國立交通大學

資訊學院資訊科技（IT）產業研發碩士班

碩 士 論 文

以 P2P 為基礎之巨量發佈系統

– 伺服端與系統架構研究

The Study of Massive Deployment System Based on P2P Technology

– Servers and System Architecture

研 究 生：林哲毅

指導教授：吳毅成　教授

中 華 民 國 九 十 七 年 九 月

以 P2P 為基礎之巨量發佈系統

－ 伺服端與系統架構研究

The Study of Massive Deployment System Based on P2P Technology

－ Servers and System Architecture

研 究 生：林哲毅　　Student：Che-Yi Lin

指導教授：吳毅成　　Advisor：Yi-Cheng Wu

國 立 交 通 大 學

資訊學院資訊科技（IT）產業研發碩士班

碩 士 論 文

A Thesis

Submitted to College of Computer Science

National Chiao Tung University

in partial Fulfillment of the Requirements

for the Degree of Master

in

Industrial Technology R & D Master Program on

Computer Science and Engineering

Aug 2008

Hsinchu, Taiwan, Republic of China

中華民國九十七年九月

以 P2P 為基礎之巨量發佈系統
– 伺服端與系統架構研究

研究生：林哲毅　　指導教授：吳毅成

國立交通大學資訊學院產業研發碩士班

摘要

本論文提出一套以 P2P 技術為基礎的快速檔案散佈系統，能夠節省伺服器頻寬、提升客戶端下載速度。而考量到企業使用的狀況，在設計上提供良好的擴充性，並具有容錯能力，能夠在系統一部份毀損的情況下，持續提供服務。

此系統名為 Massive Deployment System，簡稱 MDS，主要含有兩個伺服端元件 MDS Server 與 Super Seeder，以及一個客戶端元件 MDS Client。本論文將針對伺服端元件進行詳細研究與探討。

The Study of Massive Deployment System Based on P2P Technology – The Servers and System Architecture

Student: Che-Yi Lin Advisor: Yi-Cheng Wu

Industrial Technology R & D Master on
Computer Science College
National Chiao Tung University

Abstract

This paper presents a system based on P2P technology for the fast file deployment, which can save huge bandwidth of servers, enhance client download speed. This system called Massive Deployment System, or as MDS, contains two server components MDS Server and the Super Seeder, and a client component MDS Client. This paper only focused on study the server components.

誌謝

感謝我的指導教授，吳毅成先生，一位願意把大部分時間花在學生身上的好老師。在他的指導之下，我學到了相當多的知識，也漸漸地瞭解，什麼是作研究。

他對我總是充滿信任，讓我盡情發揮所長，並給予我許多肯定。我期許自己，未來能不辜負老師的期望，開拓自我。

感謝我的父母，許多年來對我的包容，並在我選擇的方向上，持續給予支持與肯定。

感謝實驗室的學長、同學與學弟們，因為有你們在，讓這兩年充滿著歡樂；感謝 P2P 小組，以及我的工作伙伴，鼎量，和你討論時，你的想法總是能讓我有所啟發，令我不致一意孤行。

最後，我要感謝我的女友恬恬，一直陪伴在我身邊，給予我關心，讓我有動力，能夠完成這篇論文。

僅以此論文，獻給每位支持我的朋友。

目錄

圖目錄

表目錄

第一章、 緒論

本章會說明 P2P 在目前網路上的應用行為，並提出本論文的動機與目的，最後會說明整篇論文的論文大綱。

1.1 研究動機與目的

P2P (Peer-to-Peer)為目前網路上相當熱門的話題，眾多應用相應而生，其中最受到歡迎的便是 P2P 檔案分享服務。藉由每個使用者貢獻自己的上傳頻寬，來達到快速且有效率的檔案分享，使得網際網路上資訊的交換變得更加迅速。許多著名的軟體計畫如 FreeBSD、Linux 等，紛紛使用 P2P 的方式，來發佈其所發行的軟體，以降低發行時伺服器的負載與頻寬使用。

雖然使用現有的 P2P 方案來發佈檔案相當有效率，但其目的僅為傳送檔案至客戶端，並未考量到時效性。著名之 P2P 方案如 eDonkey[1]、BitTorrent[2]，皆遵循公平原則，讓提供大量上傳的客戶端，能夠快速完成檔案；而提供小量上傳的客戶端，則需要數倍的時間才能完成檔案。

對於常需發佈重要且有時效性檔案的公司而言，目前大部分都

還是採用傳統的 Client / Server 方式，以 HTTP 或 FTP 來提供檔案下載，如 Microsoft 微軟公司的 Hotfix 修正檔，以及眾多線上遊戲公司，經常性釋出的遊戲更新檔。

這些採用傳統 Client / Server 架構的公司，都會面臨當某重大更新釋出時，伺服器瞬間負荷過高、頻寬使用也瞬間暴衝，遠高於平時的使用量，而影響到公司其他服務等眾多問題。

本論文的目的，即是設計並實作一套基於 P2P 技術的檔案發佈系統，除了具有 P2P 節省伺服端頻寬的優點外，還能夠讓欲發佈的檔案，傳送至客戶端的整體速度提升。並在設計上考量擴充性與容錯能力，以期能夠在企業部署、應用上達到高可用度的水準。

1.2 論文章節說明

第二章將會介紹 P2P 相關技術背景以及發展現況；而第三章會對本系統的架構與功能運作方式作一個介紹；第四章會說明系統實作上的細節，以及開發過程中所解決的問題；第五章則說明實驗設計與結果的分析，最後，第六章會提出結論以及未來的工作與展望。

第二章、 背景說明

本章節介紹 P2P 技術背景，以及與論文主題相關的各種 P2P 檔案分享 (file sharing) 應用技術。

2.1 P2P技術

P2P (Peer-to-Peer) 為一種新型的網路技術，藉由使用在 P2P 網路中客戶端的頻寬與計算能力，而非依賴少數幾台伺服器的服務，來快速完成目的。

最早 P2P 技術的設計為非中央化，也就是不需中央伺服器，網路上每個客戶端皆有相同的地位，同時具備伺服端與客戶端的能力，這種 P2P 類型我們稱之為「純 P2P 技術」。

然而因應用上的特性與需求，出現了所謂的「混合 P2P 技術」。此類 P2P 網路有著中央伺服器，負責保存每個客戶端的資訊，並回覆來自客戶端對於這些資訊的請求。

純 P2P 以及混合 P2P 各有其優缺點，也各適用於不同的應用環境。目前的 P2P 技術發展，已逐漸地將兩者的優點結合，並已套用在一些廣受歡迎的 P2P 應用上。

2.2 P2P相關應用

以下將介紹數種使用 P2P 技術的相關應用。

2.2.1 Napster

1999 年，年僅 19 歲的 Shawn Fanning 創造了 Napster[3]，第一個應用 P2P 技術且受到大眾歡迎的音樂分享服務。Napster 的出現影響了人們使用網路的方式，使得在網路上交換 MP3 音樂檔案相當容易。但也因此受到唱片公司的控告。

正在 Napster 流行之際，法院在 2000 年下令 Napster 網站關閉，停止服務。雖然 Napster 已經消失，但卻為後來眾多 P2P 檔案分享服務，開啟了一條道路。

在 2003 年，Napster 恢復營運，改為提供付費音樂下載的服務。最早以 P2P 技術提供免費音樂分享的 Napster 已不存在。

2.2.2 Gnutella

Gnutella[4] 是一種不需要中央伺服器的「純 P2P 技術」應用。最初由 AOL 美國線上的工程師在 2000 年撰寫，並以 GNU GPL 授權公開讓大家下載。雖然隔天就被 AOL 撤下，但 Gnutella 的 P2P 網路已經啟動。過了一段時間，Gnutella 的協定被逆向工程破解，從此

與 Gnutella 協定相容的客戶端便陸續誕生。

Gnutella 並沒有中央伺服器，只要網路上有兩個以上的客戶端，Gnutella 網路便會存在，因此很難被關閉。

現有的 Gnutella 客戶端大多為 Open Source，如 LimeWire、Gnucleus、Ares Galaxy、以及 Shareaza。

2.2.3 eDonkey2000

eDonkey2000[1]，簡稱 eDonkey 或 ed2k，由 Jed McCaleb 在 2000 年所創造，為一使用 P2P 技術的檔案分享服務。eDonkey 的中央伺服器用來尋找檔案，而檔案的傳輸則在客戶端之間進行。

eDonkey 首創將檔案切割成片段，讓客戶端能夠下載單一檔案內的不同片段，有效地讓頻寬使用分散至所有客戶端，而非擁有完整檔案的客戶端。

eDonkey 使用 MD4 摘要演算法來辨識文件，使得客戶端擁有的檔案有其獨一性，並可藉由向 eDonkey 伺服器查詢，來得到欲下載檔案的來源資訊。

在 2002 年，Hendrik Breitkreuz 集合了一群出色的程式設計師，

啟動了 eMule[5] 計畫，目的為創造相容 eDonkey、加入新功能、並擁有絕佳使用介面的新軟體。

在 eDonkey 停止發展後，eMule 已成為最受歡迎的 eDonkey 軟體，擁有超過 80%以上的佔有率，且已被移植到各種平台上。

2005 年，在美國唱片協會的要求下，eDonkey 網站關閉，也不再繼續發展。但 eDonkey 的 P2P 網路仍靠著 eMule 與眾多 eDonkey 相容客戶端，存在於網路上。

2.2.4 BitTorrent

2002 年，Bram Cohen 在 CodeCon 發表 BitTorrent[2][6] 軟體與其協定，為一 P2P 檔案分享機制。有別於以往眾多 P2P 檔案分享，BitTorrent 有著更利於檔案傳輸的特性。藉由 BitTorrent 協定中制訂的幾個規則，檔案的交換效率能更為提升。

BitTorrent 的規則如下：

1. Rarest Piece First Policy

   優先傳送最稀有的檔案片段。

2. Strict Priority

   盡快完成目前正在下載的檔案片段。

3. Tit-for-Tat Choking Policy

   依照對方給予的檔案片段數量，給予回報。

4. Optimistic Unchoking

   定時隨機挑選連線，以尋找能力更好的對象。

5. Anti-Snubbing

   若一段時間內，皆未從某對象收到片段，便不再上傳給它。

另外還有處理第一片檔案片段的 Random First Piece 以及最後數片檔案片段的 Endgame Mode。這些規則讓 BitTorrent 的客戶端，能夠盡可能地使用自己的上傳頻寬，以提升整體傳輸速度。

目前，有許多社群維護著自己的 BitTorrent 軟體，在這些社群的努力下，各項新功能被加入，BitTorrent 仍在快速成長中。

本論文實作系統中，客戶端 MDS Client、以及 MDS Client 與 Super Seeder 之間的通訊，便是參考自 BitTorrent。論文「以 P2P 為基礎之巨量發佈系統 - 客戶端與效能分析研究」[7] 將會詳細介紹 BitTorrent，以及我們所參考的細節。

第三章、 系統介紹

本章節將對 Massive Deployment System (巨量發佈系統，簡稱 MDS)的系統架構、運作流程與各項機制作詳細的介紹。

3.1 MDS架構

MDS 巨量發佈系統，依照 2.1 節所述，為一「混合型 P2P」應用。由中央伺服器保存、維護客戶端資訊，並回覆來自客戶端對彼此資訊的請求。

與其他相類似的 P2P 檔案分享機制相比，MDS 在檔案的發佈上，採用中央伺服器控管的方式，由此伺服器來決定現在要傳送的檔案，並主動將其資訊傳送客戶端。

而 MDS 為了能更快速地發送檔案至客戶端，加入了另一個，擁有完整檔案的伺服器，藉由有效利用此伺服器的頻寬，加上客戶端之間的 P2P 檔案片段交換行為，讓整體下載更加快速。

為了便於使用，我們將 MDS 分割為兩部分。一部份為系統核心，含有上述伺服器與客戶端元件，不需額外設定；而另一部份則為使用 MDS 的公司自行架設的元件，可依需求進行設定與調整。

MDS 架構，如圖 3-1 所示：

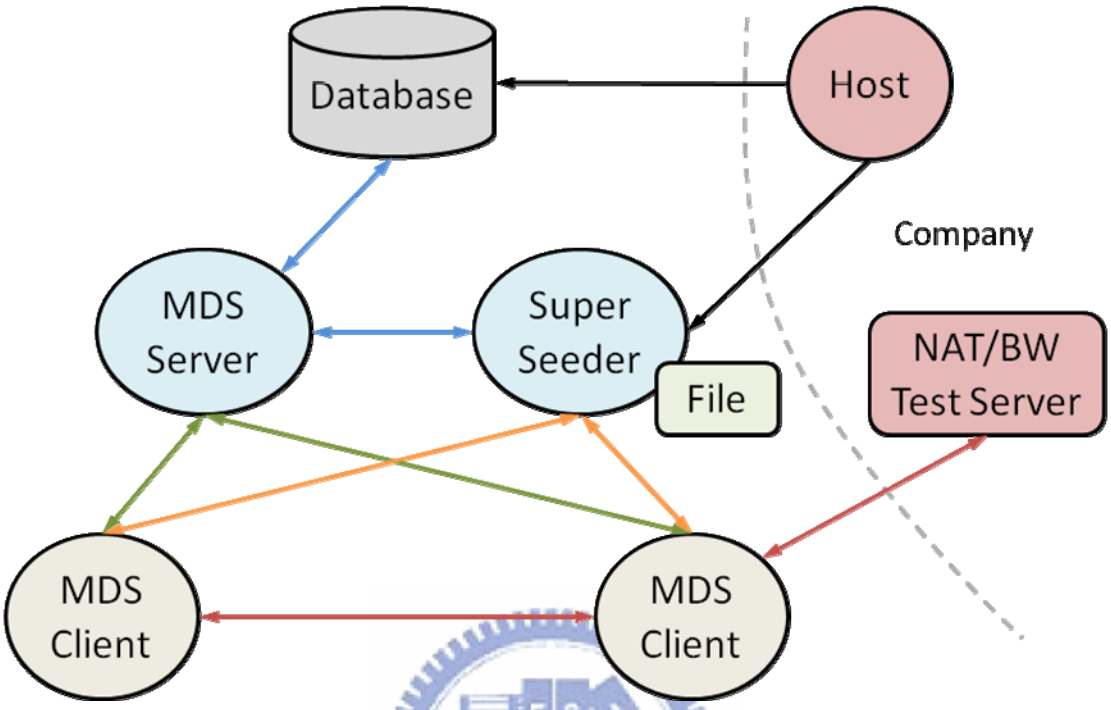

圖 3-1　MDS 架構圖

其中右側虛線以左，為 MDS 核心元件；而虛線以右則為使用 MDS 系統之公司，所需架設與操作的元件。

各元件說明如下：

- **File：**

  欲傳遞至 MDS Client 的檔案，可為單一檔案或資料夾。

- **File Signature：**

  未在架構圖中顯示。為 Host 針對 File 所製作的簽章，內含該 File 的結構、檢查碼與傳輸所需參數。

- **Host：**

  由使用 MDS 系統之公司所操作。其功能為：

  - 製作 File Signature。
  - 儲存 File Signature 至 Database。
  - 傳送 File 至 Super Seeder。

- **MDS Server：**

  為 MDS 中最重要的伺服端元件，其功能為：

  - 協調 Super Seeder 與 MDS Client。
  - 傳送 File Signature 至 Super Seeder 與 MDS Client。
  - 傳送 MDS Client List 至 MDS Client。

- **Super Seeder：**

  MDS 中負責散佈檔案的伺服端元件，其功能為：

  - 維護完整的 File。
  - 傳送 File 片段至 MDS Client。

- **MDS Client：**

  即為客戶端元件，其功能為：

  - 向 Super Seeder 要求 File 片段。
  - 與其他 MDS Client 交換所擁有的 File 片段。

- **Database：**

  在 MDS 中扮演資料儲存的角色，儲存的項目為：

  - File Signature 檔案簽章。
  - MDS Server、Super Seeder 與 MDS Client 資訊。
  - 系統記錄與報告。

- **NAT / Bandwidth Test Server：**

  提供 MDS Client 進行 NAT[18]類型與 Bandwidth 頻寬測試。依照其測試結果，來進行 MDS Client 之間的 NAT 穿透，以及調整 Super Seeder 傳送 File 片段、MDS Client 交換 File 片段之行為。

3.2 MDS檔案發佈流程

當使用 MDS 來發佈欲傳輸的檔案時，需要使用 Host 這個程式，來製作 File Signature、傳送 File 至 Super Seeder，以及將製作好的 File Signature 儲存到 Database。

檔案發佈流程如下：

1. Host 從欲傳送的 File 製作出 File Signature。
2. Host 將 File 透過 FTP Protocol 傳送至 Super Seeder。
3. Host 將 File Signature 儲存至 Database。

當 File 傳送至 Super Seeder 且 File Signature 儲存至 Database 後，檔案發佈即完成，此時便可以啟動 MDS Server 與 Super Seeder 來進行檔案傳輸。

3.3 MDS檔案傳輸流程

欲進行檔案傳輸，需要先啟動 MDS Server，由它來控管 Super Seeder 以及 MDS Client，並傳遞其個別所需要的資訊。在傳輸過程中，MDS Server 始終與 Super Seeder、MDS Client 保持連線，以監控它們的狀態。

檔案傳輸流程如下：

1. MDS Server 連線至 Database，取得初始化資訊與 File Signature。
2. Super Seeder 登入 MDS Server，取得 File Signature 後，依其內容初始化從 Host 傳送過來的 File，並檢查 File 完整性。
3. MDS Client 啟動，連線至 NAT / Bandwidth Test Server 進行 NAT 類型與頻寬測試。測試完成後，登入 MDS Server，將測試結果傳送至 MDS Server。
4. MDS Server 傳送 File Signature 給 MDS Client，並依照 MDS Client 的 NAT / Bandwidth 測試結果，回傳適當的 MDS Client List。MDS Client List 的第一項即為 Super Seeder。

5. MDS Client 將會先連線至 Super Seeder，取得一小部份 File 片段，接著連線至 MDS Client List 中其他 MDS Clients，互相交換 File 片段。

至此，MDS Client 將持續地向 Super Seeder 要求 File 片段，也會不停地和其他 MDS Client 交換彼此不同的 File 片段，直到取得所有片段，完成檔案傳輸。

3.4 MDS多伺服器架構

在 3-1 節中介紹的為 MDS 單一伺服器架構，即一台 MDS Server 搭配一台 Super Seeder。在一般的狀況下，此一架構即可正常運作。但若在大型企業中使用時，將會面臨數個問題：

- 使用者數量太龐大，超過單一 MDS Server 或 Super Seeder 的承載上限。
- 當任一伺服器發生網路中斷、程式發生錯誤、甚至機器當機時，服務將會中斷。

因此，在設計 MDS 時，便考量到未來在企業部署上，擴充彈性與容錯能力的需求，因此可輕易擴充至 MDS 多伺服器架構。

MDS 多伺服器架構具備下列優點：

- 負載平衡 (Load balancing)
- 提高可靠度 (Reliability)
- 提高可用度 (Availability)

以下介紹 MDS 多伺服器架構，如何達到負載平衡、提高可靠度以及可用度。

3.4.1 負載平衡 (Load balancing)

首先介紹多 Super Seeder 如何達到負載平衡：

下列圖中的元件同 MDS 架構圖，使用縮寫，DB 表示 Database，MS 表示 MDS Server，SS 表示 Super Seeder，MC 表示 MDS Client。

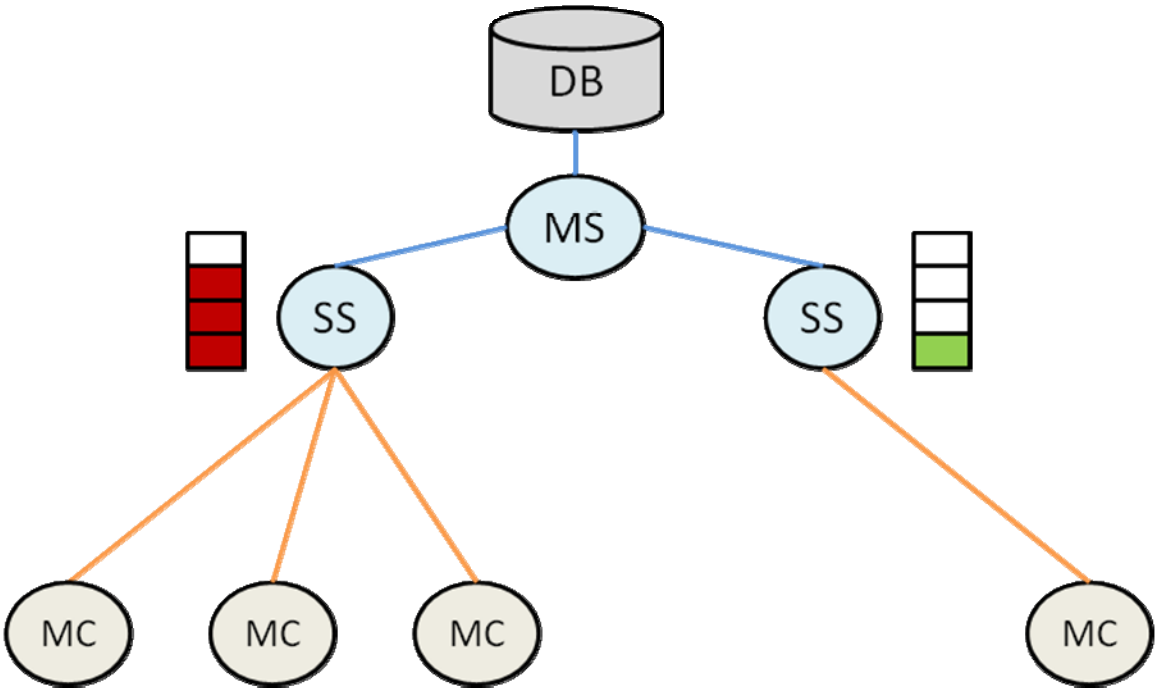

圖 3-2　多 Super Seeder 負載平衡 步驟一

在圖 3-2 中，中央的 MDS Server 控管兩台 Super Seeder。左方的 Super Seeder 正在服務三台 MDS Client，所以左方的負載計量顯示為三格；而右方的 Super Seeder 只有服務一台 MDS Client，因此右方的負載計量顯示為一格。

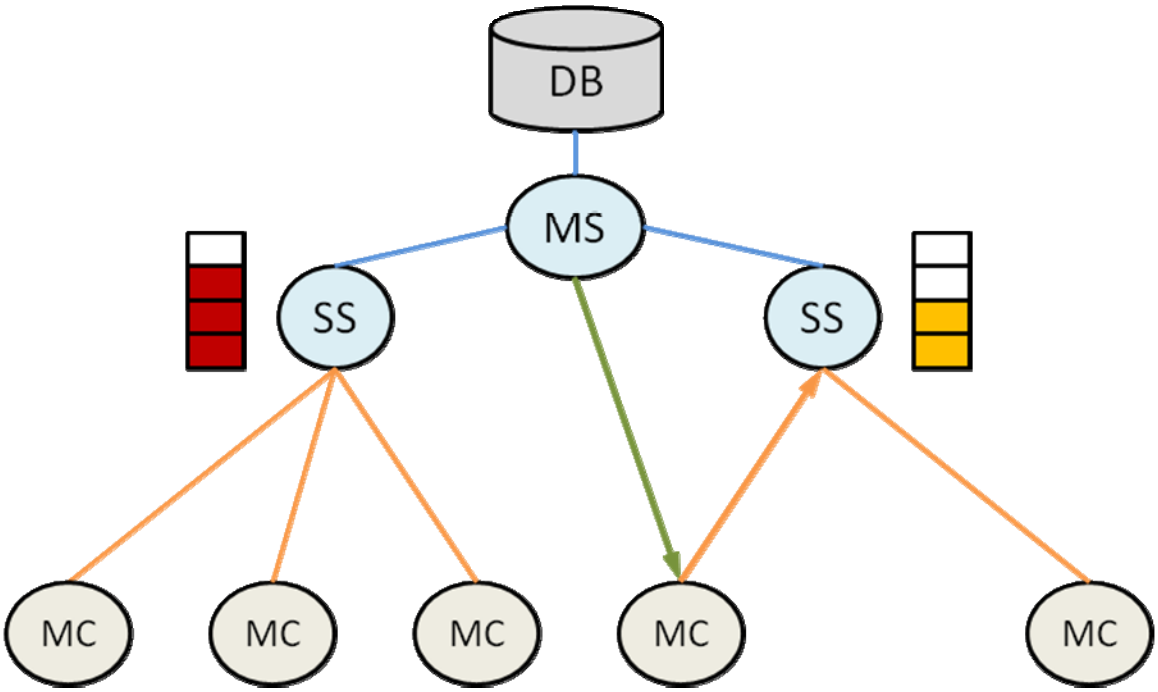

圖 3-3　多 Super Seeder 負載平衡 步驟二

如圖 3-3，現在，有一台新的 MDS Client 登入 MDS Server，MDS Server 依照 Super Seeder 所回報的負載量，告知此 MDS Client 去使用右邊負載較低的 Super Seeder，此時右方 Super Seeder 的負載計量顯示為二格。

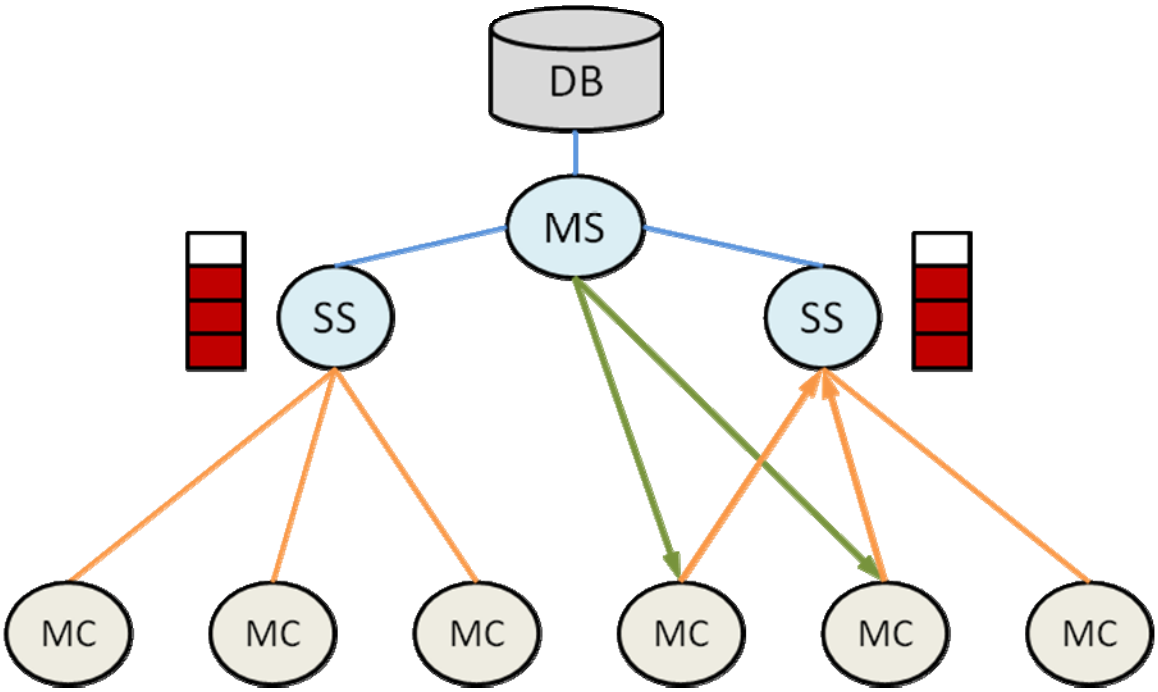

圖 3-4　多 Super Seeder 負載平衡 步驟三

如圖 3-4，而後，又有一台新的 MDS Client 登入 MDS Server，MDS Server 會依照 Super Seeder 的負載量，告知此 MDS Client 去使用右方的 Super Seeder。至此左右兩方的 Super Seeder 負載相同，顯示多個 Super Seeder 的負載平衡效果。

3.4.2 提高可靠度 (Reliability)

以下介紹多 Super Seeder 如何提高可靠度：

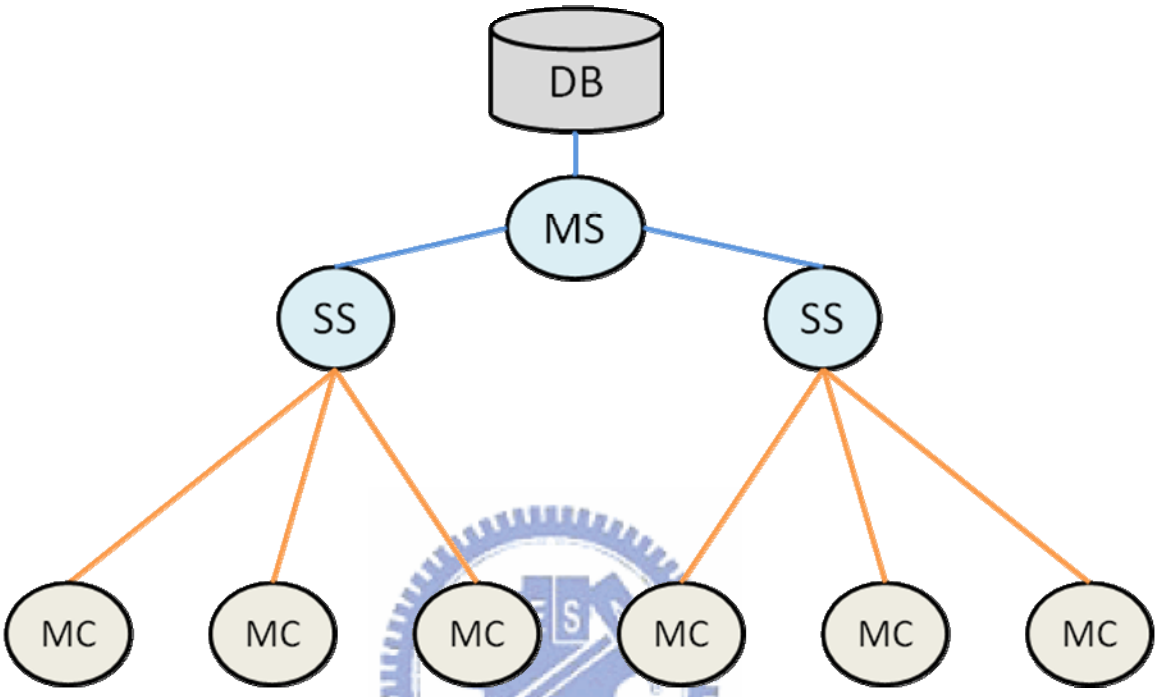

圖 3-5　多 Super Seeder 提高可靠度 步驟一

如圖 3-5，在上述負載平衡後，左右方的 Super Seeder 皆服務三台 MDS Client。

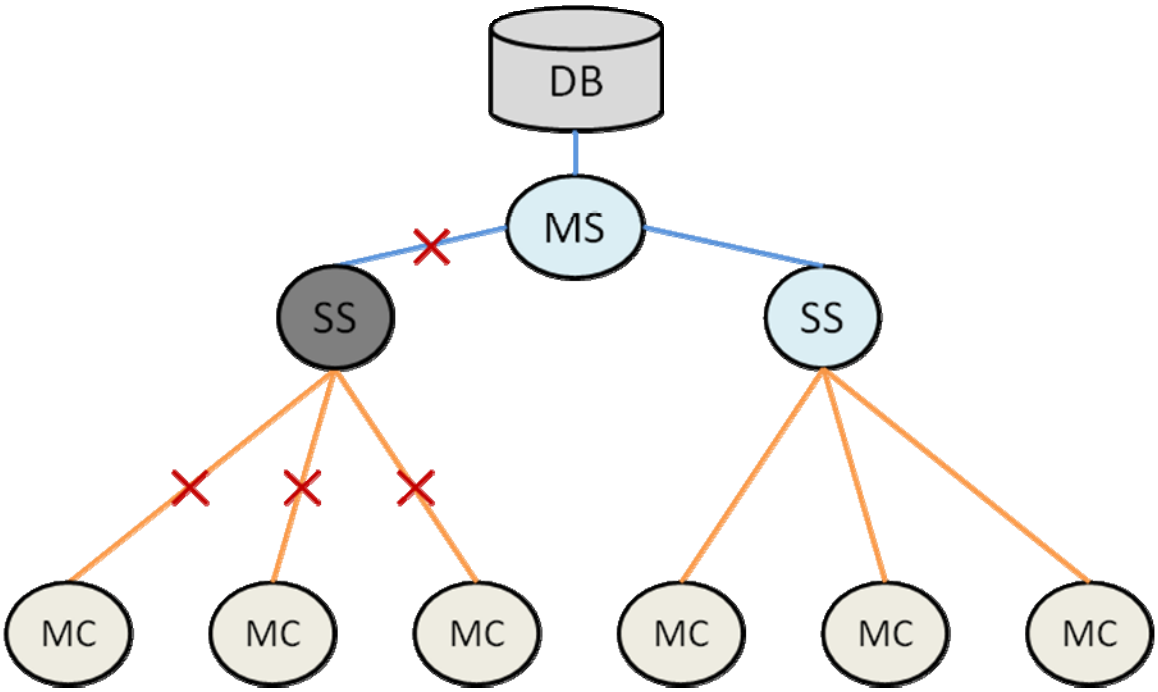

圖 3-6　多 Super Seeder 提高可靠度 步驟二

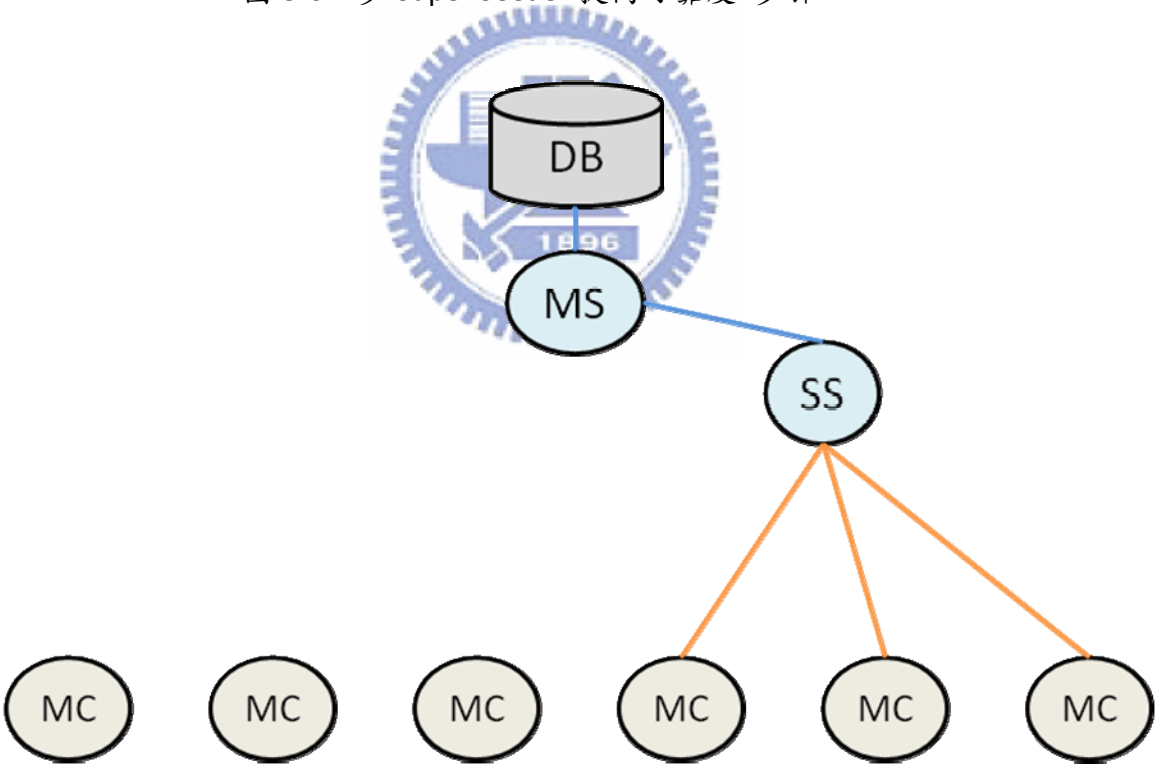

圖 3-7　多 Super Seeder 提高可靠度 步驟三

此時，左方的 Super Seeder 發生網路中斷問題，圖中有紅叉的連線隨即中斷，如圖 3-6、3-7 所示。

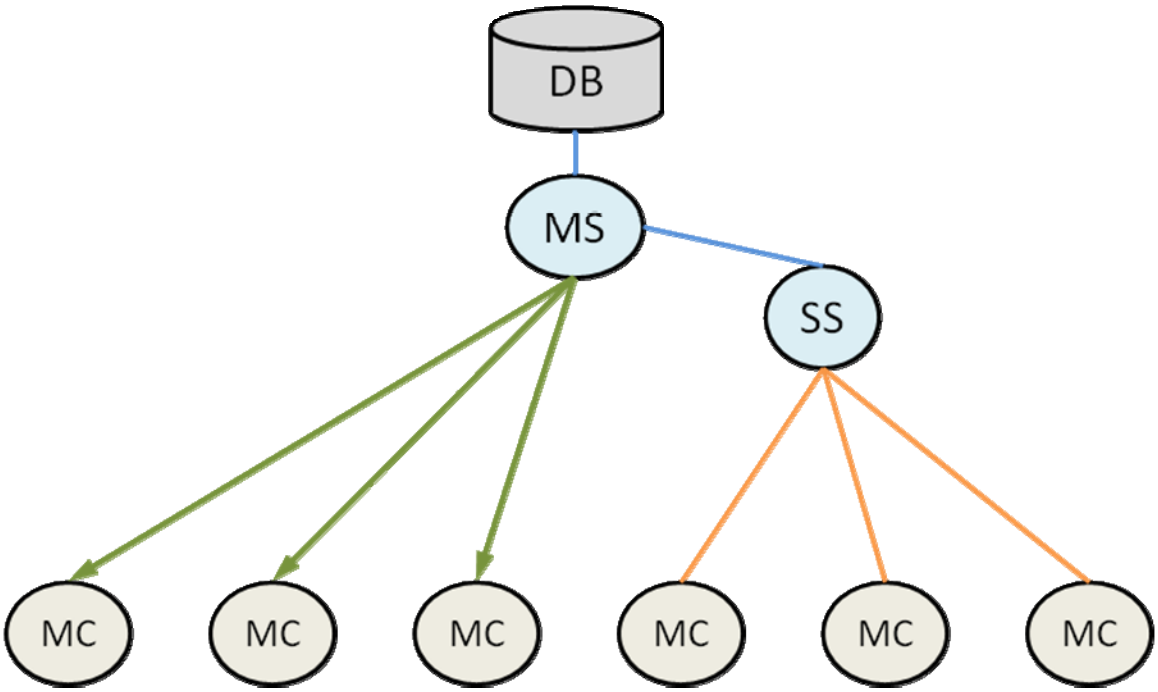

圖 3-8　多 Super Seeder 提高可靠度 步驟四

如圖 3-8，此時，MDS Server 會將左方 Super Seeder 從自己維護的 Super Seeder 清單中去除，並告知原先使用左方 Super Seeder 的三台 MDS Client，可以使用右方的 Super Seeder。

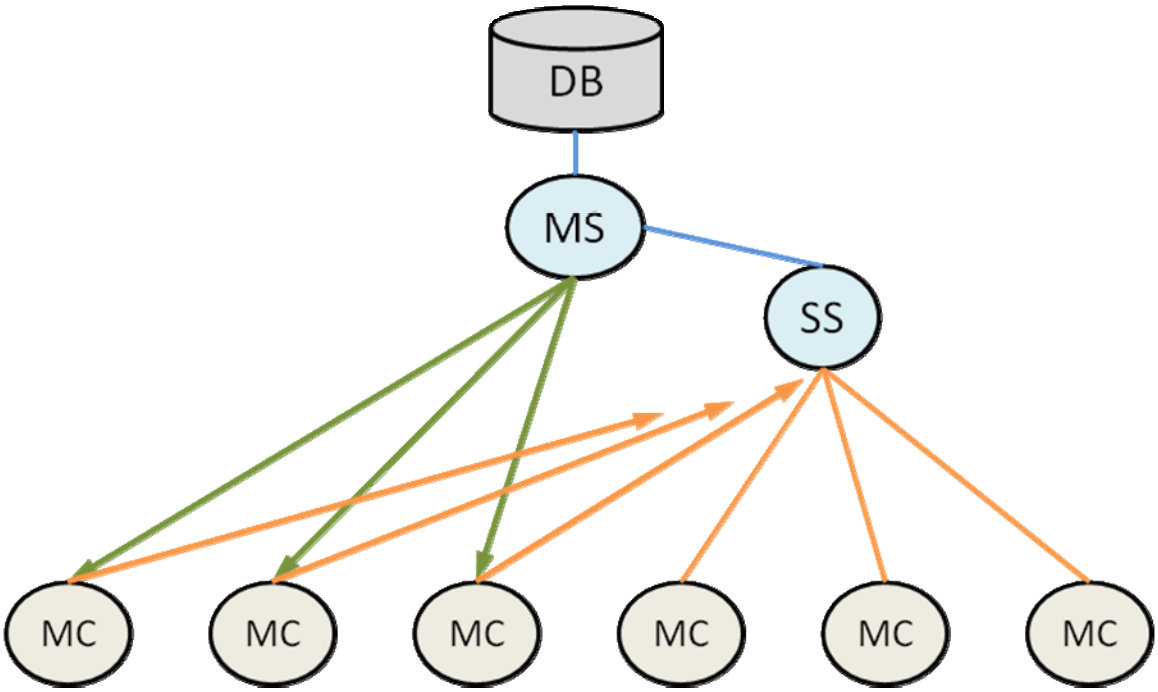

圖 3-9　多 Super Seeder 提高可靠度 步驟五

如圖 3-9，此時，左方三台 MDS Client 連線至右方正常的 Super Seeder，繼續進行 File 片段的傳送。

此一流程顯示多個 Super Seeder 在其中之一停止運作時，也能夠提供可靠服務。

3.4.3 提高可用度 (Availability)

在 MDS Server 以及 Super Seeder 皆有多台的情況下，MDS 能夠具備較高的可用度。

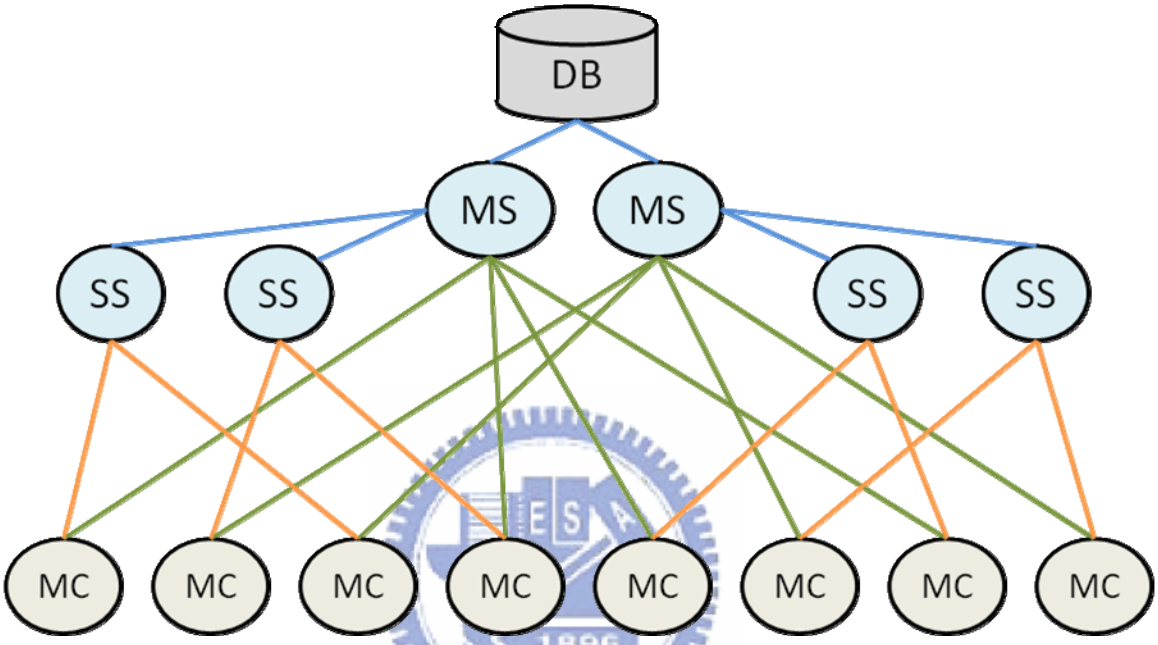

圖 3-10　多 MDS Server 與 Super Seeder 提高可用度 步驟一

如圖 3-10 所示，系統含有兩台 MDS Server，分別控管兩台 Super Seeder，以及四台 MDS Client。在前述負載平衡機制下，左右四台 Super Seeder 會平均服務下面的八台 MDS Client。

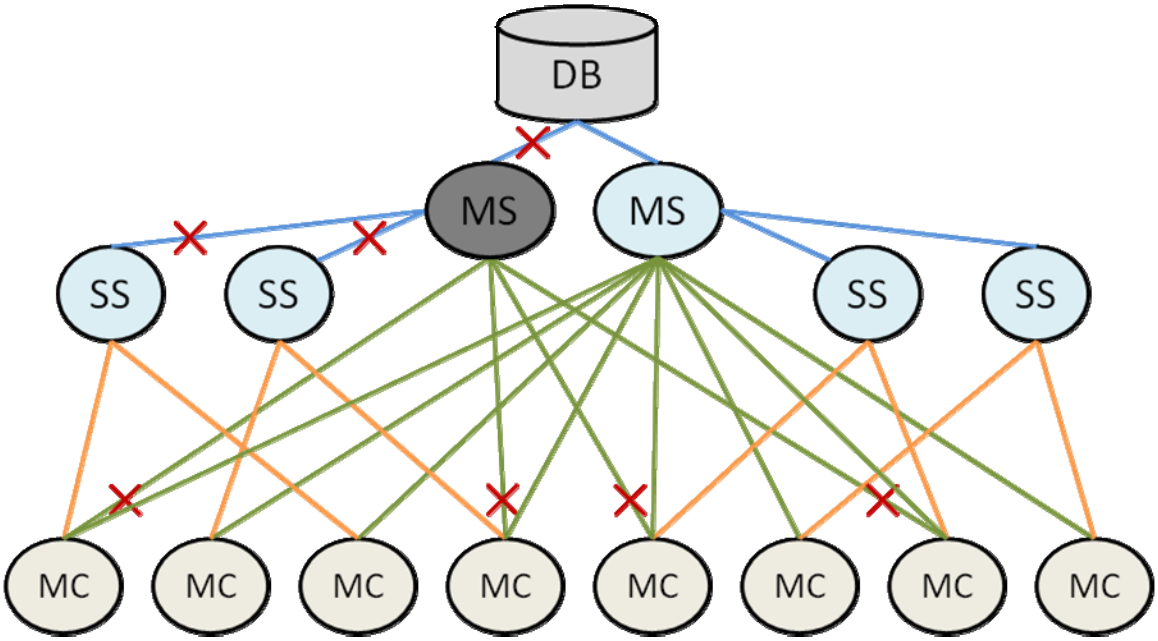

圖 3-11　多 MDS Server 與 Super Seeder 提高可用度 步驟二

此時，左方的 MDS Server 發生網路中斷問題，圖中有紅叉的連線隨即中斷，如圖 3-11 所示。

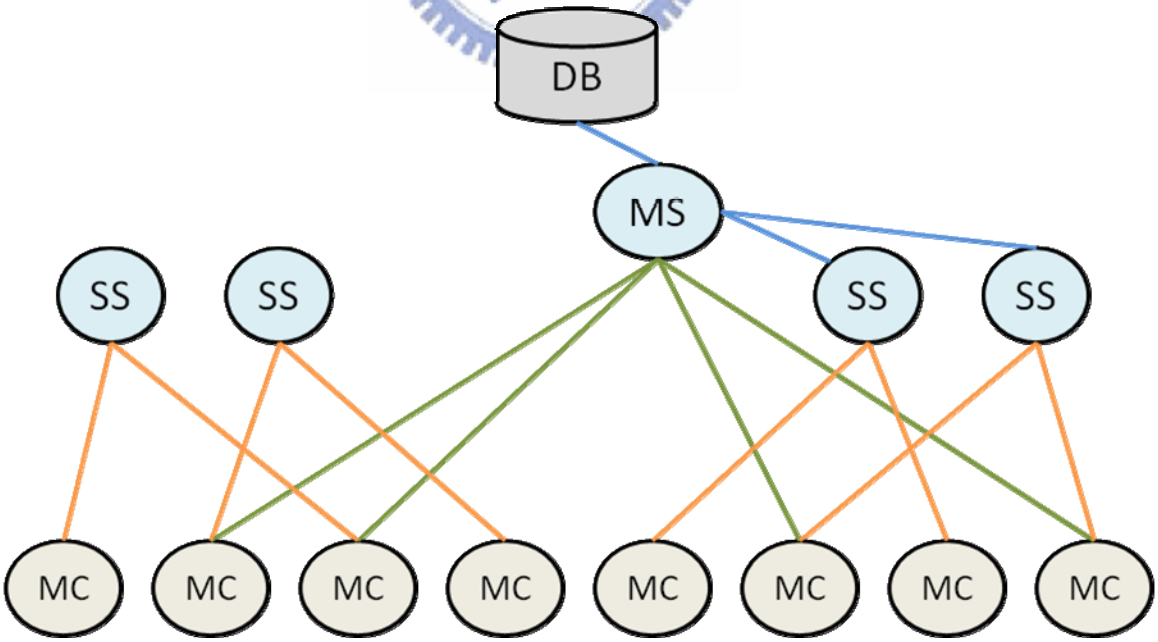

圖 3-12　多 MDS Server 與 Super Seeder 提高可用度 步驟三

在圖 3-12 中可以發現，左方兩台 Super Seeder 雖然已與 MDS Server 斷線，但仍可繼續服務與其連線的 MDS Client，並不會因失去控管而停止服務。

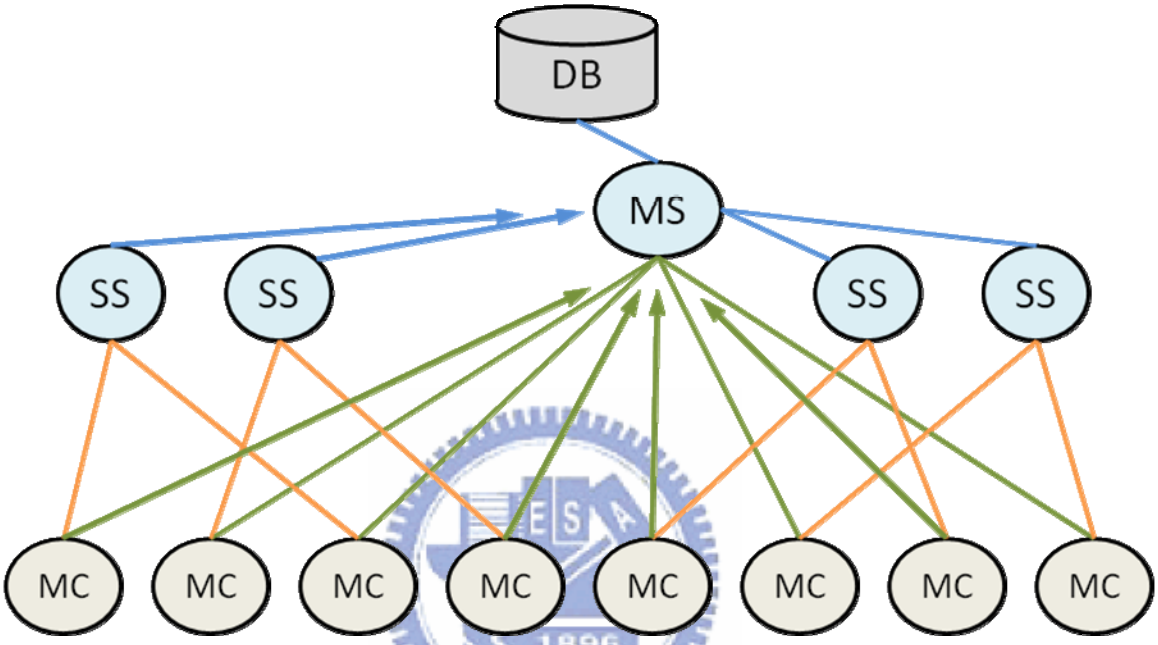

圖 3-13　多 MDS Server 與 Super Seeder 提高可用度 步驟四

如圖 3-13，與 MDS Server 斷線後，Super Seeder 與 MDS Client 會試圖連線原來的 MDS Server，失敗數次後，便會從自身所含的 MDS Server 清單中，挑選一台來進行連線。如圖所示，失去控管的兩台 Super Seeder 與四台 MDS Client，皆連線到右方仍正常運作的 MDS Server，繼續接受控管。

此流程顯示，多個 MDS Server 在其中之一停止運作時，其先前所控管的 Super Seeder 以及 MDS Client 仍可繼續運作，並持續尋找 MDS Server，直到尋找到可用的為止。

若能架設多台 MDS Server 與 Super Seeder，便可提高整個系統的可用度。只要還有一台 MDS Server 與 Super Seeder 存活，便可讓系統保持運作。

第四章、 系統實作

本章會說明系統在實作上的所有細節，包含程式架構以及實作內容。

4.1 程式架構

在程式架構上，考量到 MDS Server、Super Seeder 皆會承載大量的連線與封包處理，因此需要一個效能較高的 I/O 架構。

4.1.1 選擇I/O策略

傳統上，處理同時連線 (Concurrent connections) 的網路程式皆會使用 select() 系統呼叫[8]，達到快速處理多個 socket 的能力。然而在大部分的系統中，為了不讓 select() 超過其能負載的範圍，皆定義了 FD_SETSIZE 這個巨集常數來限制其能力，以 FreeBSD 與 Linux 來說，其值為 1024。也就是說，select() 最多只能夠同時處理 1024 個 socket，這對我們的系統來說，是絕對不夠的。

而傳統的 poll() 系統呼叫[9]，雖然沒有限制最大能處理的 socket 數目，但實際上當數量達到一兩千時，速度便相當慢了，原因在於掃瞄每個 socket 需要花費大量時間。

因此，MDS 需要能夠快速處理超大量 socket，且反應速度夠快的系統呼叫。

在著名的 The C10K Problem[10] 網頁中，提出了種種的 I/O 處理策略，我們選擇了 Event-Driven 的方式，並使用 Non-blocking I/O (非阻塞式 I/O) 以及 Readiness Change Notification (就緒狀態改變通知) 的高效能系統呼叫：FreeBSD 的 kqueue()[11] 以及 Linux 的 epoll()[12]。

而為了讓程式能夠跨平台，我們使用了 libevent 這套 Open Source 開放原始碼函式庫[13]。libevent 在編譯函式庫時，能針對編譯的系統環境，選擇效能最佳的系統呼叫，如 FreeBSD 的 kqueue()、Linux 的 epoll()、Solaris 的 /dev/poll[14]與 event ports[15]，以及傳統的 select() 與 poll()。因此，我們的程式可以在不修改程式碼的情況下，輕鬆地在多個平台上編譯、執行。

此外，libevent 作者採用 BSD License (3-clause BSD License) 授權協議[16]，若將 MDS 運用於企業內，便不易產生侵犯授權爭議。這也是使用 libevent 的優點之一。

4.1.2 Event架構

使用 libevent[13] 函式庫來架構程式後，程式在處理網路 I/O 的部分可以相當簡化。

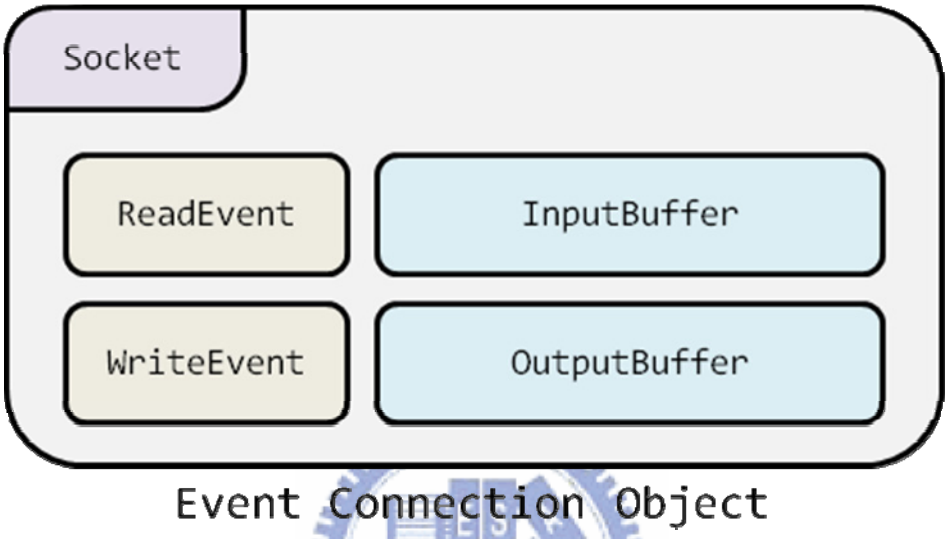

圖 4-1 Event Connect Object 連線物件

如圖 4-1 所示，Event Connection Object 連線物件為每個客戶端連線被接受後，所建立的物件，其內含：

- 一個 Socket fd (Socket 描述子)
- 一組 Read / Write Event (讀取 / 寫入 事件結構)
- 一組 Input / Output Buffer (輸入 / 輸出 緩衝區)

當特定事件發生時，設定好的 Callback 函式將會被呼叫。

以下說明當 Read Event (讀取事件) 以及 Write Event (寫入事件) 發生時的程式行為。

4.1.2.1 Read Event讀取事件發生

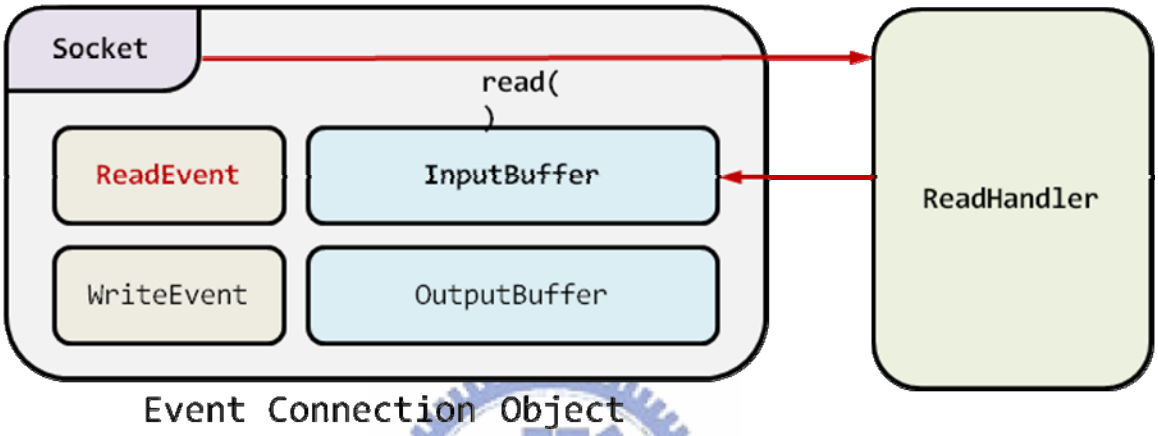

圖 4-2 Read Event 讀取事件發生

如圖 4-2，當系統偵測到此連線物件的 Socket 目前可以讀取，Read Event 讀取事件將會發生，並呼叫設定好的 Callback 函式 ReadHandler。ReadHandler 會從 Socket 讀取資料，並附加到 InputBuffer 輸入緩衝區的尾端。

4.1.2.2 Write Event寫入事件發生

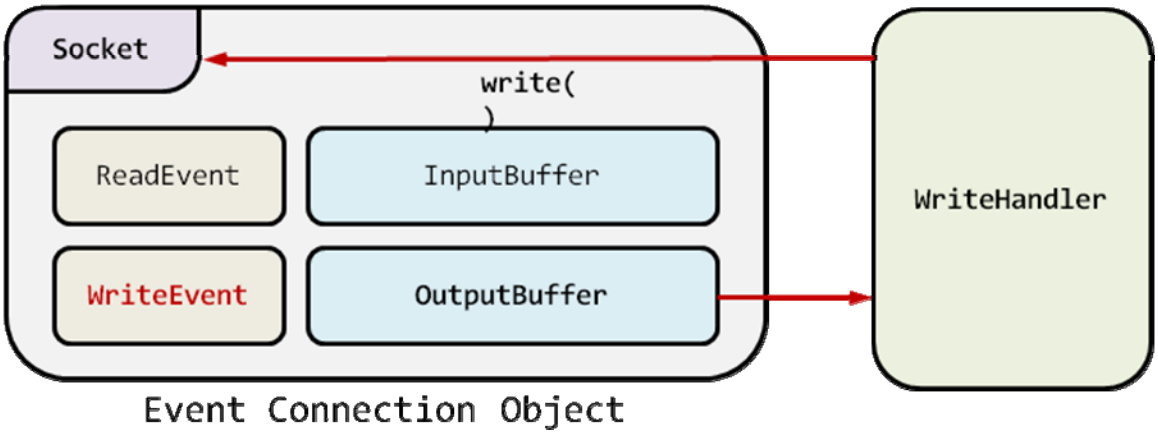

圖 4-3　Write Event 寫入事件發生

如圖 4-3，而當系統偵測到此連線物件的 Socket 目前可以寫入，Write Event 寫入事件將會發生，並呼叫設定好的 Callback 函式 WriteHandler。WriteHandler 會從 OutputBuffer 寫入緩衝區的開頭讀出資料，並將其寫入 Socket。

4.2 實作內容

MDS Server 與 Super Seeder 以 C++ 實作而成，為 single-threaded 單一執行緒程式，並以 single-process 單一程序服務所有連線。

由於 MDS Server 與 Super Seeder 皆為需處理大量同時連線的程式，因此兩者共用相同、使用 libevent[13] 撰寫的程式核心。

目前 MDS Server 與 Super Seeder 已在下列系統中測試：

- FreeBSD 6.3-Release-p3 (i386 32bit)
- FreeBSD 7.0-Release (amd64 64bit)
- CentOS Linux 2.6.18 Release 5.2 (i386 32bit)
- Debian Linux 2.6.18 Release 4.0 (i386 32bit)

因程式在 UNIX 環境執行運作，所以兩者皆設計以 daemon process 在系統背景執行，透過指令稿與額外的控制程式，發送 POSIX Signal 來控制 MDS Server 與 Super Seeder 進行啟動、停止、重新啟動、回報狀態等動作。並使用 syslog 服務記錄程式輸出的訊息，使其和 UNIX 系統整合，提高使用便利性。

以下說明針對 MDS Server 與 Super Seeder 個別的實作問題與解決方法。

4.2.1 MDS Server

MDS Server 在系統中扮演中樞控制的角色，提供 Super Seeder、MDS Client 登入，傳送所需資訊，並維護其清單等功能。

以下介紹與系統穩定度相關的 Super Seeder 與 MDS Client 狀態維護，清除已發生問題之 Super Seeder 與 MDS Client 所設計的 KeepAlive 機制，以及與系統效能相關的 MDS Client NAT 穿透問題。

4.2.1.1 Super Seeder與MDS Client狀態維護

Super Seeder 除了在登入時會傳送其服務資訊給 MDS Server 外，在運行的過程中，也會定時傳送 Report 資訊給 MDS Server，內容為負載 (loading)、每秒處理幾個片段要求 (requests/sec) 以及頻寬使用率 (bandwidth usage)。MDS Server 會依照目前所有 Super Seeder 的負載狀況，來決定要指派哪一個 Super Seeder 給 MDS Client。

而 MDS Client 在登入時會傳送 NAT 與頻寬測試資訊以及目前狀態。其狀態分為兩種：

- 已完成所有 File 片段 (Seeder)

- 未完成所有 File 片段 (Leecher)

當 MDS Client 開始或繼續下載一個新發佈的 File 時，狀態為 Leecher；而當 MDS Client 完成所有 File 片段時，會傳送已完成資訊給 MDS Server，讓 MDS Server 更新此 MDS Client 的狀態。

MDS Client 狀態不同，在挑選、組合 MDS Client List 時的策略也有所不同。例如：當某個 MDS Client 已經成為 Seeder 後，MDS Server 就不需要再給它內含 Seeder 的 MDS Client List，因為它已經不需要其他 MDS Client 傳送 File 片段給它。MDS Server 這時會給他全部都是 Leecher 的 MDS Client List，讓它盡全力去傳送 File 片段；而在 MDS Client 仍是 Leecher 時，MDS Server 會給它一部份 Seeder、一部份 Leecher 的 MDS Client List。除了讓它能和 Seeder 們要求 File 片段外，也能和其他 Leecher 交換彼此沒有的 File 片段。

4.2.1.2 KeepAlive機制

在經過一段時間的運作後，總是會發生一些例外狀況，如 Super Seeder 所在的伺服器當機，或者某 MDS Client 所在的使用者電腦無預警斷電，甚至是某一家 ISP 業者發生網路故障等。這些會造成不正常斷線、甚至不知道有沒有斷線的狀況，導致 MDS Server 沒有偵測到 Super Seeder、MDS Client 的離開，而繼續維護其狀態。

這些不正確的狀態資訊，會讓其他 MDS Client 取得不可用的 Super Seeder 與 MDS Client List，因此要設法讓 MDS Server 知道每個 Super Seeder、MDS Client 是否真正存活。

在這邊，我們讓 MDS Server 記錄 Super Seeder 與 MDS Client 最後傳送資訊的時間戳記 (timestamp)，並定時檢查目前時間與該時間戳記的差值。超過預定的時間 (如 600 秒) 時，發送 KeepAlive 封包，並期待回傳 KeepAlive Acknowledge 封包。若一段時間過後仍未收到 KeepAlive Acknowledge，則判定對方已發生問題，MDS Server 將主動切斷連線，並釋放其所使用資源。

不使用 TCP Socket Option 中的 SO_KEEPALIVE[17] 選項，是因為此機制需要 2 小時內皆沒有任何傳輸，才會發出探測封包。兩小時是相當久的時間，若同時有許多 MDS Client 發生問題，對系統的效能會有相當大的影響。雖然有方法能夠更改此時間，但各種平台上上的支援程度並不相同，因此決定自行實作。

4.2.1.3 處理 MDS Client NAT穿透

由於 MDS Client 散佈在 Internet 網際網路上，加上目前寬頻網路的盛行，MDS Client 位於 NAT[18]後方的機率相當高。

MDS Server 需要處理兩種 MDS Client 的 NAT 問題：

1. 位於 NAT 後的 MDS Client 如何與其他 MDS Client 連線？
2. 位於同一個 NAT 後的 MDS Client，如何直接互相連線？

依照 3.3 MDS 檔案傳輸流程一節中，MDS Client 啟動時進行的 NAT 類型測試[19]，可以得到四種測試結果，如表 4-1 所示。

NAT 測試類型	測試結果說明
Public IP	MDS Client 沒有在 NAT 後方，使用 Public IP
Cone NAT	MDS Client 位在 Full / Restricted / Port Restricted NAT 後方，使用 Private IP
Limit NAT	MDS Client 位在 Symmetric NAT 後方，使用 Private IP
Unknown	測試發生問題或失敗

表 4-1　NAT 類型測試結果

對於兩個 MDS Client 皆在 Cone NAT 後的狀況，可以使用 UDP Hole Punching[20] 技術來進行穿透；而位在 Symmetric NAT 後的兩個 MDS Client，則無法以此方法來進行穿透，需使用 TURN (Traversal Using Relay NAT)[21] 技術，在外部架設 Relay Server 來轉送封包資料，來達到穿透的效果。

從上文可得知，目前並沒有可同時穿透 Cone / Symmetric NAT 的解決方案。UDP Hole Punching 需要使用 UDP 協定，而 TURN 則需要額外的 Relay Server 與頻寬，兩者皆不適用於我們的系統。

在作過實際的 NAT 測試後，我們發現目前使用 Public IP 以及位在 NAT 後 (無論 Cone 或 Symmetric NAT) 的比例約各佔 50%[22]。因此我們選擇使用 Reverse Punching 作為 NAT 穿透策略，讓位在 NAT 後方的 MDS Client 主動連線至使用 Public IP 的 MDS Client。

為了達成 Reverse Punching，MDS Server 使用每個 MDS Client 在登入時所傳送的 NAT 類型測試結果，將 MDS Client 分類，並組合出適合該 MDS Client 的 MDS Client List，傳送至該 MDS Client。

由此可得，有效處理第一個問題的方式，便是利用正在使用 Public IP 的 MDS Client，加上 MDS Server 傳送給 MDS Client 的特製

MDS Client List，讓所有的 MDS Client 皆可互相連線。

而第二個問題的解決方式，則是透過比對 MDS Client 在連線 MDS Server 時所取得的 Public IP。若有兩個 MDS Client 皆在 NAT 後方，且 Public IP 相同，則判定這兩個 MDS Client 位在同一個 NAT 後方。而 MDS Server 傳送給雙方的 MDS Client List 中，對方的 IP 即為內部的 Private IP，因此雙方可在 NAT 內直接連線，不需繞過 NAT。

不過此一方式仍有缺點，便是只能應付單層架構的 NAT，若遇上多層架構的 NAT，或內部切割不可互連的子網路時，便會連線失敗。因此在第二個問題的解決上，還有改善空間。

整理 MDS Client List 選擇策略如表 4-2。

	使用 Public IP	位於 NAT 後
已為 Seeder	使用 Public IP 且為 Leecher	(使用 Public IP 或 在同 NAT 後) 且為 Leecher
仍為 Leecher	使用 Public IP Seeder/Leecher 皆可	(使用 Public IP 或 在同 NAT 後) Seeder/Leecher 皆可

表 4-2　MDS Client List 選擇策略

4.2.2 Super Seeder

Super Seeder 為系統中最為忙碌的元件，需要不停地處理來自 MDS Client 的 File 片段請求，並回傳該片段給 MDS Client。Super Seeder 與 MDS Client 之間的行為參考自 BitTorrent 協定[6]，含有一系列的狀態與資料封包格式。

而為了保持 Super Seeder 的穩定，以及達成 3.4.1 一節所述的負載平衡，除需要定時回報狀態給 MDS Server 外，也需要控制其輸出的頻寬。

以下將說明 Super Seeder 如何與 MDS Client 通訊、並靈活地回報狀態給 MDS Server，以及如何控制輸出頻寬。

4.2.2.1 與MDS Client通訊

BitTorrent 協定主要用於交握、確認已有 File 片段、以及要求 File 片段、傳遞 File 片段。各封包有其順序，依序如下：

1. **Handshake：**

   Super Seeder 與 MDS Client 之間傳送的第一個封包，內含軟體版本資訊及目前所使用 File Signature 的一組雜湊值 (hash)。我們將

原本 BitTorrent 的 Handshake 擴充，額外加入發送端的 NAT 類型與頻寬，未來可利用此資訊進行更細微的最佳化。此封包由 MDS Client 主動送出，Super Seeder 收到後回傳自己的 Handshake 封包，完成交握。

2. **BitField：**

BitField 內記載發送端所擁有的 File 片段資訊，一個 bit 表示一個片段。Super Seeder 在完成交握後，將傳送每個 bit 都是 1 的 BitField 封包給 MDS Client，告知對方自己擁有全部 File 片段。

3. **Interested：**

此封包由 MDS Client 發出，表示想要從 Super Seeder 下載 File 片段。

4. **UnChoke / Choke：**

Super Seeder 收到來自 MDS Client 發出的 Interested 後，可選擇回覆 UnChoke 或 Choke。UnChoke 表示同意 MDS Client 開始請求 File 片段；而 Choke 表示禁止 MDS Client 請求 File 片段。此機制是 BitTorrent 協定中用來控制傳輸的，有一套 Choking Policy 來控制。然而 Super Seeder 在目前的設計中，是公平傳送 File 片段給每個發出要求的 MDS Client 的，因此當 Super Seeder 收到 Interested

封包時，一律回覆 UnChoke，讓 MDS Client 能夠來請求 File 片段。

5. Request：

MDS Client 發出的 File 片段請求，內含片段編號 (piece number)、開始位置 (offset) 以及長度 (length)。雖然將 File 分成若干片段來傳送，但實際上每次請求的長度，才是真正在網路上傳送的大小。在我們的系統中，長度均為 16384 Byte (16 KB)。

6. Piece：

Super Seeder 傳送給 MDS Client 的 File 片段，內含片段編號 (piece number)、開始位置 (offset) 以及長度 (length)，最後接上 File 片段資料。MDS Client 可從封包內容得知自己已經收到了哪一個片段的部分內容。

每一個 MDS Client 與 Super Seeder 皆需經過上述封包流程，在開始發送 Request 片段請求後，便不停地重複此步驟，直到完成所有 File 片段。在完成所有 File 片段、成為 Seeder 之後，便會主動與 Super Seeder 斷線。

4.2.2.2 狀態回報

為了精確監控 Super Seeder 的運作狀況，每一秒鐘便會執行一個用來統計資訊的函式，而每執行五次此函式，便會將這五次所統計資料的平均值，傳送給 MDS Server。內容如 4.2.1.1 節所述，含有負載 (loading)、每秒處理幾個片段要求 (requests/sec) 以及頻寬使用率 (bandwidth usage)。

若 Super Seeder 目前並不忙碌，每五秒傳送一次 Report，便顯得有點浪費了。因此我們設計了一個閒置計數器，此計數器數值越大，表示 Super Seeder 越不忙碌。藉由此計數器，可以讓 Super Seeder 在忙碌的時候，每五秒便傳送一次 Report 給 MDS Server，以即時監控狀態；而在漸漸不忙碌的時候，拉長傳送 Report 的時間間隔；直到完全不忙碌的時候，傳送幾次 Report，讓 MDS Server 維護的 Super Seeder 狀態符合現況後，便停止傳送 Report。

4.2.2.3 控制輸出頻寬

MDS 系統的目的之一，便是讓使用此系統的公司、企業，能夠降低頻寬上的花費。倘若 Super Seeder 不限制輸出，讓系統持續佔用頻寬直到滿載，便失去使用 MDS 的意義。因此，讓 Super Seeder

能夠有效地控制輸出頻寬，是相當重要的。

回顧 4.1.2.2 節中提到的 Write Event 寫入事件發生：

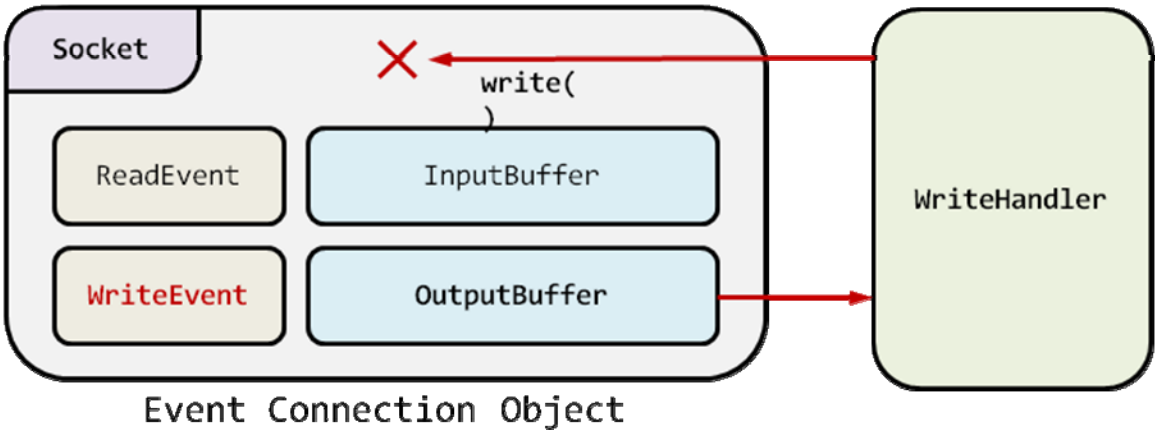

圖 4-4　Write Event 控制輸出頻寬

如圖 4-4，只要在寫入事件發生時，讓 WriteHandler 不要將資料從 OutputBuffer 寫入 Socket，便可達到控制輸出頻寬的目的。而未輸出的資料，則會留在 OutputBuffer 中，直到下寫入事件發生，才有機會被寫入 Socket，輸出至遠端。

由於傳送給 MDS Client 的 File 片段為主要輸出項目，且長度也比通訊用封包長了許多倍，為了不影響其他通訊，輸出控制只針對傳送 File 片段所用的 Piece 封包。

Super Seeder 控制輸出頻寬的步驟流程如下：

1. 設置一輸出計數器，初始值為零。

2. 每當送出一個 Piece 封包時，便累加其長度於輸出計數器。
3. 當寫入事件發生時，檢查輸出計數器，若小於預先設定的頻寬限制大小，便寫入 Socket；若大於，則不寫入 Socket，並將此 Event Connection Object 連線物件的 Write Event 暫停，使其不會再發生，並將其儲存至一清單。
4. 每一秒將輸出計數器列印到記錄檔，並將輸出計數器歸零，然後將步驟三中，儲存在已暫停連線物件清單內，每一個連線物件的 Write Event 啟動。由於輸出計數器已歸零，且之前被暫停的 Write Event 也已被啟動，下次發生寫入事件時，若輸出計數器小於頻寬限制，便可寫入 Socket。

由於 MDS Client 會偵測 Super Seeder 回傳 Piece 封包的速度，來調整發出 Request 片段請求的頻率，因此不必擔心連線物件內的 Input / OutputBuffer 內會囤積太多未處理的資料，而影響 Super Seeder 的效能與資源使用。

第五章、 實驗結果

本章說明我們的實驗設計、結果以及分析。

5.1 實驗設計

在實驗的設計上，我們模擬一家使用 MDS 的線上遊戲公司，正要釋出一套遊戲的重大更新，並模擬一般家用 ADSL 網路環境。為了加快實驗速度，網路傳輸速度均為真實環境的 5 倍，且在一不受外界干擾的 Gigabit 內部網路進行實驗。

實驗設定如表 5-1。

設定項目	設定說明
MDS 架構	單一 MDS Server、單一 Super Seeder 架構
傳輸 File 大小	50 MB (200 片段)
Super Seeder 網路環境	上傳 2000 KB/sec
MDS Client 網路環境	上傳 120 KB/sec、下載 600 KB/sec 使用 Public IP
MDS Client 數量	50 台，所有 MDS Client 設定相同

表 5-1 實驗設定

而為了做出比較，我們設計了三組實驗，如表 5-2 所示：

	實驗說明
實驗一	模擬傳統 Client / Server 架構
實驗二	MDS P2P 架構，所有 MDS Client 互相連線
實驗三	MDS P2P 架構，部分 MDS Client 互相連線

表 5-2　實驗內容說明

在實驗架構圖中，由於 MDS Server 不在操作變因中，因此沒有顯示。而 Super Seeder 與 MDS Client 同 3.4.1 一節，使用縮寫。SS 代表 Super Seeder，MC 則代表 MDS Client。

5.2　實驗結果與分析

5.2.1　實驗一、模擬傳統Client / Server架構

實驗架構如圖 5-1 所示：

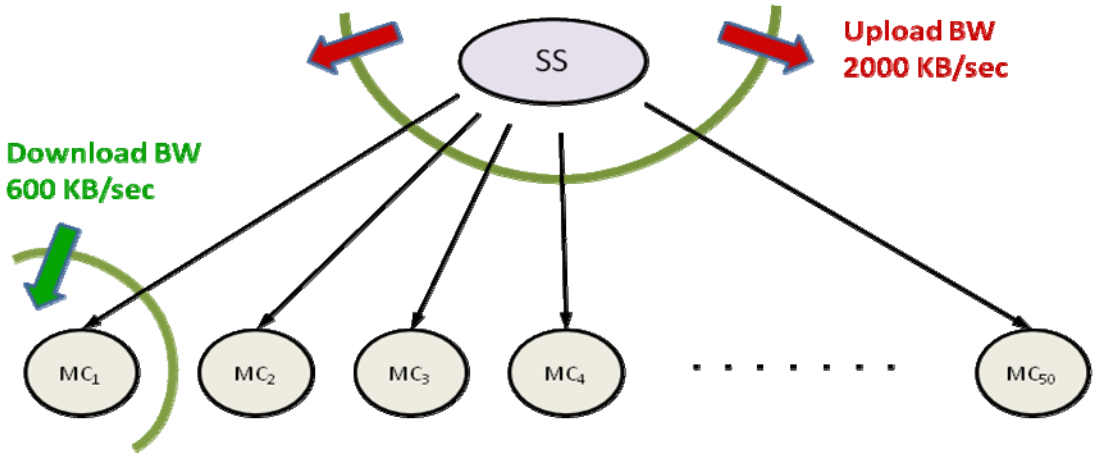

圖 5-1　實驗一 模擬傳統 Client / Server 架構

模擬傳統 Client / Server 架構的實驗，主要用來和 MDS P2P 架構作一本質上的比較。實驗結果如表 5-3 所示：

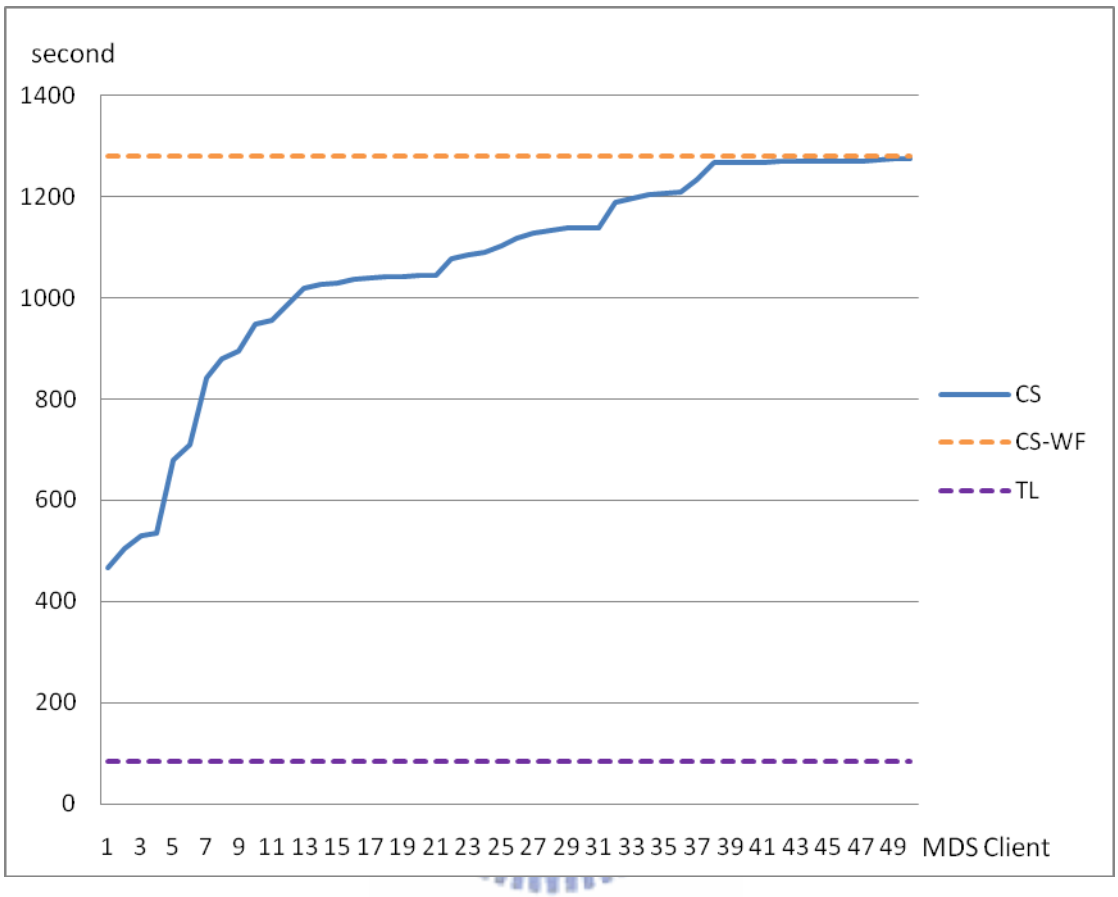

表 5-3　實驗一結果 模擬傳統 Client / Server 架構

表 5-3 中橫軸為 MDS Client 完成 File 的順序，左至右為最快至最慢；縱軸則為 MDS Client 完成 File 所花費的秒數。由於橫軸為快至慢的順序，因此曲線的走向必為左低右高。

下方標示 TL 的直虛線，為單一 MDS Client 最快完成時間，其計算方式為：

$$\frac{50\,MB}{600\,KB/sec} = 85.3$$

此虛線表示，無論系統效能再好，也無法超越單一 MDS Client 能夠達到的最快完成時間。

標示 CS 的實線，即為傳統 Client / Server 架構的實驗結果，從最快的 466 秒到最慢的 1274 秒。而標示 CS-WF 的直虛線為傳統 Client / Server 架構的理論整體最快完成時間，計算方式為：

$$\frac{50\,MB \times 50}{2000\,KB/sec} = 1280$$

CS 實線的末端與 CS-WF 直虛線的末端交錯，表示此次實驗符合理論計算。

5.2.2 實驗二、MDS P2P架構，所有MDS Client互相連線

實驗架構如圖 5-2 所示：

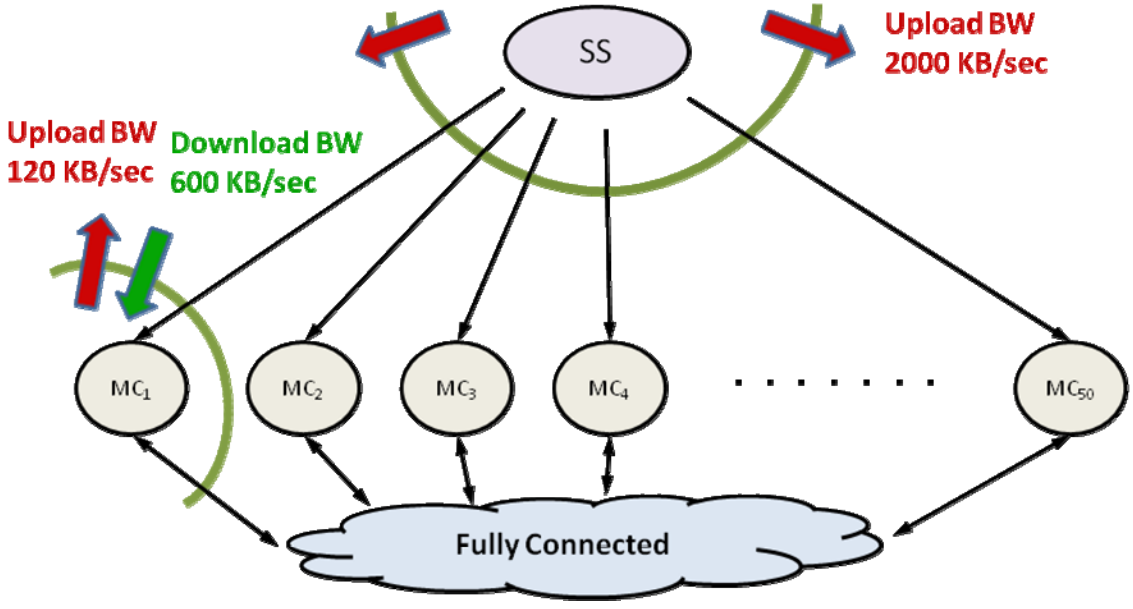

圖 5-2 實驗二 MDS P2P 架構，所有 MDS Client 互相連線

實驗結果如表 5-4 所示：

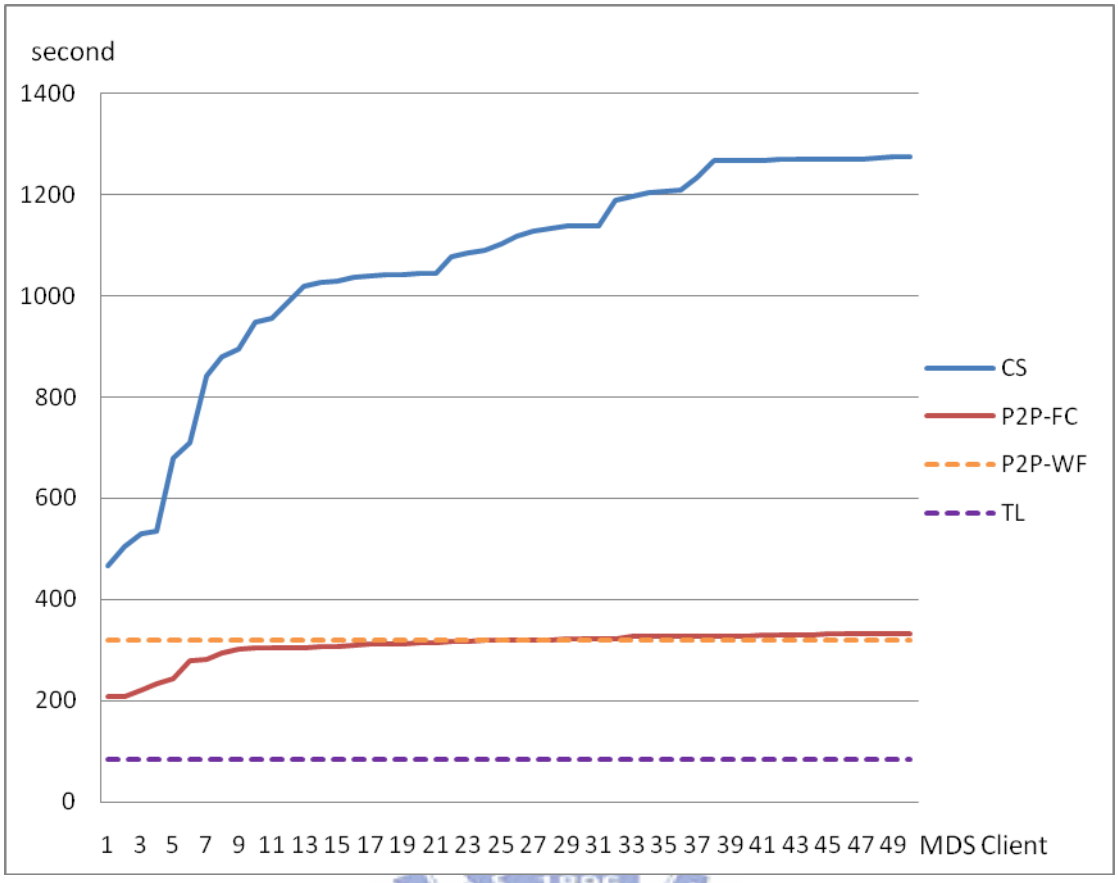

表 5-4　實驗二結果 MDS P2P 架構，所有 MDS Client 互相連線

表中標示 P2P-FC 的實線，為 MDS P2P 架構，所有 MDS Client 互相連線的實驗結果，從最快的 208 秒到最慢的 333 秒。與前次實驗的 CS 實線相比，可輕易發現整體效能較傳統 Client / Server 架構高出許多。

標示 P2P-WF 的直虛線為 MDS P2P 架構的理論整體最快完成時間，計算方式為：

$$\frac{50\ MB \times 50}{2000\ KB/sec + 120\ KB/sec \times 50} = 320$$

此次理論值計算方式與實驗一稍有不同，但原理皆為：

$$\frac{\text{總下載}}{\text{總上傳}}$$

總下載量因為 MDS Client 數量不變，所以沒有變化；但總上傳頻寬則因 P2P 架構，需加上各 MDS Client 所貢獻的上傳頻寬，因而有所不同。

此次實驗中，P2P-FC 實線與 P2P-WF 直虛線有交錯，表示實驗符合理論計算，足以採信。

5.2.3 實驗三、MDS P2P架構，部分MDS Client互相連線

實驗架構如圖 5-3 所示：

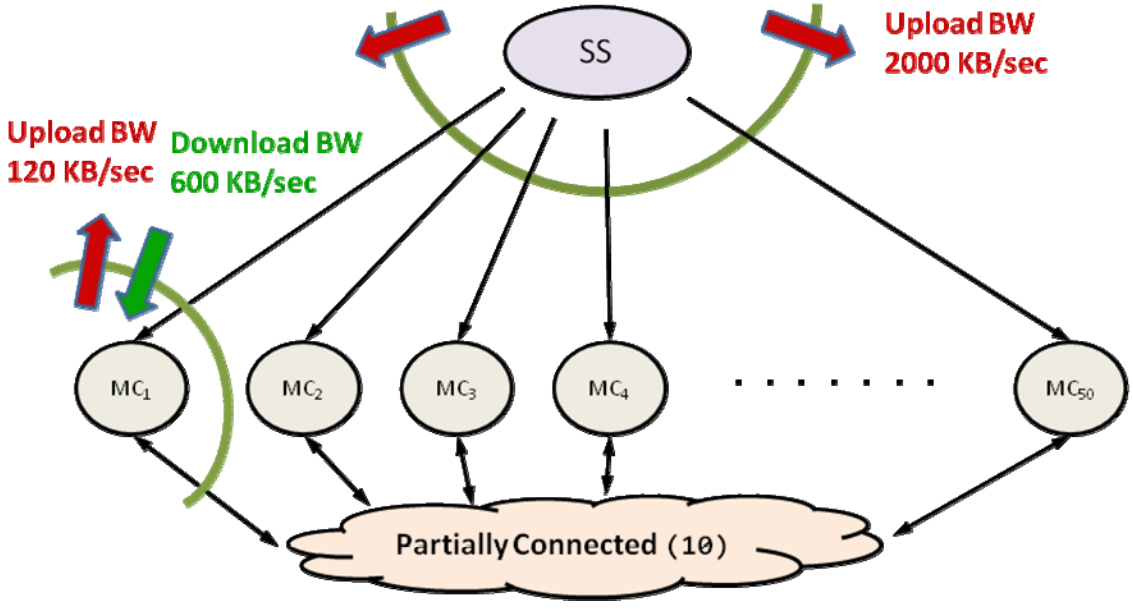

圖 5-3 實驗三 MDS P2P 架構，部分 MDS Client 互相連線

實驗三有別於實驗二，MDS Client 只有部分互相連線，藉由控制 MDS Client List，讓每個 MDS Client 最多只能與其他 10 個互相連線、交換 File 片段。

實驗結果如表 5-5 所示：

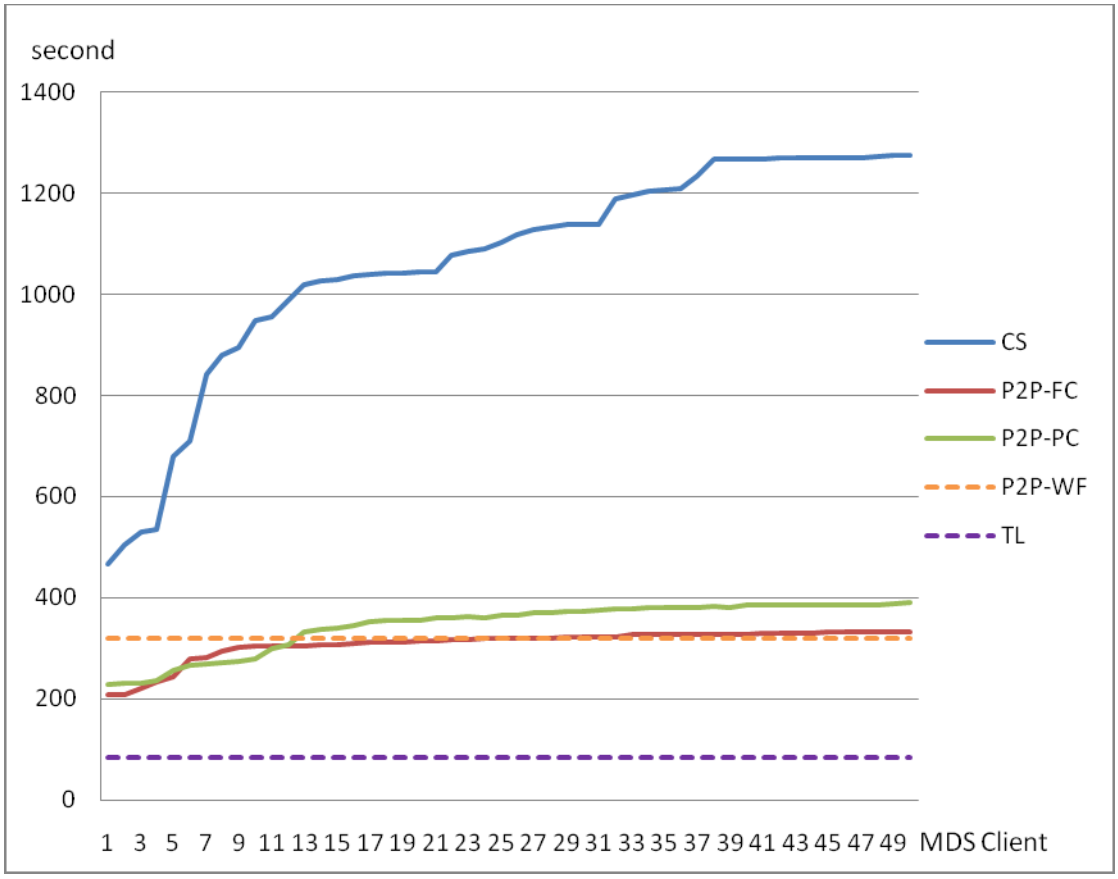

表 5-5　實驗三結果 MDS P2P 架構，部分 MDS Client 互相連線

在表中可以發現，標示 P2P-PC 實線的本次實驗結果，與實驗二標示 P2P-FC 實線的差距並不大。整體上，P2P-PC 比 P2P-FC 速度慢了一些。

為了方便觀察，我們將實驗一的 CS 實線去除，如表 5-6 所示：

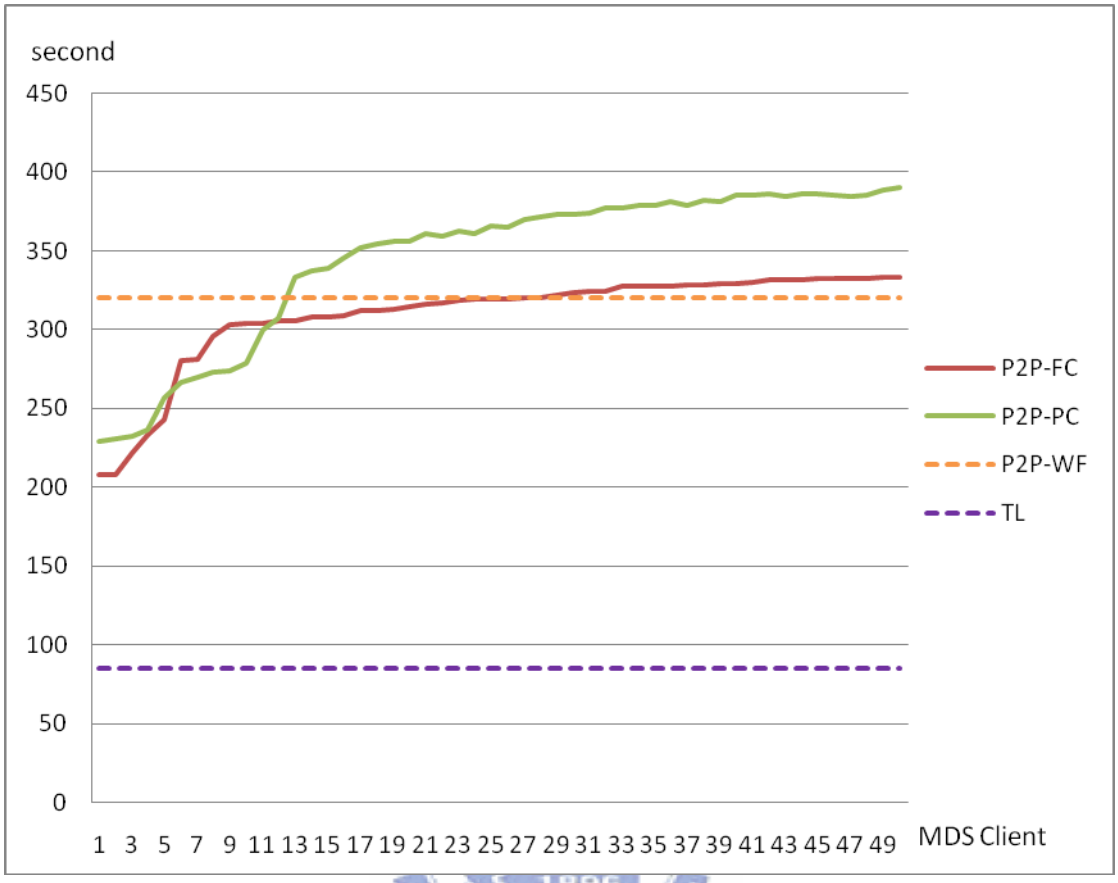

表 5-6　實驗三結果 MDS P2P 架構，部分 MDS Client 互相連線 去掉 CS 數據

在表中可以清楚觀察到，部分 MDS Client 互相連線的結果，從最快的 229 秒到最慢的 390 秒，比所有 MDS Client 互相連線的結果差，但是並不會相差很多。

由於在網際網路環境中，不可能讓所有的 MDS Client 全體互相連線，而本實驗證明，即使只有部分連線，也能夠有不錯的效能。因此，可以預期我們的系統，在真實世界的網際網路中，只讓部分 MDS Client 互相連線，即可發揮 P2P 應有的效能。

因為 MDS Client 數量、上下傳頻寬，以及 Super Seeder 的上傳頻寬皆為改變，所以實驗三的理論整體最快完成時間和實驗二相同，以標示 P2P-WF 的直虛線表示。

而 P2P-PC 實線與 P2P-WF 直虛線有交錯，表示實驗三的結果也符合理論計算，為一可信的實驗結果。

第六章、 結論與未來展望

6.1 結論

本論文實作出一套能夠快速發佈檔案至客戶端的系統，並能夠針對客戶端的網路環境，進行客戶端群體的調整與最佳化。

其中，MDS Server 扮演重要的中樞角色。維護 Super Seeder 與 MDS Client 的各項狀態、資訊，使用 KeepAlive 機制處理發生例外狀況的 Super Seeder 與 MDS Client，以及傳送針對每個 MDS Client 所製作的 MDS Client List，使其能穿透 NAT，互相連線。

而 Super Seeder 使用 BitTorrent 協定，將 File 片段傳送至每一個 MDS Client，讓 MDS Client 之間的片段交換效率提升，且能夠有效控制自己的輸出頻寬，達到降低頻寬花費的效果。

在 MDS 多伺服器架構中，可以同時存在多個 MDS Server，每個 MDS Server 可以控管多個 Super Seeder，以及與其連線的眾多 MDS Client。而 MDS Server 之間透過 Database 共享所有狀態資訊。

藉由共享這些資訊，MDS 多伺服器架構能夠自動負載平衡、增加可靠度，並提升系統可用性。

6.2 未來展望

在實驗過程中，我們發現 MDS 確實能夠發揮 P2P 應有的效能，但在論文 「以 P2P 為基礎之巨量發佈系統 - 客戶端與效能分析研究」 中可以發現，MDS 在效能上仍有改善的空間。

在效能改善上，MDS Server 可以使用尚未充分利用的 MDS Client 頻寬資訊，來最佳化 MDS Client List 的製作；而 Super Seeder 也能夠用此資訊來改變傳送 File 片段的行為，讓擁有不同頻寬能力的 MDS Client，能以不同的優先權與速度下載 File 片段。

而因為所有資訊皆儲存在 Database 中，MDS Server 存取 Database 的頻率相當高，當使用 MDS 多伺服器架構時，Database 將會更忙碌，每個 MDS Server 因等待 Database 回傳結果的時間也會拉長。因此 MDS Server 可以設計一套快取機制，將從 Database 取回的資訊快取，持續使用一段時間，待快取過期 (expire) 後，才再次向 Database 發出請求。

另外，Super Seeder 目前的頻寬控制機制，是設定一個硬性上限 (hard limit)，使輸出頻寬不超過這個上限。然而當 Super Seeder 相當忙碌、無法有效舒緩頻寬需求時，這個硬性上限，會使得系統整體

效能受到限制，許多 MDS Client 將會浪費許多時間來等待 Super Seeder 所回傳的 File 片段。

因此，Super Seeder 應該要能動態頻寬控制，在負載升高、忙碌時，自動提高頻寬至一個限度，而後在系統漸漸不忙碌時，將頻寬限制降低到原先的設定。

最後，雖然 MDS 多伺服器架構，有負載平衡、增加可靠度，提升系統可用度的能力，但距離完全容錯 (fully fault-tolerance) 仍有一段差距。未來應加入伺服器自動備援、從錯誤恢復運作等機制，建構一套更為強健、高可用度的系統。

參考資料

[1] MetaMachine, *eDonkey2000 (eDonkey, ed2k)* http://www.edonkey2000.com , 2000 - 2005.

[2] BitTorrent, Inc., *BitTorrent* http://bittorrent.com , 2002 - 2008. Bram Cohen, *Incentives Build Robustness in BitTorrent*, May 2003.

[3] Shawn Fanning, Napster, Inc., *Napster* http://www.napster.com , 1999 - 2001, 2004 - 2008.

[4] Justin Frankel, Tom Pepper, *Gnutella* http://en.wikipedia.org/wiki/Gnutella , 2000 - 2008.

[5] eMule-Team, *eMule* http://www.emule-project.net , 2002 - 2008.

[6] BitTorrent Specification http://wiki.theory.org/BitTorrentSpecification , February 2008

[7] Chou, Ting-Liang, *The Study of Massive Deployment System Based on P2P Technology - Clients and Performance Analysis*, August, 2008.

[8] Berkeley Software Distribution, 4.2BSD, *select(2)* http://www.freebsd.org/cgi/man.cgi?query=select

[9] AT&T, Inc., System V UNIX, *poll(2)* http://www.freebsd.org/cgi/man.cgi?query=poll

[10] Dan Kegel, *The C10K Problem* http://www.kegel.com/c10k.html, September 2006.

[11] FreeBSD 4.1, *kqueue(2)* http://www.freebsd.org/cgi/man.cgi?query=kqueue

[12] Linux 2.5.44, *epoll(4)* http://www.xmailserver.org/linux-patches/epoll.txt

[13] Niels Provos, libevent API http://monkey.org/~provos/libevent, 2000 - 2008.

[14] Sun Microsystems, Inc., Solaris, */dev/poll* http://access1.sun.com/techarticles/devpoll.html

[15] Sun Microsystems, Inc., Solaris, *event ports* http://developers.sun.com/solaris/articles/event_completion.html

[16] Regents of the University of California, 3-clause BSD License http://www.linfo.org/bsdlicense.html , 1990.

[17] R. Braden, *RFC 1122*, *TCP Keepalive* http://www.faqs.org/rfcs/rfc1122.html , October 1989.

[18] K. Egevang, P. Francis, *RFC 1631, The IP Network Address Translator (NAT)* http://tools.ietf.org/html/rfc1631 , May 1994.

[19] J. Rosenberg, J. Weinberger, C. Huitema, R. Mahy, *RFC 3489, STUN - Simple Traversal of User Datagram Protocol (UDP) Through Network Address Translators (NATs)* http://tools.ietf.org/html/rfc3489 , March 2003.

[20] Bryan Ford, Pyda Srisuresh, Dan Kegel, *Peer-to-Peer Communication Across Network Address Translators* http://www.bford.info/pub/net/p2pnat , February 2005.

[21] J. Rosenberg, R. Mahy, P. Matthews, *IETF Behave Draft, Traversal Using Relays around NAT (TURN): Relay Extensions to Session Traversal Utilities for NAT (STUN)* http://tools.ietf.org/id/draft-ietf-behave-turn-09.txt , July 2008.

[22] *群想科技，客戶端 NAT 與頻寬分析 內部文件*, August 2008.