 則是負責資料傳輸等負擔較重的工作。Metadata 存放在記憶體中，且沒有目錄的存在，所有的檔案皆以絕對路徑表示，

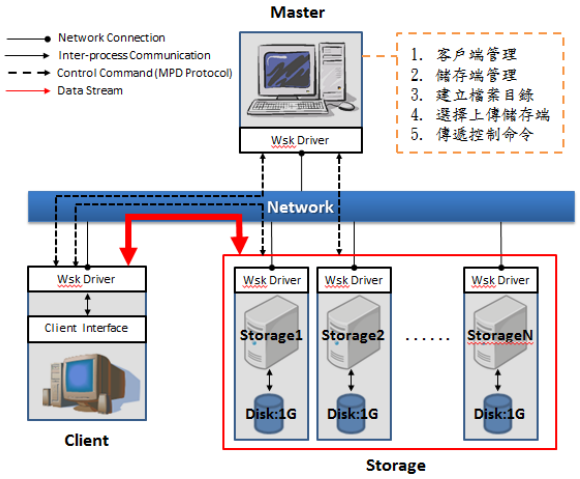

圖 1 系統架構

3. MPD Distributed Data Storage Architecture

在本篇論文中，我們提出了一套類似 Google File System, 擁有單一 master 多個 chunk server 的分散式資料儲存架構。其中我們突破了 Windows 7 作業環境的系統核心，並利用載入核心模式驅動程式的方式，來實現我們所提出的整體網路系統的工作平台，稱之為「MPD Distributed Data Storage Architecture」。

在這個章節中，首先會介紹整個 MPD Distributed Data Storage Architecture 的系統架構。接著我們將依序從客戶端、主控端和儲存端的軟體架構層面做介紹，最後舉例說明我們檔案在上傳模式和下載模式中的作業流程。

3.1 系統架構

如圖 1，在我們的系統架構中，可以分為三個部分來看，分別是 Client 端、Master 端還有 Storage 端，這三個部分都是透過我們的網路 Network 來做溝通。

首先，我們看 Master 的部分，在我們的 Master 裡面，他負責客戶端還有儲存端的管理、建立檔案的目錄、還有在客戶端上傳檔案時要負責選擇儲存端，跟傳遞控制命令。

在儲存端的部分，底層是我們實體的儲存媒介；而驅動程式的部分，我們會透過一致性的網路存取命令，實現檔案資源的保存和操作，並且與客戶端進行資源的存取。

最後在客戶端的部分，我們會提供一個客戶端操作介面(Client Interface)，負責與客戶端核心驅動程式做溝通，而客戶端的核心驅動程式，則會利用我們自定義的 MPD Protocol 作為溝通協定，跟我們的主控端做存取命令的操作，或對儲存端做檔案資源的傳輸。

3.2 客戶端應用程式介面

圖 2 為我們客戶端應用程式介面的軟體架構，當進行資料上傳操作時，只需要把檔案拖曳到我們的應用程式，應用程式就會自行去抓取檔案的資訊，例如 GetFileName 此副程式，利用 strrchr 函式可獲得上傳檔案的名稱，GetFileSize 此副程式，利用 fseek 的方式可獲得此檔案的大小，最後將兩個參數組合成指令 INFO(表 1)，透過 Win32 API 與核心驅動程式溝通。

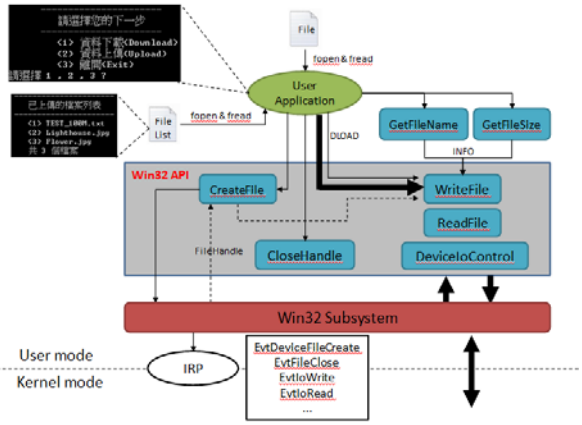

圖 2 客戶端應用程式介面軟體架構

在資料下載操作時，我們在已上傳的檔案列表中選擇想下載的檔案，發送 DLOAD 指令給客戶端核心驅動，表 1 中列出了 DLOAD 指令的形式，包含要下載的檔案名稱，利用「^」符號作為參數的區隔。

表 1 應用層向核心驅動請求命令

Request Command	Remark
INFO^File Name^File Size	檔案上傳，包含要上傳檔案名稱及檔案大小
DLOAD^File Name	檔案下載，File Name 為要下載的檔案名稱

3.3 客戶端核心驅動

圖 3 與圖 4 分別畫出了客戶端核心驅動在上傳與下載模式中的軟體架構。

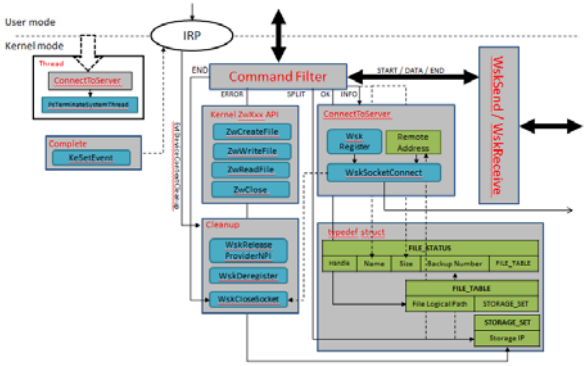

圖 3 客戶端核心驅動上傳模式軟體架構

在上傳模式中，當客戶端下達上傳指令(INFO)時，我們會先將此時要上傳的檔案資訊儲存在我們自定義的結構(FILE_STATUS)內，這有利於在下載模式中，快速地找尋到本身想要下載的檔案，而不用每次都向 Master 要求上傳資料表；接著利用 WskSocketConnect，連接 Master，將此時所要上傳的資訊傳送出去，當 Master 收到資料後，會發送指令(SPLIT)，告知客戶端核心驅動此資料要切割成幾份、發送至那些儲存端，因此客戶端核心驅動再透過 WskSocketConnect 連接儲存端，同時將成功連線的 IP 儲存在我們自定義的結構(STORAGE_SET)內；當資料完成傳輸，可收到儲存端回送的指令(OK)，得知此時檔案在儲存端的 File Logical Path，儲存在自定義的結構(FILE_TABLE)內，同步更新 Master 中的 FILE_STATUS；若是連接的 Storage 發生問題，則會回傳指令 ERROR，此時客戶端核心驅動將會刪除 STORAGE_SET 內的 Storage IP。

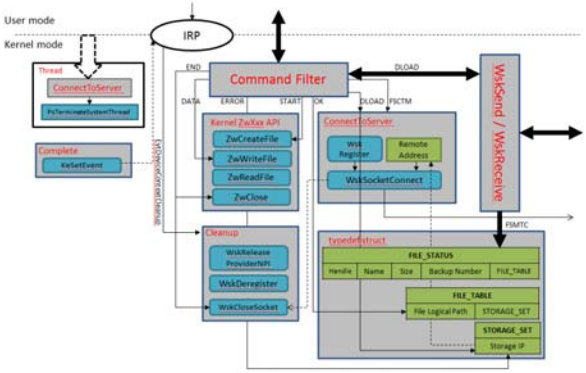

圖 4 客戶端核心驅動下載模式軟體架構

在下載模式中，收到 DLOAD 指令後，先根據要下載的資料名稱在本身的 FILE_STATUS 找尋，找到後根據 STORAGE_SET 內的 IP 連接儲存端，成功連線後，依序傳遞 START(開啟檔案)、DATA(檔案傳輸)、END(關閉檔案)等命令，若是在自定義的結構內找尋不到我們想要下載的資料，此時要透過 FSCTM，向 Master 要求更新自身的 FILE_STATUS， Master 透過 FSMTC 回傳 FILE_STATUS 給客戶端核心驅動。

3.4 主控端核心驅動

圖 5 與圖 6 分別畫出了主控端核心驅動在上傳與下載模式中的軟體架構。主控端並不負責檔案的傳輸，只負責控制命令的傳遞，檔案不經主控端可避免主控端有過重的負擔，同時在多人連線下，能有更高效率的處理。當核心驅動程式執行時，我們開啟一個執行緒，負責對外連線的監聽，當接收到外來的連線後，先由 Command Filter 判斷此時連線的為客戶端，抑或是新增的儲存端，同時抓取已建立連線的 IP 位址，分別存進 USER 結構(圖 7)、STORAGE 結構(圖 8)中，最後重複監聽下一個新的連線。

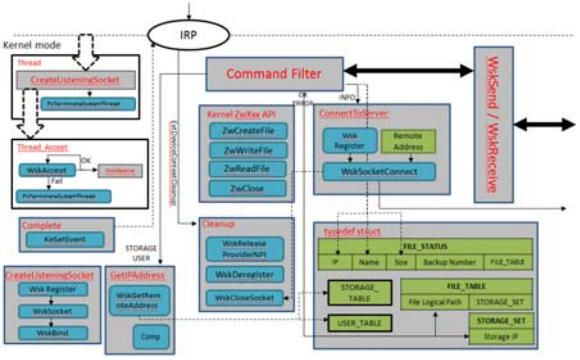

圖 5 主控端核心驅動上傳模式軟體架構

在上傳模式中，當接收到客戶端核心驅動傳遞來的 INFO 指令，先將資料來源的 IP、檔案名稱及檔案大小存進自定義的結構(FILE_STATUS)，同時根據資料的大小，由我們固定的 Block Size 可得知資料切割的塊數，加上自訂備份的次數與所選擇儲存端點的 IP、Port，合成 SPLIT 指令回傳給客戶端核心驅動，等待客戶端回傳 OK 指令，若是 ERROR，則將錯誤的 IP 位置從 STORAGE 結構內刪除。

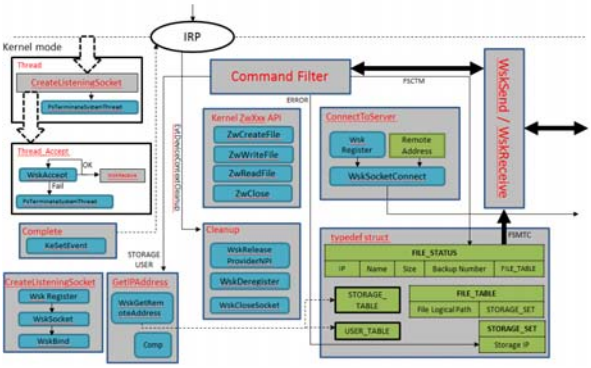

圖 6 主控端核心驅動下載模式軟體架構

在下載模式中，由於 Client 可自行由本身紀錄的 FILE_STATUS 連接儲存端，因此在下載模式中，

主控端只需要負責接收客戶端的回傳訊息；當收到命令 OK，則更新 FILE_STATUS 內的 File Logical Path；收到 ERROR，則將錯誤的 IP 位置從 STORAGE 結構內刪除；若是收到 FSCTM，表示 Client 要更新本身的 FILE_STATUS，則回傳 FSMTC 命令。

USER_TABLE		
IP	Socket	Ready

圖 7 USER 架構

STORAGE_TABLE			
IP	Port	Free Space	Ready

圖 8 STORAGE 架構

3.5 儲存端核心驅動

圖 9、10 為儲存端在上傳與下載模式中的軟體架構，較主控端和客戶端不同的是，在儲存端除了在每次安裝要向主控端註冊(發送 STORAGE 指令)外，還要隨時監聽來自於客戶端或者主控端的命令要求，這些要求包含傳輸的資料以及控制的指令。

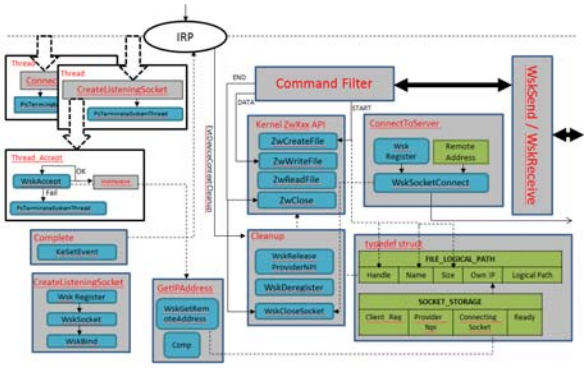

圖 9 儲存端核心驅動上傳模式軟體架構

在上傳模式中，首先利用函式，獲得已建立連線的客戶端 IP 位址，同時將 IP 儲存到資料結構 FILE_LOGICAL_PATH 中的 OwnIP，一旦收到客戶端的 START 指令，則開始調用 CreateFile 函式，在硬碟中建立一個檔案，供使用者寫入，期間 DATA 開頭的指令，代表 DATA 後的參數即為上傳檔案資料，調用 WriteFile 將資料寫入到硬碟中，最後 END 指令，調用 ZwClose，釋放 Handle 同時關閉檔案。

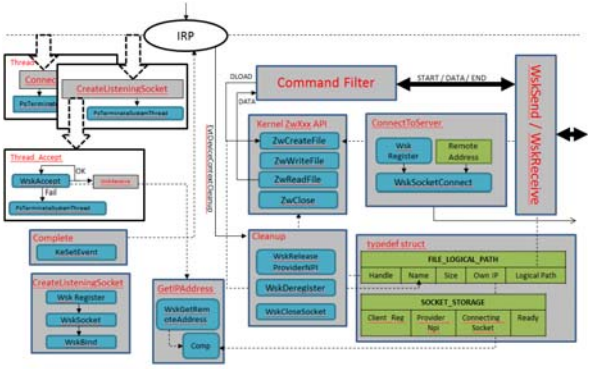

圖 10 儲存端核心驅動下載模式軟體架構

在下載模式中，當接收到一個新的連線時，第一個步驟就是根據 IP 位址，先判斷本地儲存端是否有使用者的儲存檔案，透過 WskGetRemoteAddress 與 FILE_LOGICAL_PATH 內的 OwnIP 做字串的比對。接著根據 DLOAD 指令的檔案邏輯路徑，去尋找使用者在此儲存端的資料片段，透過 ZwReadFile 函式將檔案回傳給客戶端，最後同樣利用 START、DATA、END 的指令告知客戶端。

對於分割檔案在儲存端的邏輯位置，我們直接以檔案擁有人的 IP 位址、檔案名稱與分割區塊，作為檔案的命名，並以「^」作為檔案名稱參數的分割符號，以便在執行檔案下載時，可透過邏輯路徑抓取到目的檔，如圖 11 所示。

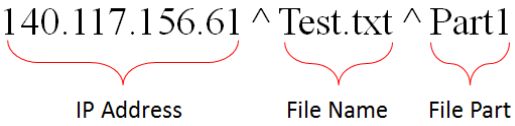

圖 11 儲存端檔案命名方式

3.6 傳輸範例

圖 12 畫出了系統在檔案上傳時的運作流程。

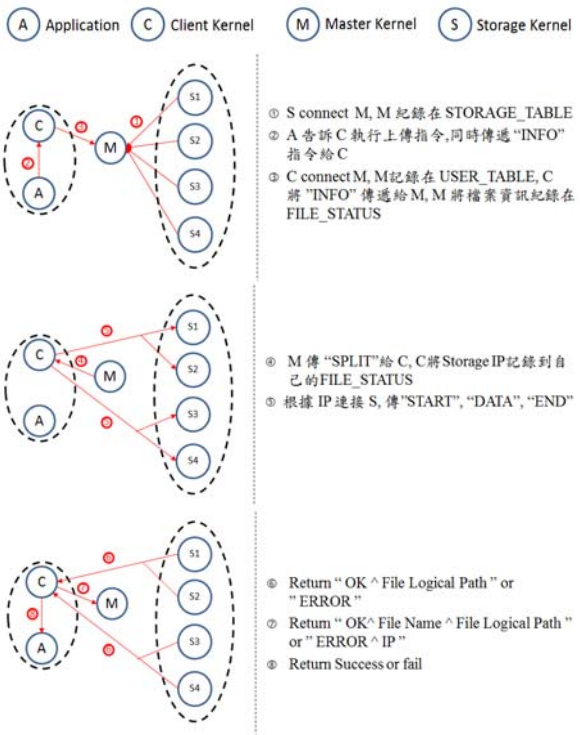

圖 12 檔案上傳範例圖

圖 13 畫出了系統在檔案下載時的運作流程。

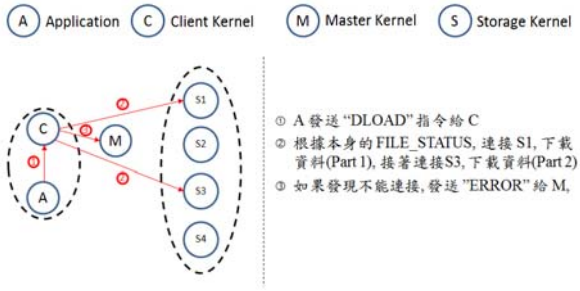

圖 13 檔案下載範例圖

4. 實作和實驗結果

此章節首先會介紹我們實驗模擬的環境，包含電腦的配備、網路卡速度、儲存端的數量等。接著介紹客戶端的應用程式介面，及如何利用我們所提供的程式介面完成資料上傳、下載動作。最後我們分別會測試不同檔案在上傳、下載模式中所花的時間，以及檔案在上傳、下載模式中的平均傳輸速率，最後與直接儲存在本地硬碟所耗費的時間做相比。

4.1 實驗環境

我們將 9 台電腦、1 台路由器和 3 台交換器佈置如圖 14，其設備規格如表 2。其中 8 台電腦當作 Storage、1 台電腦當作主控端，分別安裝我們的 Storage Kernel Driver 及 Master Kernel Driver。

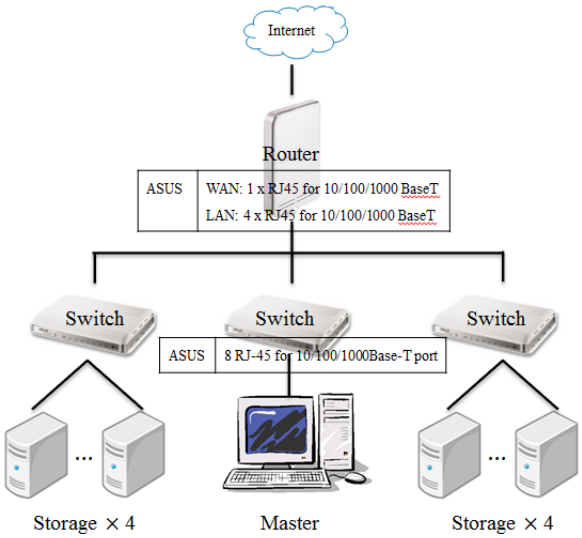

圖 14 實驗環境設備圖

表 2 實驗設備規格

電腦配備	
處理器	AMD Athlon(tm) 64 X 2 Dual Core Processor 5000+ 2.60GHz
記憶體	2GB
系統類型	Windows7 64 位元作業系統
網路卡速度	100 Mbps

4.2 實驗一，上傳模式

圖 15 為使用者點開客戶端應用程式時的介面，我們提供三種操作指令(資料下載、資料上傳、離開)。在資料上傳的模式中，我們只需要將我們的檔案，直接拖曳到此客戶端應用程式上，資料就會自動被程式所抓取、分析並上傳至儲存端。圖 16 為檔案上傳成功執行的畫面。

圖 15 客戶端應用程式介面

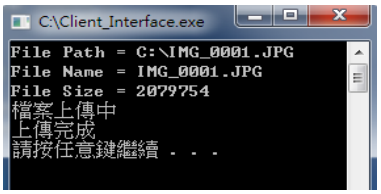

圖 16 客戶端應用程式執行檔案上傳

圖 17 Upload 為我們上傳各種不同大小的檔案所花時間。並且固定切割大小為 64MB，每筆傳送資料大小為 1KB。Local 為檔案不經由網路發送，直接儲存在本地端所花費的時間，最後圖 18 為不同檔案上傳的平均速度。

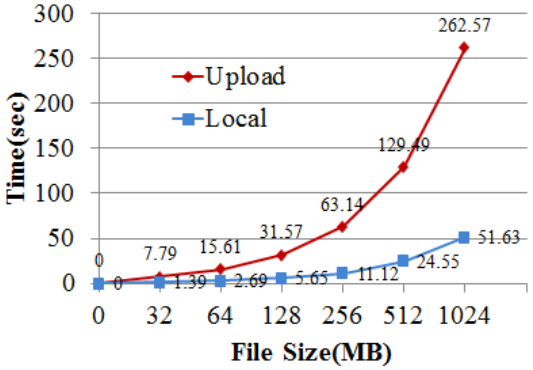

圖 17 檔案上傳花費時間

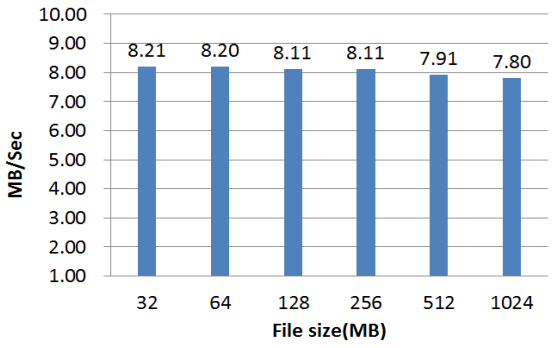

圖 18 檔案上傳平均速度

4.3 實驗二，下載模式

當完成資料上傳後，再次打開客戶端的應用程式介面(圖 19)，此時在已上傳的檔案列表內，可以看到剛剛上傳的檔案已出現在檔案列表中，因此在選擇資料下載時，只需要輸入要下載檔案其對應的數字，即可成功從網路空間拿回自己想下載的資料了，而下載下來的資料則預設會建立在 C 槽中，如圖 20。

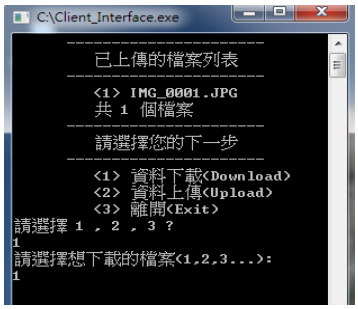

圖 19 客戶端應用程式介面

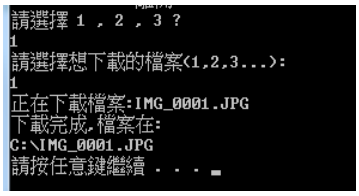

圖 20 客戶端應用程式執行檔案下載

圖 21 Download 為我們將剛剛上傳實驗中的檔案再次下載回來，記錄所花的時間。Local 同樣為檔案不經由網路發送，直接儲存在本地端所花費的時間，最後圖 22 為下載的平均速度。

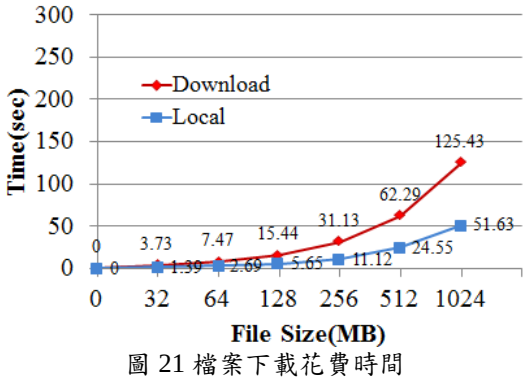

圖 21 檔案下載花費時間

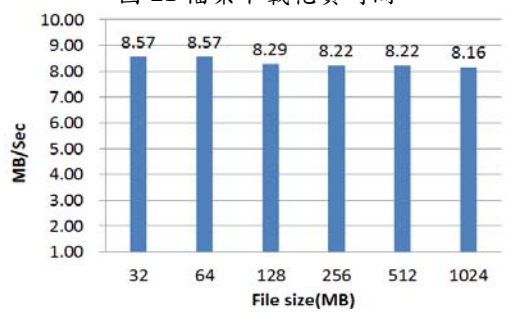

圖 22 檔案下載平均速度

4.4 分析與探討

我們將實驗數據整理如表 3，我們可以看到在上傳和下載模式中，上傳所花的時間大約比下載多了兩倍。會造成如此的原因，是因為我們在上傳的時候必須將檔案做完全備份，因此同一份檔案客戶端必須比下載多一次傳輸的時間。

至於平均的速度可能會根據客戶端的距離，而有所不同的傳輸速度，不過我們可以看到在這次實驗中，下載的平均速度較上傳快了 1.03 倍，我認為是在上傳的時候，我們的檔案是從客戶端的應用層被拖曳進去，也就是說多了一個由客戶端應用程式傳輸到客戶端核心驅動的步驟；而下載時，儲存端的檔案資源，是直接被儲存端核心驅動程式所讀取，不需要經過應用層，因此下載的速度較上傳快。

表 3 實驗數據整理

Mode \ File Size	32MB	64 MB	128 MB	256 MB	512 MB	1024 MB	平均速度 MB/sec
Upload	7.79s	15.61s	31.57s	63.14s	129.49s	262.57s	8.057
Download	3.73s	7.47s	15.44s	31.13s	62.29s	125.43s	8.338
Local	1.39s	2.69s	5.65s	11.12s	24.55s	51.63s	20.777

5. 結論

本論文提出了一套新的分散式資料儲存架構，它是運行在 Windows 7 作業系統核心模式下，透過我們編譯的核心驅動程式，實現分散式的資料儲存。

透過我們實作的分散式資料儲存架構，我們進行了兩個實驗，分別是上傳實驗和下載實驗。在固定了檔案切割大小為 64MB，每筆檔案傳輸大小為 1KB 的前提下，分別記錄了不同檔案大小的資料傳輸時間。在上傳實驗中，平均的上傳速度約為 8.057MB/sec。而在下載實驗中，平均的下載速度約為 8.338MB/sec。

參考文獻

[1] OpenAFS,2012,OpenAFS, Available: http://www.openafs.org/

[2] Google, 2003, Google File System,Available: http://static.googleusercontent.com/external_content/untrusted_dlcp/research.google.com/zh-TW//archive/gfs-sosp2003.pdf

[3] SUN, 1985, Design and Implementation or the Sun Network File system,Available: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.14.473

[4] Alex FTP File system,Available: http://www.ludd.luth.se/~kavli/alex.html

[5] Min Xu, Junrui Zhou, Wei Zhou, Hong An, “CHMasters: A Scalable and Speed-efficient Metadata Service in Distributed File System”, 2011 12th International Conference on Parallel and Distributed Computing, Applications and Technologies, 2011, page 397-399

[6] Quan Zhang, Dan Feng, Fang Wang, “Metadata Performance Optimization in Distributed File System”, 2012 IEEE/ACIS 11th International Conference on Computer and Information Science, 2011, page 476-481