國立臺灣大學電機資訊學院資訊工程研究所

碩士論文

Graduate Institute of Computer Science and Information Engineering
College of Electrical Engineering and Computer Science
National Taiwan University

Master Thesis

改善 Linux 網路儲存系統上 Samba 伺服器檔案寫入之效能

Performance Optimization of the Samba write service on Linux-based Network-Attached Storage Systems

黃昶竣

Chang-Chun Huang

指導教授：洪士灝 博士

Advisor: Shih-Hao Hung, Ph.D.

中華民國 97 年 7 月

June, 2008

國立臺灣大學電機資訊學院資訊工程學系

碩士論文

Department of Computer Science and Information Engineering

College of Electrical Engineering and Computer Science

National Taiwan University

Master Thesis

改善 Linux 網路儲存系統上 Samba 伺服器檔案寫入之效能

Performance Optimization of the Samba write service on Linux-based Network-Attached Storage Systems

黃昶竣

Chang-Chun Huang

指導教授：洪士灝 博士

Advisor: Shih-Hao Hung, Ph.D.

中華民國 97 年 7 月

July, 2008

誌謝

首先感謝指導教授 洪士灝教授兩年來在研究方面的指導，從決定研究題目、指導研究過程問題的解決、到論文的指導，一路下來不辭辛勞的教導，沒有洪教授耐心的指導，這篇論文是不可能完成，非常感謝其不求回報的付出。於生活上洪教授也提供了我們許多寶貴的經驗，非常感謝。同時也感謝郭大維教授、施吉昇教授、以及陳任凱總經理百忙之中抽空來擔任我的論文口試委員，並針對本論文提出許多寶貴的建議。

感謝實驗室的學長、同學、學弟，這兩年來一起學習、吃飯、及互相鼓勵，陪我度過這充實又快樂且難忘的兩年生活。最後感謝我的家人與朋友對我支持，感謝大家。

中文摘要

隨著網路技術的進步，連網式儲存裝置也變得越來越普遍，特別於 Network-Attached Storage（NAS）更為普遍，隨著 NAS 的普遍，NAS 的價格與效能也越來越受重視，另外隨著網路頻寬的增加，使得較低階的處理器沒辦法負擔網路協定的處理以及資料的搬移。為了解決這些問題，有研究提出 Remote Direct Memory Access（RDMA）及 TCP Offload Engine（TOE）等技術來解決這些問題。然而這些解決方案必須添加額外的硬體支援，還需要軟體的配合才能有效率地減輕 CPU 的負擔。

本論文利用效能追蹤量測工具及技術，找出 NAS 上所需改善的效能瓶頸，並利用追蹤到的資料協助軟硬體整合的工作。我們成功地將 Samba NAS server 處理資料的主要功能遷移到 Linux 作業系統核心（kernel），以減輕對 CPU 在資料搬移的負擔。由於軟體架構的改變，我們得到 1.01~1.31 的寫檔效能改進。未來搭配 TOE 之後，對於大檔案的寫入，預計可達到 2.62 的效能增進。

Abstract

As the Ethernet performance increases, network-enabled storage solution becomes popular, especially for the Network Attached Storage （NAS） appliances. Some low-end NAS servers come with affordable prices and thus, the trade between performance and cost in such low-end NAS servers become an important issue. In fact, while processing the I/O requests from the client machine, the control processor in the low-cost NAS server spends most of the time handling data copy operations and network protocols. To boost the performance, software （zero-copy） and hardware （TCP Offload Engine） solutions are provided to remove data copy operations and handle network protocols, respectively. However, few attempts are made to discuss this hardware-software codesign issue about the software effort made to take advantage from the offload engines.

In this thesis, we tackle the hardware-software codesign issue by presenting a throughout performance study of target system. We first diagnose the performance of target system. The result shows that data copy operations and data processing in network protocol stack are two major performance bottleneck. For the software part, we migrate the Samba engine to the kernel and remove unnecessary data copy operations. For the hardware part, we predict the performance after adding a offload engine to handle data processing in network protocol stack. We show that an 1.01~1.31 speedup is achieved for the software part compared to original code. And, 1.51~2.62 performance improvement is enhanced if the network protocol process can be offload to a dedicated hardware.

目錄

圖目錄

表目錄

第1章 序論

1.1 概序

近年來，隨著網路技術的進步，網路的速度被推進到每秒 10 gigabits 的傳輸速度，使網路應用得以傳送大量資訊，然而資訊的處理也可能成為最主要的效能瓶頸。處理機（CPU）為了處理來自於網路的封包，必須花費計算資源來處理網路協定工作以及將資料在不同的軟硬體單元或是硬體單元之間進行複製及搬移。

因為上述的問題，使得近年來廣泛使用的網路儲存系統（network-attached storage，NAS），無法有效利用網路頻寬。為了解決 CPU 效能的瓶頸，在早期研究中注重於解決資料的搬移，如 splice [1]，sendfile [2]，MMBUF [3]，等。近年，則出現了 POE（Protocol Offload Engine）[4]，進一步以硬體處理網路協定。同時，我們看到以 Internet Wide Area RDMA Protocol（iWARP）[5-7] 搭配 TCP Offload Engine（TOE）[8]的方式來降低 CPU 負擔的解決方案。

雖然有上述硬體方法可用以解決 CPU 的負擔，但其所需的成本也較多，使得傳統的中小階產品，較不願意加入 TOE 及 iWARP。但近年來系統晶片 System-On-Chip（SOC）以及晶片設計工具的技術進步，將 TOE 加入系統的代價降低，並且搭配 zero-copy [9, 10]的相關技術來達到高效能低成本的嵌入式網路儲存式伺服器，這是設計上的趨勢。

1.2 研究動機

研究發現，僅僅憑藉著 TOE，可能還是無法將 NAS server 的效能大幅提升，還需要軟體與硬體的整合及 zero-copy。一般 NAS server 針對讀取，已經支援 sendfile[11]的系統功能，以達到近似 zero-copy 的作法；然而在寫入方面，依然是需要多次的資料複製。因此為了達到高效能低成本的嵌入式網路儲存式伺服器，

我們分析軟體處理的瓶頸並針對寫入的效能加以改善，以及提出軟體與硬體整合方案的效能。在開放式的平台 Linux 上，改進開放原始碼的 NAS 服務程式 Samba [12] 的效能。這個研究所使用的效能分析及改進技術也可適用於類似的嵌入式網路儲存式系統。

1.3 論文架構

本論文其餘章節組織如下：第二章介紹本論文研究主題的相關技術及背景知識；進行相關技術的探討，第三章描述實驗平台、相關測量工具、實驗方法及相關測量方式；第四章描述我們在實驗平台上探測效能並找出主要瓶頸所在的過程及方法；第五章提出我們針對主要的瓶頸所實作的效能改善方法；在第六章中，我們報告實驗的結果並加以討論；在第七章，我們加以預測加入 TOE 後的效能；最後，在第八章提出本論文的結論及未來展望。

第2章 相關研究及背景知識

2.1 相關研究

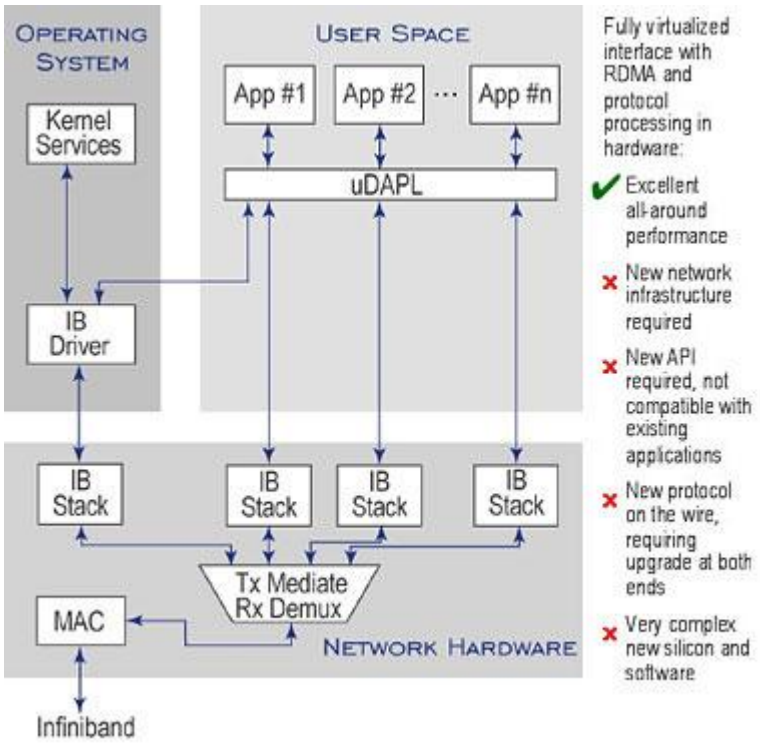
目前市面上已有的低階 NAS 產品，效能都不盡裡想。舉例說明，Tom's Guide 網站[13]針對於 D-Link MediaLounge DSM-G600 Wireless G Network Storage Enclosure 所做的效能檢測報告，列出 DSM-G600 的 specification（參見圖表 1 圖表 3）。由於採用低成本的 170MHz 的 PowerPC，於 1Gbps Ethernet 下最高的寫入效能只達到~7MBps = 56Mbps（參見圖表 4），只使用到 1 gigabits 網路頻寬的 5%左右。由此得知，目前低階的 NAS 還具有很大的效能改善空間。

為了處理高速網路上資訊的傳輸，CPU 往往必須負擔大量的網路處理及 I/O 運算。因此便有相關研究提出利用 TCP Offload Engine 的方式來達到減輕 CPU 的負擔。此一作法是將原本利用軟體執行的 TCP/IP 網路協定處理的工作轉交由硬體來處理，以減輕 CPU 的負擔[14, 15]。

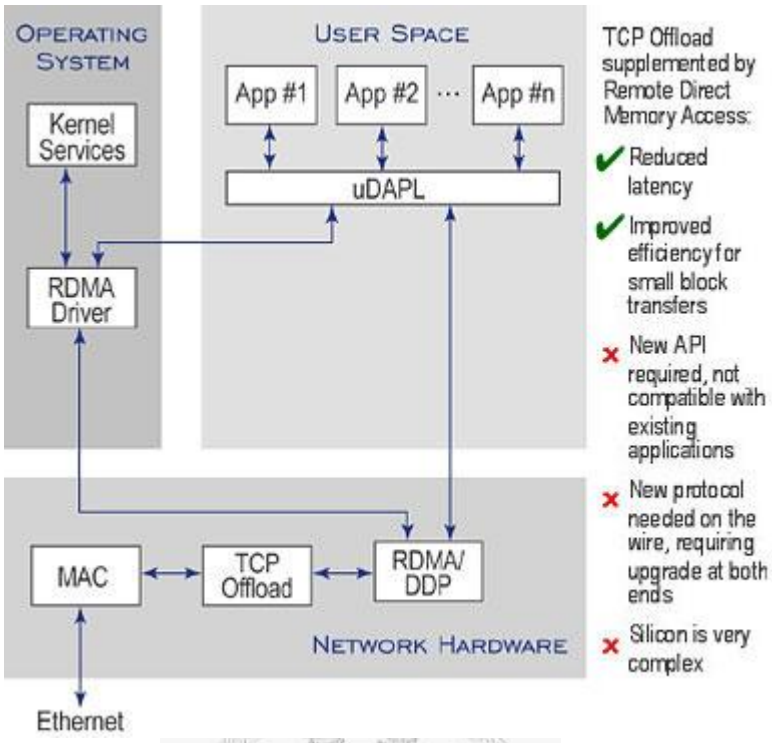
除了花費大量資源在網路協定處理外，CPU 還花費了大量時間於應用程式層（application layer）與作業系統（OS）間的資料搬移，於是便有研究提出以 OS-bypass 的一個方式將資料不經過 OS 直接傳入到 application layer，也就是 Remote Direct Memory Access（RDMA），直接將資料傳入到 application layer，以達到 zero-copy[16]。zero-copy 主要的目的就是要消除 application 與 OS 間的資料搬移，要達成 zero-copy 可以有好幾種方法，例如利用 Linux 所提供的系統呼叫 sendfile、splice，直接將資料從 kernel 緩衝區送至 socket 緩衝區，省略了將資料搬移到 user 緩衝區的動作，減少 CPU 的使用率。InfiniBand[17]架構也是達成 zero-copy 的一種方法（圖表 1），InfiniBand 架構是一種支持多併發鏈接的轉換線纜技術，每種鏈接都可以達到 2.5Gbps 的運行速度，但並不適用於 Wide Area Network（WAN），只應用於伺服器與伺服器，伺服器和儲存設備，以及伺服器和網路之間的通信。

雖然 InfiniBand 可以解決資料搬移的問題，但卻需要一種新的操作系統及新的應用軟體來運作，所需的成本也較高。

此外也有相關研究提出所謂的 RDMA over TCP/IP 也就是 iWARP[18]（圖表 2），主要也是為了達成 zero-copy，但需要額外 API 來與使用者程式溝通。使得程式設計者必須重新撰寫，以符合 iWARP 的介面，以致於難達成且不切實際。有研究顯示將 iWARP 實作於硬體並加上 TOE 會有相當大的效能改善，但所花費的成本與複雜度也相對提高[19]。



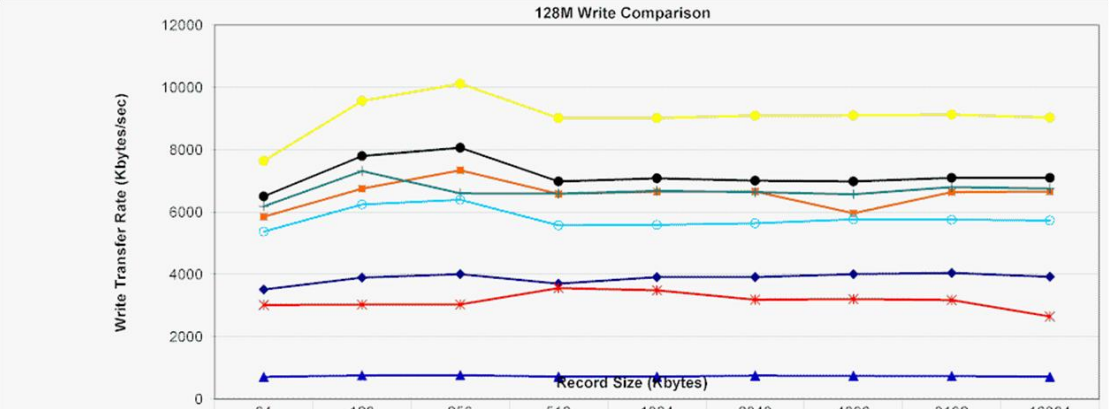
圖表 1 InfiniBand 架構



圖表 2 iWARP 架構

Specification	
CPU	170 MHz Freescale MPC8241 with MPC603e Motorola PowerPC core
RAM	32 MB ESMT M12L128168A
Flash ROM	4 MB Spansion S29GL032M90
USB	2 port USB2.0 NEC D720101GJ
IDE controller	Acard ATP-865
LAN	10/100/1000Mb IC Plus IP1000A
WIFI	2.4GHz 54Mbps Ralink RT2560 miniPCI
Serial	9600 8N1 TTL
Bootloader	U-Boot 0.2.0 without network support
Stock kernel	Linux-2.4.21-pre4
Stock C library	uClibc 0.9.26
Benchmarks	DSM-G600 PPC benchmarks

圖表 3 DSM-G600 Specification



	64	128	256	512	1024	2048	4096	8192	16384
Iomega StorCenter (1G)	7642	9562	10115	9016	9016	9096	9099	9126	9032
Iomega StorCenter	6503	7796	8065	6983	7081	7009	6984	7097	7095
Synology DS-101	5850	6748	7341	6585	6656	6659	5959	6641	6657
D-Link DSM-G600 (1G)	6182	7315	6599	6589	6682	6637	6567	6798	6756
D-Link DSM-G600	5371	6243	6392	5573	5587	5639	5764	5752	5728
Maxtor Shared Storage	3515	3898	4008	3701	3913	3910	4009	4047	3920
LaCie Ether Disk mini	3013	3028	3036	3559	3486	3185	3206	3174	2649
D-Link DSM-G600 (11g)	710	755	763	717	713	749	738	733	716

圖表 4 SMB Write Performance（D-Link DSM-G600 NAS）

2.2 相關背景知識

2.2.1 Samba 服務常式簡介

SMB/CIFS（Server Message Block/Common Internet File System）[20]是由微軟開發的一種軟體程序級的網路傳輸協議，主要用來使得一個網路上的機器共享電腦文件、印表機、串列埠和通訊等資源。它也提供認證的行程間通訊機能。它主要用在裝有 Microsoft Windows 的機器上，在這樣的機器上被稱為 Microsoft Windows Network。

Samba[21]是在 1991 年由 Andrew Tridgwell 所開發出來的軟體，使 UNIX 系列的作業系統能與微軟 Windows 作業系統的 SMB/CIFS（Server Message Block/Common Internet File System）網路協定做連結。此軟體在 Windows 與 UNIX 系列 OS 之間搭起一座橋梁，讓兩者可經由網路互相存取檔案等資源。

及 1996 年，約於昇陽推出 WebNFS 的同時，微軟提出將 SMB 改稱為 Common Internet File System（CIFS）。此外微軟還加入了許多新的功能，比如符號連結、硬

連結、提高文件的大小。微軟還試圖支持直接聯繫，不依靠 NetBIOS，不過這個試圖依然處於嘗試階段，並需要繼續完善。微軟向互聯網工程工作小組提出了部分定義作為網際網路草案。不過這些提案現在均已過期。

Samba 主要又分為三個 deamon：nmbd、smbd、winbind。

- nmbd：用於管理工作群組，NetBIOS name 的解析，利用 UDP 協定開啟 port 137，138 來負責名稱解析的任務。
- smbd：主要用於管理 Samba 主機分享的目錄、檔案與印表機。利用 TCP 協定來傳輸資料，開放的 port 為 139，445。
- winbind：用於統一管理 UNIX 和 NT 上的帳號，於 UNIX 主機裡，可以查看 NT 的用戶及群組訊息，就好像這些訊息是 UNIX 本地的一樣。

以伺服器的模式來說，smbd 主導了檔案傳輸的重要部分，故我們就對於 smbd 做相關主要的探討。其中以程式大小來看，Samba 程式碼約 50 萬行，而 smbd daemon 以及相關 lib 約佔 30 萬行，單 smbd daemon 就占了 Samba 約 60%，因此我們沒辦法以單看 smbd source code 來了解 smbd 的行為，我們需利用下列工具來幫助我們了解 smbd 的行為。

經由以下的工具我們可以快速的得到 smbd daemon 大概分為主要的五個處理部分（圖表 5），等待資料接收完並存在於 TCP poll，接著進行讀取以 64KB 為上限（因 TCP 處理協定暫存區的限制），接著讀取客戶端的命令，並取得相關資料長度的資料到使用者暫存區，接著處理資料，根據客戶端的要求，提供不同的處理函式，如要求寫入，則根據客戶端要求寫入的大小，寫入到磁碟暫存區，另外回應一個 ACK 給客戶端並加上 SMB header，並繼續等待下一筆資料，完成一次資料處理的循環。

圖表 5 Smbd 流程圖

2.2.2 Samba 讀取效能

Samba NAS server 針對於讀取，已經支援 sendfile[11]的系統功能，以達到近似 zero-copy 的作法，於 Samba 讀取的效能，我們也有進行相關量測、分析和改進，過程及結果在王俊文的論文中詳述[22]。

2.2.3 TCP/IP Offload Engines（TOE）

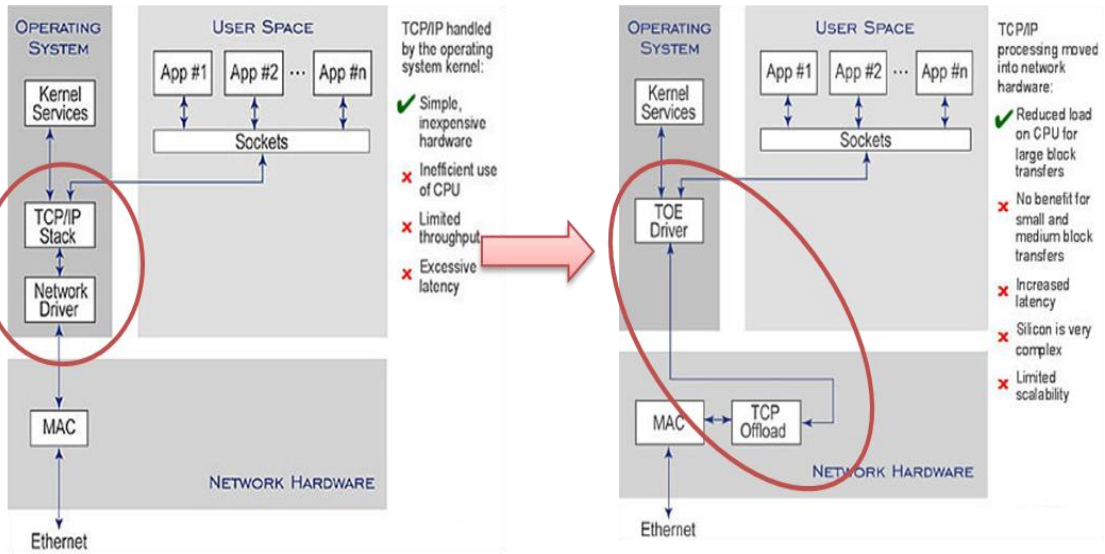
由於網路頻寬和速度的快速成長，傳統運行 TCP/IP 協定處理的方式已經越來越成為高效能網路計算的瓶頸。在目前的乙太網路中，TCP/IP 協定的處理都是透過軟體方式在中央處理器上實現。當網路速度達到 Gigabit 數量級時，CPU 就越來越繁忙，其中大部分處理的負擔都是來自於對 TCP/IP 協定的處理，例如對 IP 數據包的校對處理、對 TCP 數據流的可靠性和一致性處理。大量協定數據還需要透過

I/O 中斷進行操作，不斷在網路接口緩衝區和應用程式內存之間進行數據交換，這些額外負擔大大地降低了 CPU 的處理效率，增加了應用計算的平均等待時間。按照 CPU 對網路數據流的處理比率分析，大概 CPU 每處理 1bit 網路數據，就將消耗 1Hz 的處理性能，也就是說需要 20GHz 的 CPU 處理能力於全滿的負荷下運行才能滿足 10Gbps 乙太網路的處理要求。

同時，由於目前對 TCP/IP 協定進行處理都是採用通用 CPU 及其配套的系統結構，而這種體系下 CPU 的主要功能式進行一般計算，並非進行輸入輸出操作。因此在網路頻寬和速度快速發展下，網路速度高於 CPU 對 TCP/IP 協定的處理速度，導致系統的輸出輸入成為網路瓶頸。

為了在高速網路環境中改進和優化服務器效能，減輕對網路協定的處理負荷，TCP/IP 卸載引擎（TOE，TCP/IP Offload Engine）技術應運而生[23]。

TOE 技術的基本思想是分擔 CPU 對 TCP 和 IP 協定的處理，將協定處理過程放到高速網卡等硬體上完成，其中包括 TCP、IP、UDP、ICMP 等子協定處理。（圖表 6）將原來通過軟體方式處理的 TCP/IP 協定放在專門的硬體上完成，從而將應用和網路分離處理，會使 10G 乙太網路環境中應用服務器的 CPU 資源利用率大大提高，顯著地改善服務器性能。



圖表 6 Conventional Ethernet and TCP Offload Architecture

第3章 實驗設計與效能測量架構

為了分析 Samba service 在低階 NAS 下的效能狀況，我們於 server 端使用較低階的處理器（Intel Pentium-III 866MHz）（表格 1），以符合低階 NAS 所使用的處理器，接著於 client 端我們使用速度較快的處理器（表格 2），以防止效能瓶頸在 client 端，另外 server 與 client 連接於 1Gbps Ethernet 的環境。

於實驗上我們利用跳線直接連接 client 與 server 端，再於 client 端利用 Iometer 產生不同的 workload 來對 server 進行測試。接著從 Iometer 獲得 IOPS、bandwidth、average response time、及 client 端的 CPU utilization 等資訊。另外於遷移平台上 client 於 host 上直接與 host 的 virtual machine（server）做溝通（表格 3），於 host 端產生 workload 對 virtual machine 做測試。最後再將遷移後的程式碼轉移到實際機器來做測量。

於圖表 7 右邊為我們 NAS server，左邊則為 client，每個 client 皆執行 Iometer 根據參數產生 workloads 來對 NAS server 做測試，接著我們可以從 Iometer 取得 IOPS、bandwidth、average response time、及 client 端 CPU utilization 等資訊，並加以分析。

於圖表 8 我們以時間軸來看 client 與 server 間的關係，request 與 response 間即為 client 端的 response time，接著於 server 端藉由 *sys_select* 這系統呼叫又分為等待 I/O 的時間，以及 Samba 處理時間，故我們進一步的針對於等待 I/O 部分及 Samba 處理部分做追蹤及分析。

3.1 實驗環境

於 server 端我們選用較低階的 CPU（Intel Pentium-III 866MHz），及 256MB 的 memory，以符合較低階 NAS 的一個環境，Samba 則選用 3.0.28 的正式版本，O.S. 則為 Linux 2.6.23，network card 則選用能運作於 1Gbps 的 Intel network card。client 端則選用速度較快的 CPU（AMD Athlon™ 64 X2 Dual 2.01GHz）以防止效能瓶頸於

client，並使用較大的 memory（768MB），O.S.則為 Windows XP，network card 則為與 server 端相同能運行於 1Gbps Ethernet 的 Intel network card，並於 client 端執行 Iometer 對 NAS server 做量測。

另外因我們將 Samba 主要處理的部分遷移至核心，於遷移的過程中很容易造成核心的崩潰，為了解決這原因，我們採用 vmware 的一個開發環境，輕易將當前核心的狀況做個備份，快速的恢復核心崩潰前的狀態，以防止開發階段核心崩潰的情形。為了於 vmware 平台上開發快速，我們採用較快速的 CPU（1.86GHz Intel Core 2 Duo E6300）及較大容量的記憶體（1 GB DDR2-533），以做為我們快速開發的平台。

實驗平台規格

CPU	Intel Pentium-III 866MHz
Memory	256MB
Samba	3.0.28
O.S.	Linux 2.6.23
Network Card model	Intel® PRO/1000 GT Dual Port Server Adapter

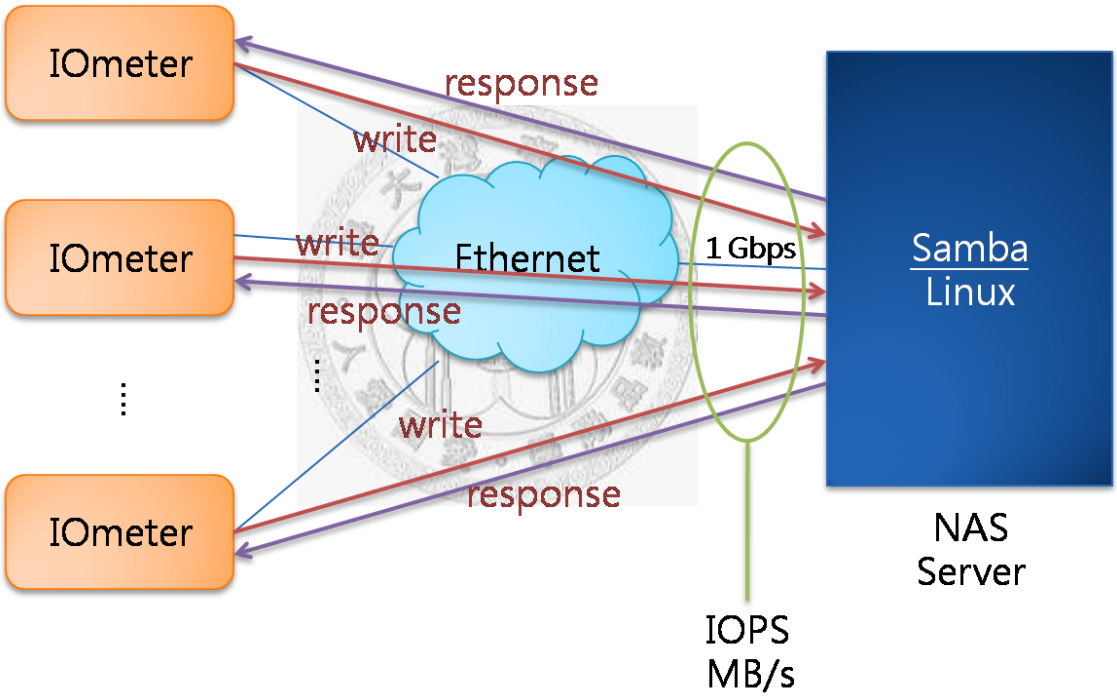
表格 1 server 端

CPU	AMD Athlon™ 64 X2 Dual 2.01GHz
Memory	768 MB
O.S.	Windows xp sp2
Network Card model	Intel® PRO/1000 GT Dual Port Server Adapter
Workload Generator	Iometer

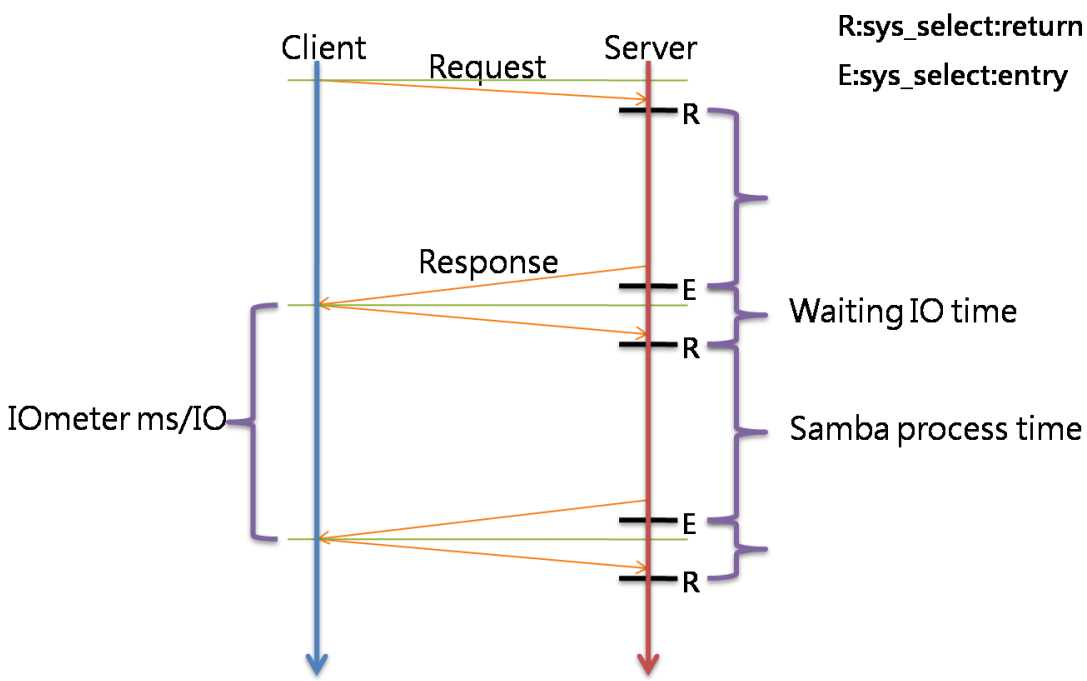
表格 2 client 端

CPU	Intel Core 2 Duo E6300
Memory	1 GB DDR2-533
O.S.	Linux 2.6.23
Network Card model	pcnet32
Workload Generator	Iometer

表格 3 遷移平台 server 端



圖表 7 實驗架構圖



圖表 8 實驗流程圖

3.2 量測工具介紹

為了瞭解整個系統的行為，我們利用多個效能量測工具，來針對 Samba 伺服器作量測研究。藉由這些工具，我們得知 Samba 的行為，及其所消耗的系統資源。再根據追蹤取得的結果，研究做最佳化處理的方法。

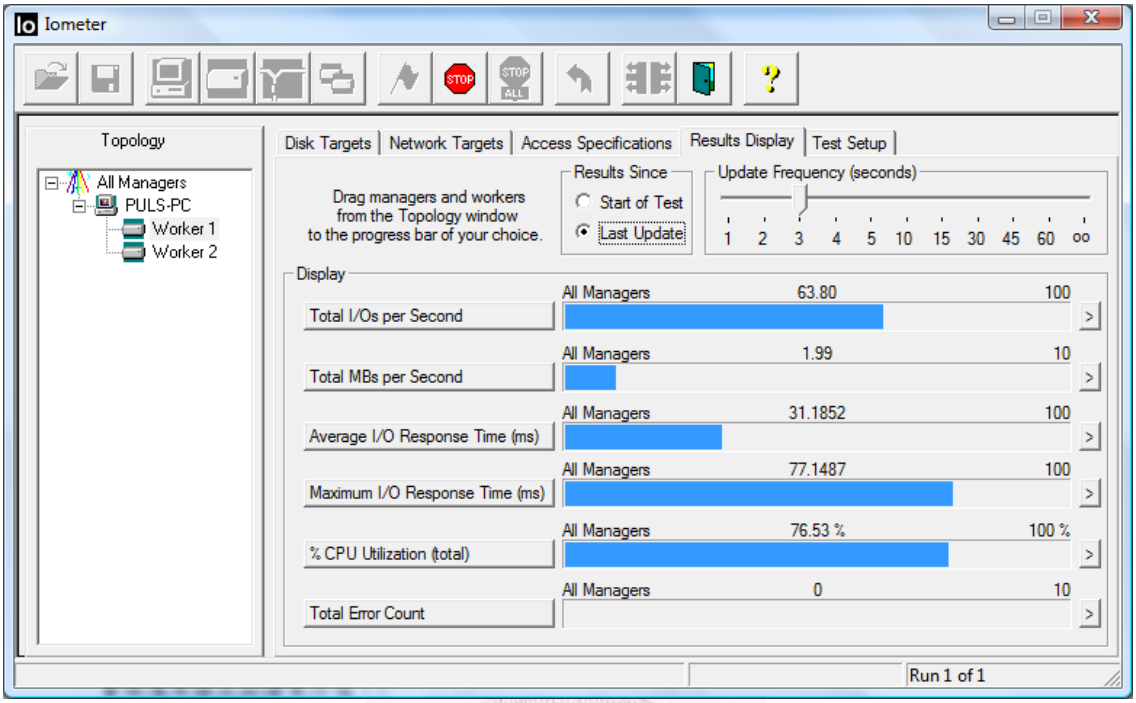
3.2.1 Iometer

Iometer[24]本身是一套開放原始碼軟體，是目前業界量測儲存系統 I/O 效能的標準工具。使用者可藉由負載產生器（workers）的參數調整，透過測試元件產生不同區塊大小得輸入輸出請求與讀取/寫入的分布比例，模擬網路伺服器、檔案伺服器以及線上交易資料庫（On-Line Transaction Processing system ，OLTP）實際應用程式的讀寫行為。由測試得出系統的最大 I/O 處理能力、資料吞吐率與 CPU 占用率（圖表 9），使用者即可評估在不同的應用環境下，伺服器儲存系統或是外接

儲存設備的效能。

並根據不同的參數設定可產生出不同的 workload 來對 Samba server 做測試，於本論文中參數設定於下表格 4 中，主要設定的參數為

- Maximum Disk Size：指 Iometer 對多大的 sector 進行存取，一般來講一個 sector 是 512Bytes，如果是 10000 的話，就是指 Iometer 只對 5MB 的 platter 進行存取。而我們將 Maximum Disk Size 設為 1 個 sector，避免 cache miss 造成 disk I/O 成為主要的效能瓶頸。
- Outstanding I/Os：client 端可同時執行的指令數。
- Workers：於 client 端同時執行的工作數。
- Transfer Request Size：用於設定傳輸的檔案大小；一般來說檔案越大，IOPS 就越小。檔案的大小與 IOPS 的乘積就可以得到系統的頻寬。
- Read/Write：指定讀或寫的動作佔所有 I/O 的百分比。
- Random/Sequence：只用於隨機操作和循序操作的機率分布。
- Transfer Delay：指每次 I/O 操作所花費的延遲時間，若為 0 即為連續的 I/O 操作
- Ramp Up Time：指開始測量多久後，Iometer 再開始紀錄傳輸的狀態。
- Run Time：指運行的時間，若為 0，則直到使用者按下 stop 按鈕，才會停止。



圖表 9 Iometer

Maximum Disk Size	1
Outstanding I/Os	1/2/32
Workers	1
Transfer Request Size	512Byte~1MByte
Read/Write	100% write
Random/Sequence	100% sequential
Transfer Delay（ms）	0
Ramp Up Time	10sec.
Run Time	1min
Workload Generator	Iometer

表格 4 Iometer configuration

3.2.2 Mpstat

Mpstat（Multiprocessor Statistics）[25]，為實時系統監控工具，可以報告使用者指定的週期 CPU 在週期中的使用狀況，例如：分配給系統、用戶、等待、空閒時間、系統呼叫、鎖競爭、中斷、錯誤、交叉呼叫，並將訊息存放在/proc/stat 文件中（如圖表 10）。

於本論文中將會分析%user（使用者空間所花費的時間），%sys（核心所花費的時間），%irq(中斷所花費的時間)，%soft(softirq 即 network driver 所花費的時間)，%idle(CPU 閒置的時間)。

```
[root@localhost ~]# mpstat 5
Linux 2.6.23 (localhost.localdomain)    07/07/2008

04:12:03 PM  CPU   %user   %nice    %sys %iowait    %irq   %soft  %steal   %idle    intr/s
04:12:08 PM  all    0.20    0.00    0.20    0.00    0.00    0.00    0.00   99.60     15.23
04:12:13 PM  all    0.00    0.00    0.40    0.00    0.00    0.00    0.00   99.60     16.40
04:12:18 PM  all    0.20    0.00    0.20    0.00    0.00    0.00    0.00   99.60      5.99
04:12:23 PM  all    5.20    0.00    0.80    0.00    0.20    0.00    0.00   93.80     10.60
04:12:28 PM  all    0.60    0.00    0.60    0.00    0.00    0.00    0.00   98.80      6.99
04:12:33 PM  all    0.20    0.00    0.00    0.20    0.00    0.00    0.00   99.60      9.40
04:12:38 PM  all    0.00    0.00    0.40    0.00    0.00    0.00    0.00   99.60      6.21
04:12:43 PM  all    0.00    0.00    0.20    0.00    0.00    0.00    0.00   99.80      9.00
```

圖表 10 Mpstat

3.2.3 VTune 效能分析器

VTune[26]效能分析器是 Intel 為眾多開發者們提供的專門針對尋找軟硬體效能瓶頸的一款分析工具。根據 VTune 所得到的統計資料，能得到運行程式中活動相對密集的區域（HotSpot）。HotSpot 不僅耗費大量時間，它也經常在以下事件中被發現：快取區失誤，分頁失誤，錯誤預測分支。而這類錯誤往往非常隱密，難以發現。但只要能找出並優化這些 HotSpot，便能達到事半功倍的效果。VTune 提供了採樣、呼叫圖、計數器監視器，一系列分析方案來幫助軟硬體開發者尋找 Hot spot。

我們利用 VTune 幫助我們以採樣的分析在核心中模組所花費的時間，以及幫助我們了解與 Samba 相關互動的模組，並使我們得知所有與 Samba 執行時所相關

的模組。

3.2.4 Strace

Strace[27]是一個系統呼叫追蹤器，跟蹤程式執行時的系統呼叫和所接收的信號，並將所呼叫到的系統呼叫名稱，參數和返回值輸出到標準輸出或者輸出到-o指定的文件（圖表 11）。

另外利用 Strace 將 Samba 執行時所呼叫到的系統呼叫追蹤出來，並加以分析傳入系統呼叫的參數及回傳值，更可得到有關於系統呼叫所花費的時間資訊，使我們能有效的分析與 Samba 有關的系統呼叫。

```
select(25, [19 21 24], [], NULL, {60, 0}) = 1 (in [21], left {56, 999000})
read(21, "\0\0\2@", 4)                  = 4
read(21, "\377SMB/\0\0\0\0\30\7\310\0\0\0\0\0\0\0\0\0\0\0\0\1\0\377\376\0\0\300\234"..., 576) = 576
pwrite64(25, "g\1\307_\233\304\203\243\344\212\277VJ\310;\325P-\235\272P\250/u\372j\364L?96\263"..., 512, 11776) = 512
write(21, "\0\0\0/\377SMB/\0\0\0\0\210\1\300\0\0\0\0\0\0\0\0\0\0\0\0\1\0\377\376"..., 51) = 51
gettimeofday({1215749758, 813185}, NULL) = 0
```

圖表 11 Strace

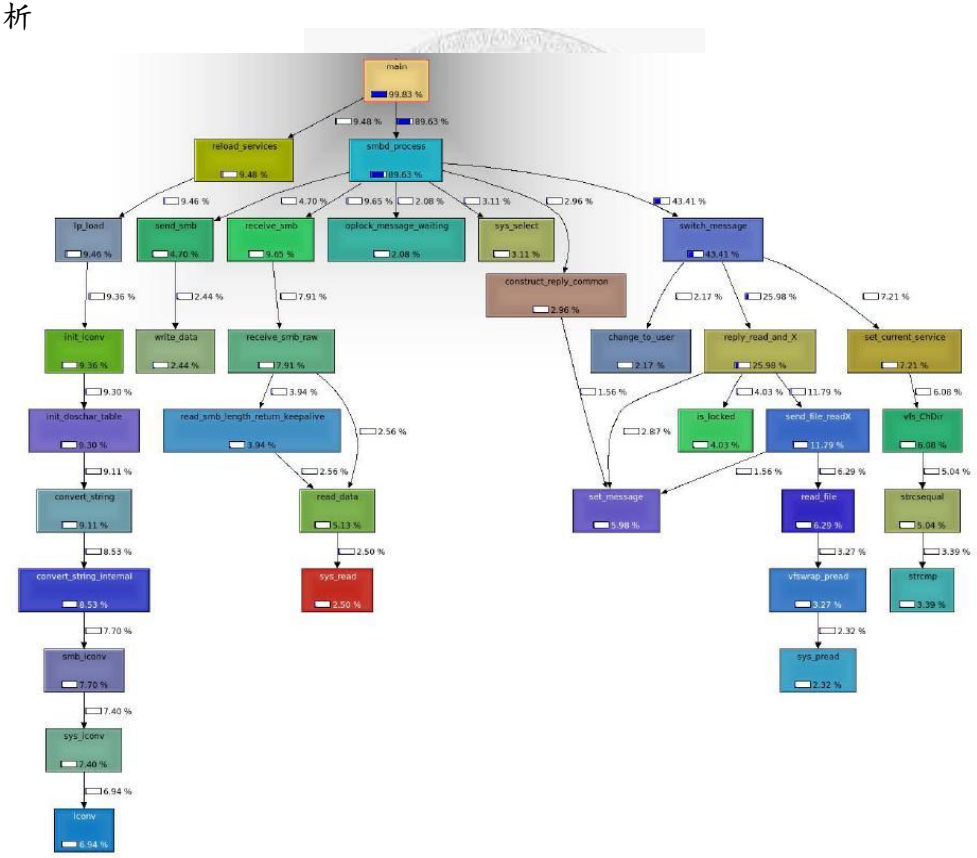
3.2.5 Valgrind

Valgrind[28]工具是一個 GPL 的軟體，用於 Linux 程序的記憶體調試和程式碼解析，可於它的環境中運行要測試的程式來監視記憶體的使用情況。Valgrind 工具包包含多個工具，如 Memcheck、Cachegrind、Helgrind、Callgrind、Massif 等。

- Memcheck 主要用於檢查未初始化的記憶體、使用已經釋放了的記憶體、使用超過 malloc 分配的記憶體空間、對堆疊的非法訪問、申請的空間是否有釋放、malloc/free/new/delete 申請和釋放記憶體的匹配、src 和 dst 的重疊。
- Callgrind 收集程序運行時的一些數據，函式呼叫關係等訊息（圖表 12）。
- Cachegrind 它模擬 CPU 中的一級快取 L1，D1 和 L2 二級快取，能夠精確的指出程序中 cache 的失誤和命中。如果需要，它還能夠為我們提供 cache 失誤次數，記憶體引用次數，以及每行程式碼，每個函式，每個模組，及整個程序產生的指令數。

- Helgrind 主要用來檢查多線程程序中出現的競爭問題。Helgrind 尋找內存中被多個線程訪問，而又沒有一貫加鎖的區域，這些區域往往是線程之間失去同步的地方，而且會導致難以發掘的錯誤。
- Massif 堆疊分析器，它能測量程序在堆疊中使用了多少內存。Massif 能幫助我們減少內存的使用，在帶有虛擬內存的現代系統中，能加速我們程序的運行，減少程序停留在交換區中的機率。

我們利用了 Valgrind 中 Callgrind 的工具幫助我們將 Samba 於執行時所呼叫到的函式以圖形的方式表示出來，並於圖中得知函式中被呼叫的次數，以幫助我們找出 Samba 中執行最頻繁的函式，並有效地針對於這些函式加以分析

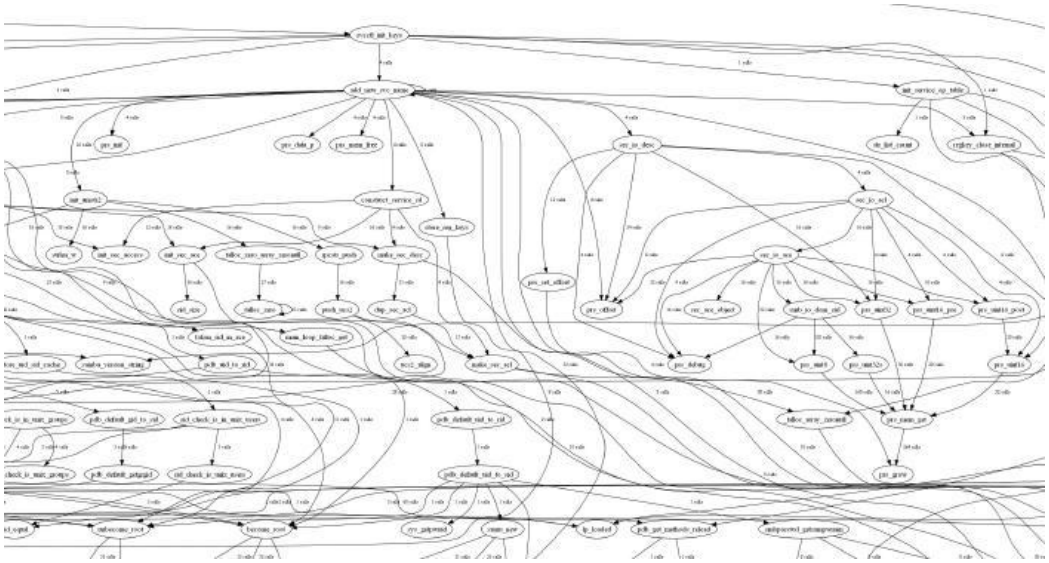


圖表 12 Valgrind 函式呼叫圖

3.2.6 Pvtrace

Pvtrace[29]主要利用 GCC function instrumentation 的機制。該機制出現於 GCC 2.x，由 Cygnus 所提出，在 GCC 中對應的選項為:"finstrument-functions"，會在每次進入與退出函式前呼叫"*__cyg_profile_func_enter*"與"*__cyg_profile_func_exit*"這兩個 hook function，故我們可在這兩個函式中收集分析數據，再利用 Graphviz[30]建立呼叫圖（圖表 13）。

於本論文中 PVtrace 主要為靜態追蹤的技術，需重新編譯原始碼，主要用於分析更詳盡的函式呼叫，以利於未來能與硬體快速整合。



圖表 13 Pvtrace 函式呼叫圖

3.2.7 Dtrace

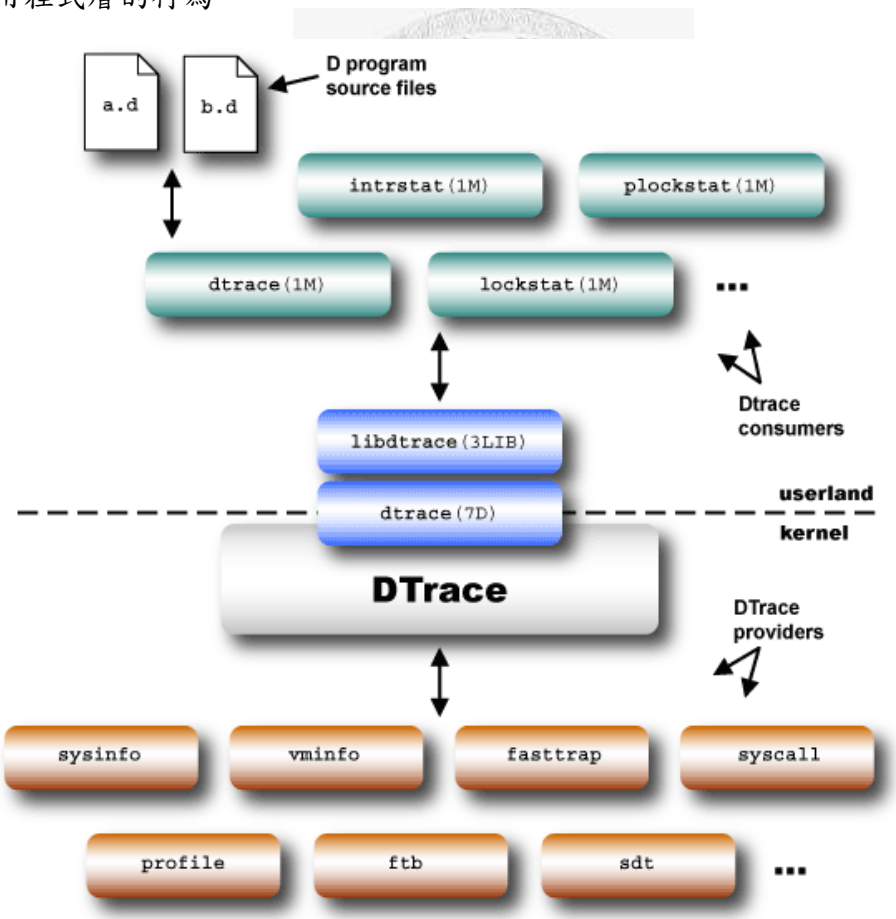
DTrace[31]即動態跟蹤（Dynamic Tracing），是 Solaris 10 的一個新功能，透過此一新功能，用戶能夠動態檢測操作系統核心和用戶進程（process），以更精確地掌握系統的資源使用狀況。

於 Solaris 中分散著 30000 多的位置指針，也叫探測器 probes，Dtrace 能驅動成千上萬的探測器，紀錄所關注的位置指定的數據，如果命中該 porbe，即可從該

地址顯示用戶進程或系統核心的數據，從而了解系統，包括:任何函式參數、核心的任何全域變數、函式呼叫時間、跟蹤堆疊，包括指名函式呼叫的代碼、函式呼叫時運行的進程、產生函式呼叫的線程。

於圖表 14 中，當執行 Dtrace 時，Dtrace 會先讀取 D program source，並與 libdtrace 結合轉譯成 intermediate code，再確定它是否為安全運行的一個程序，確定後將它載入內核，變成一塊內核模組，再利用 Dtrace providers 針對於欲追蹤的程式提供不同的追蹤資訊。

利用在 Solaris 上，與 Linux 皆為 Unix 系統平台的 Dtrace 工具，幫助我們進行在使用者空間的追蹤，了解 Samba 在執行時，函式間的控制流程，清楚了解 Samba 在應用程式層的行為。



圖表 14 Dtrace Architecture

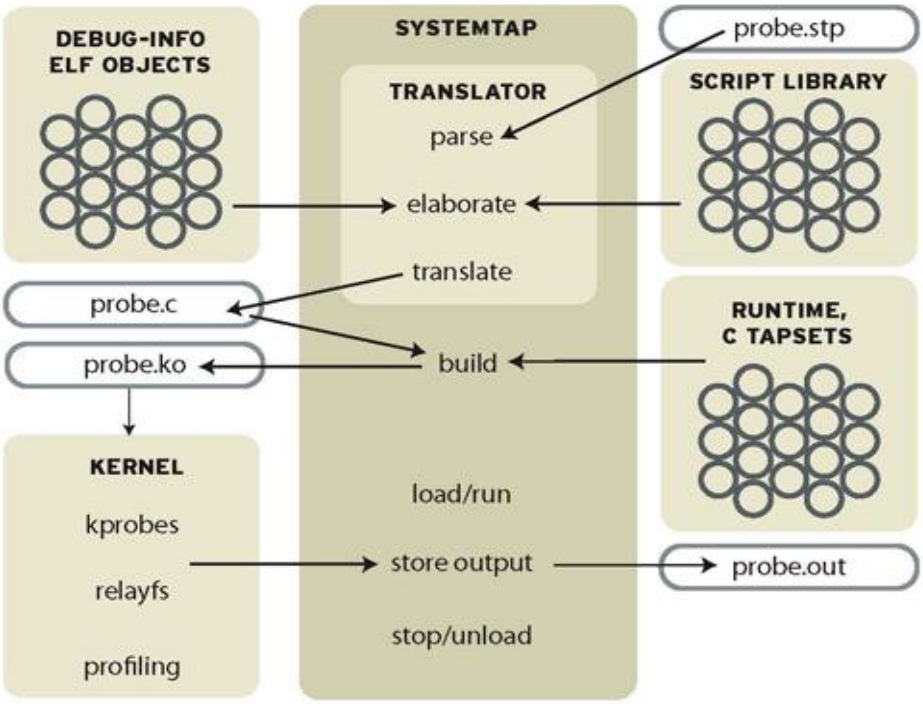
3.2.8 Systemtap

SystemTap[32]是一種新穎的 Linux 核心診斷工具，提供了一種從運行中的 Linux 核心快速和安全地獲取訊息的能力。可透過編寫或者重用簡單的腳本來收集內核的數據，而不需要再忍受修改源碼、編譯內核、重啟系統的漫長煎熬。

SystemTap 運行的過程依次分為五個階段（圖表 15），稱為 Pass 1 - Pass 5。

- Pass 1 - parse：這個階段主要是檢查輸入腳本是否存在語法錯誤，例如大括號是否配對，變數定義是否符合規範等。
- Pass 2 - elaborate：這個階段主要是對輸入腳本中定義的探測點或者用到的函式展開，不但需要綜合 SystemTap 的預先定義腳本庫，還需要分析核心或者核心模組的呼叫訊息。
- Pass 3 – translate： 在這個階段，將展開後的腳本轉換成 C 文件。前三階段的功能類似於編譯器，將.stp 文件編譯成為完整的.c 文件，因此又被合起來稱為轉換器（translator）。
- Pass 4 – build： 在這個階段，將 C 原始碼編譯成核心模組，在這過程中還會用到 SystemTap 的運行時函式庫。
- Pass 5 – run： 這個階段，將編譯好的核心模組插入核心，開始進行數據收集和傳輸。

為了得知 Samba 於執行時核心的行為，利用了 SystemTap 進一步得知 Samba 於系統呼叫下核心函式的控制路徑，也進一步的得知資料搬移的所在。



圖表 15 SystemTap 的基本原理

第4章 探測效能瓶頸

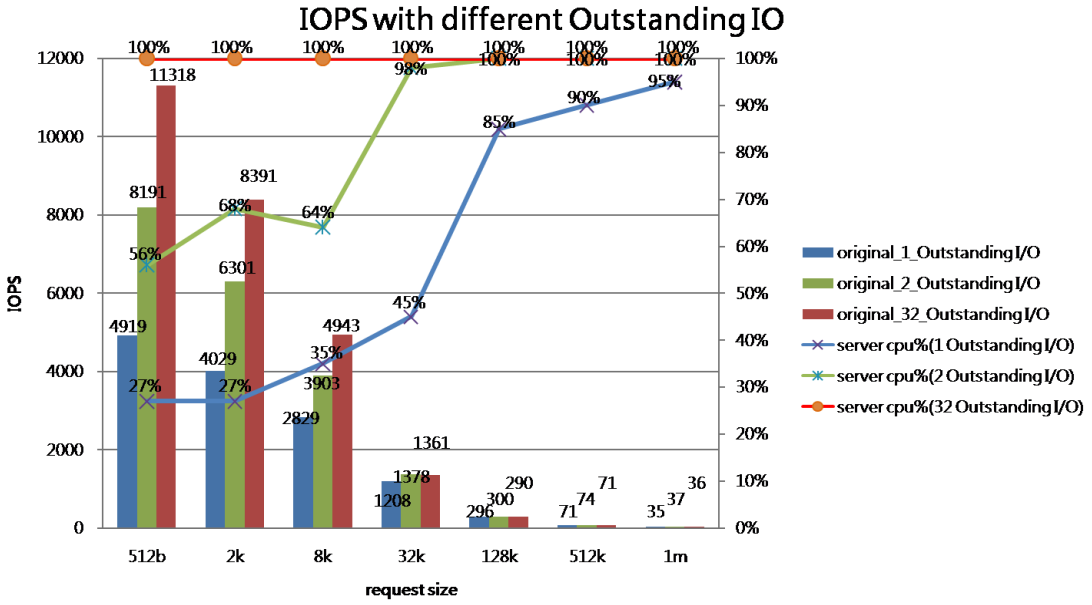
本章敘述我們利用在第三章所介紹的工具，分析 Samba 的行為以及系統呼叫（system call）的運作，同時統計 Samba 運行時各個程式功能（functions）消耗的資源，找出哪些資源花費最多。我們進一步取得函式間相互的關係，取得有時間關係的呼叫圖，以幫助我們針對 Samba 以及核心做最佳化的處理。

4.1 相異工作量之量測

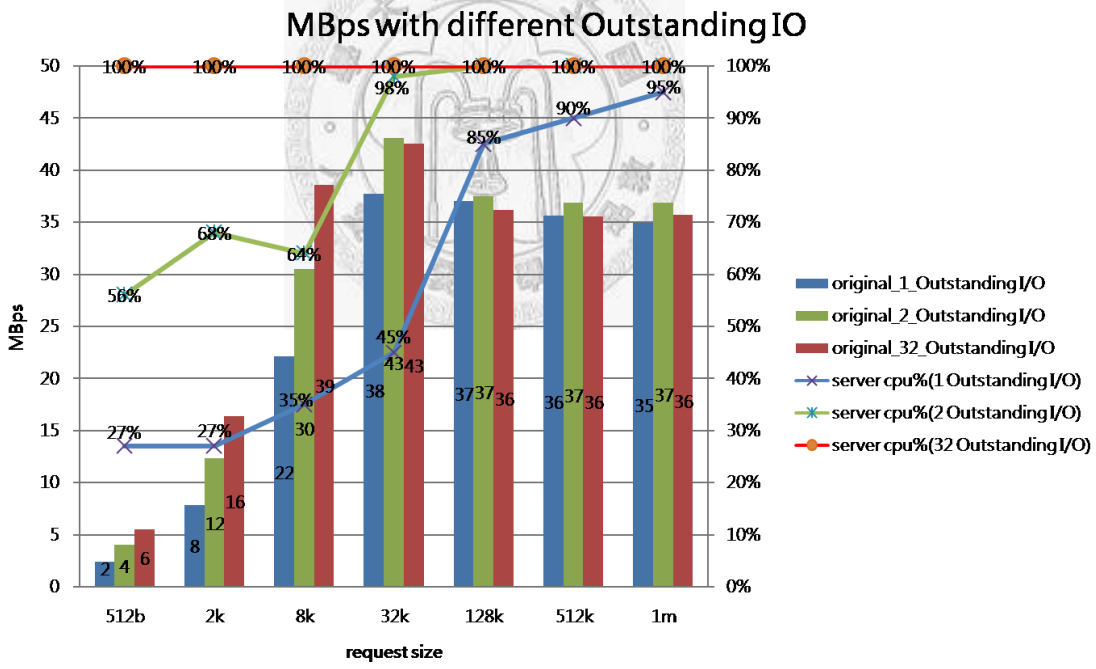
我們先以 Iometer 產生出不同的 workload 對 samba server 做寫入測試，以 1 個 outstanding I/O、2 個 outstanding I/Os 及 32 個 outstanding I/Os 去探測 samba server 所產生的效能結果。

以 IOPS 來看，圖表 16 顯示 Samba server，於 1 個 outstanding I/O 下接受遠端寫入的要求，可以達到 4919 的 IOPS。同時我們發現 CPU utilization 只有 27%，可推測 CPU 應還能處理更多的 I/O，於是我們使用 2 個 outstanding I/Os 進一步的對伺服器端做測量。於 2 個 outstanding I/Os 下 IOPS 成長到了 8194，CPU utilization 也上升到了 56%左右。因此我們可看到隨著 outstanding I/Os 的成長，IOPS 與 CPU utilization 皆有明顯的成長。更進一步的以 32 個 outstanding I/Os 來對伺服器端進行測試時，得到 11318 的 IOPS，同時 CPU utilization 已達到 100%，顯示 CPU 無法負荷更大的運算量。

接著再以頻寬來看（圖表 17），我們發現於較大的 request file size 下所達到的頻寬也被限制在 37MBps 下，從 CPU utilization 來看我們可以得知在較大的 request file size 下 1 個 outstanding I/Os 就可以達到將近 100%的 CPU utilization，因此增加 outstanding I/O 的數量，也沒辦法進一步增加頻寬。以 32KB request file size 為例 ，於 1 個 outstanding I/O 時 CPU utilization 只達到 45%； 2 個 outstanding I/Os 時，頻寬明顯的上升到 43MBps，CPU utilization 也達到將近 100%。由以上分析我們得知，CPU 為主要的效能瓶頸。



圖表 16 Iometer result（IOPS）



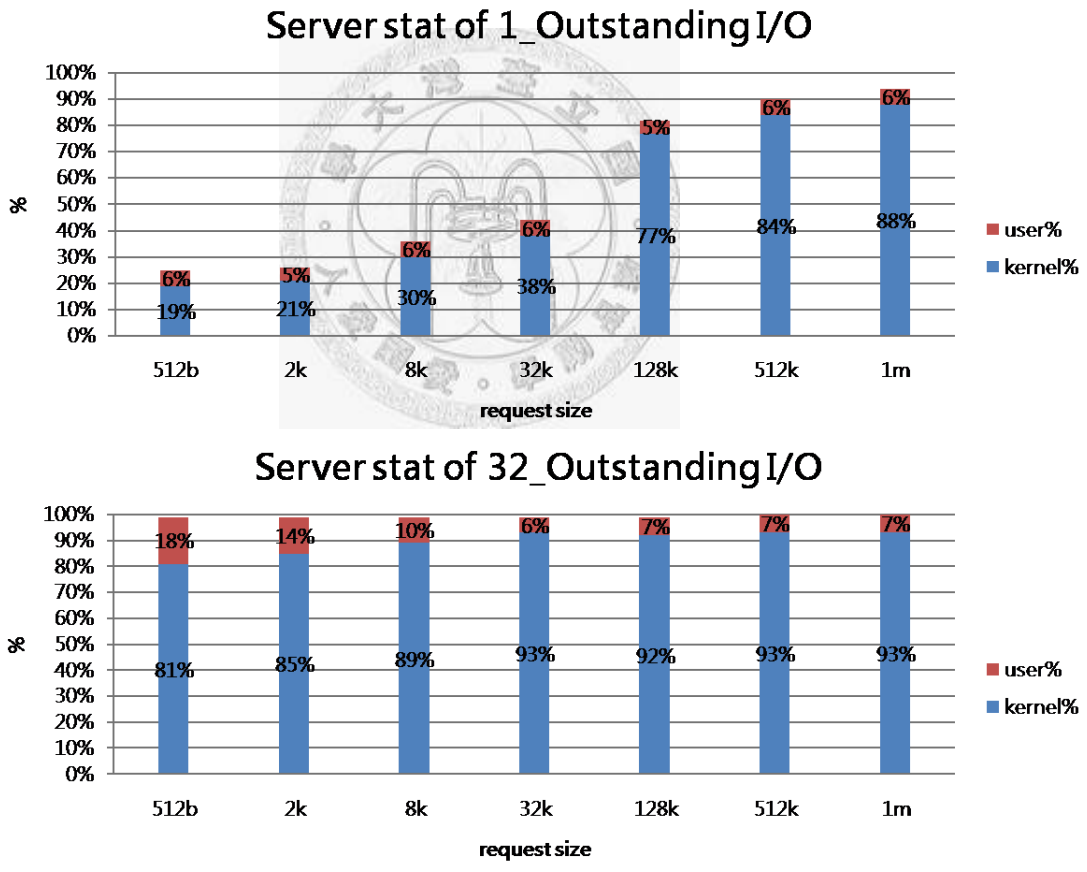
圖表 17 Iometer result（MBps）

4.2 檢視系統狀態

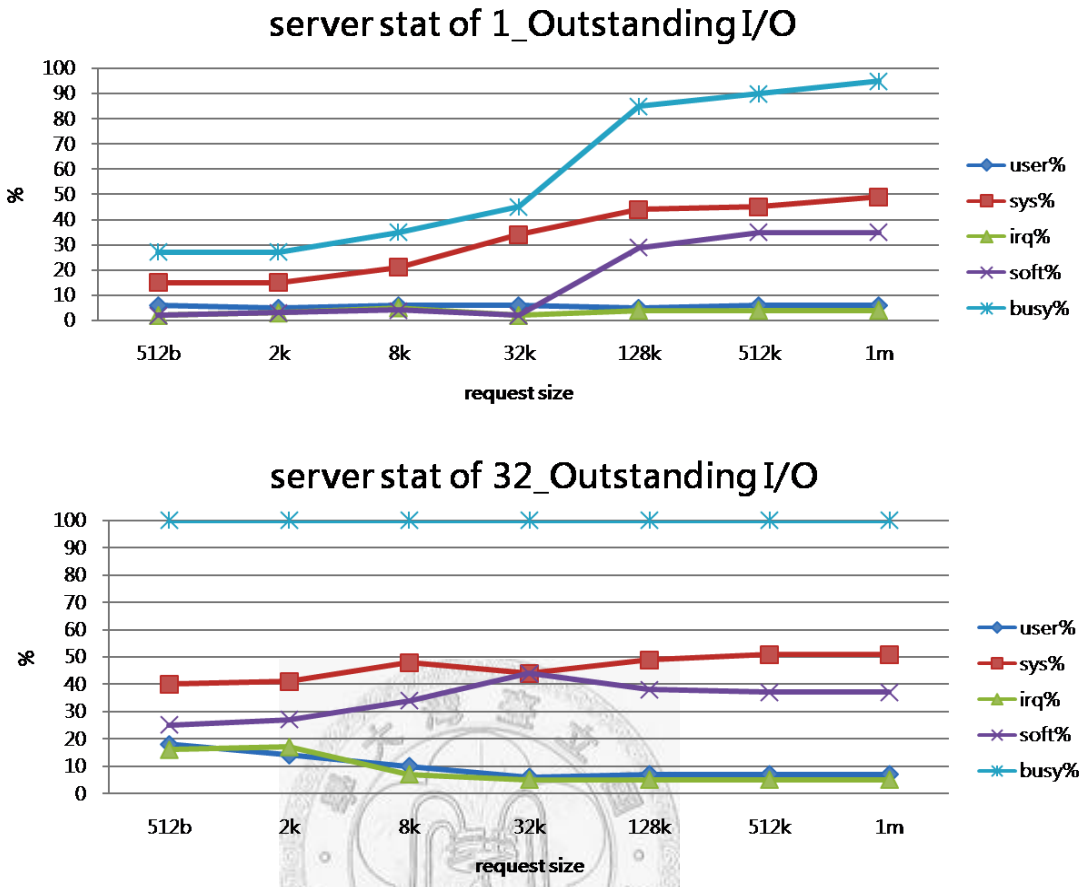
在我們得知 CPU 為主要的效能瓶頸後，我們進一步的去觀看系統中 CPU 資源分布的一個狀況，我們先以 CPU 資源於 user space 及 kernel space 的一個分配情況

來看，於圖表 18 得知於 1 個 outstanding I/O 下 user 所花費的時間僅佔整個系統的 5%左右，且於 32 個 outstanding I/Os 下平均也只有 10%左右，所佔的比例微乎其微。因此我們得知，user space 的工作並不是造成 CPU 負荷沉重的主要原因。

因此我們把重點放在對於 kernel 所花時間的資源做進一步的分析， mpstat 的報告（圖表 19）顯示，於 1 個 outstanding I/O 的情況下系統所佔的時間就將近為 CPU 總執行時間的一半。而 32KB 後 softirq 也佔了將近 30%的 CPU 總執行時間，發現單 sys 與 softirq 所佔的時間就佔了 80%的 CPU 總執行時間。且於 32 outstanding I/Os 下更明顯得知 sys 與 softirq 就佔了將近 80%的 CPU 資源，而 user 並沒占太多的 CPU 資源，因此有必要進一步針對系統所花費的部分進行檢測。



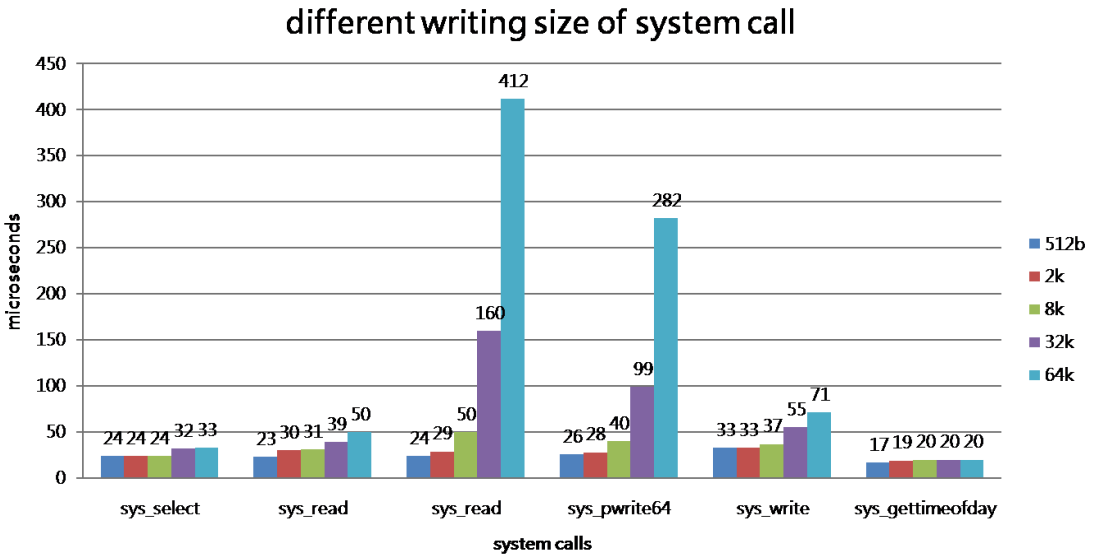
圖表 18 user 與 kernel 之 CPU 資源分布情形



圖表 19 Samba server stats

4.3 系統呼叫之評估

根據上述所得出的結果，我們得知核心占了大部分的時間，為了得知系統於 Smbd 運行時主要花費時間的部分，故我們利用 Strace 將 Smbd 運行時所執行的系統呼叫追蹤出來（圖表 20），並根據不同寫入要求的大小，做測量評估。我們得知在 Smbd 運行時會依序執行 *sys_select*、*sys_read*、*sys_read*、*sys_pwrite64*、*sys_write*、*sys_gettimeofday*。於圖表 20 我們利用 Strace 中時間的一個資訊，發現第二次的 *sys_read* 與 *sys_pwrite64* 隨著要求寫入檔案的大小越大，所花的時間也越多。原因在於 user space 與 kernel space 之間的兩次資料複製，要求寫入的檔案大小越大，所花的時間也越多。



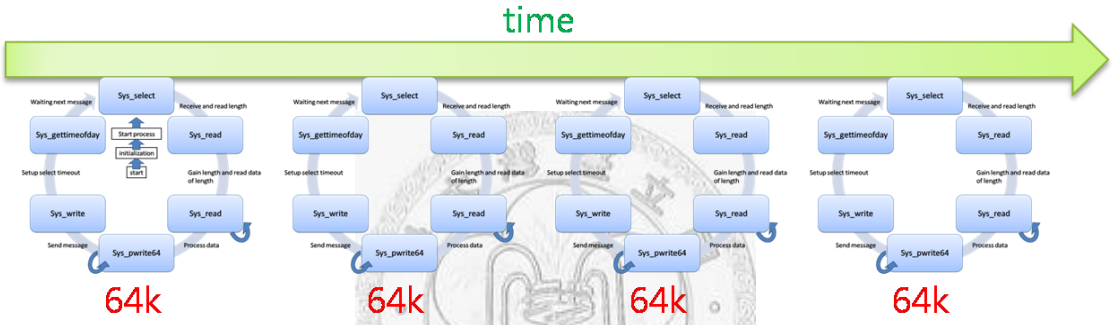
圖表 20 系統呼叫比較圖

4.4 研討 smbd 於網路部分之處理

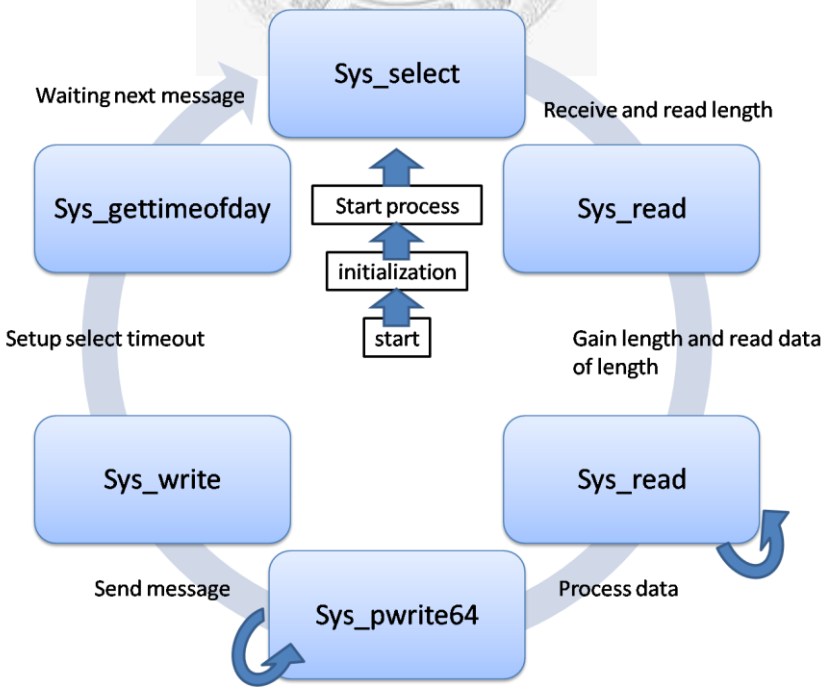
另外從網路部分來看，我們利用了 WireShark 去擷取 Samba 在運作時網路所傳送的封包資料及行為（圖表 21），於 圖表 21 我們可以得知 TCP 將資料切成多個相同大小的 payload（~1.5KB）送於 Samba 伺服器。當資料收到後，若判斷為 SMB protocol 則交由 Samba service 來做處理，於處理完成後再送訊息給客戶端表示處理完成，並發現因 TCP 處理協定暫存區的限制，必須將資料切成 64KB 大小的上限，使得 Samba 於處理較大的資料上以 64KB 為一次的循環。例如 1MB 的 request size 就會分成 16 次的 64KB 資料來處理，也就是 Samba 需要 16 次的循環，花費 96 次的系統呼叫來做處理。此外經由 strace 我們可以了解每一次的循環都包含固定的六個系統呼叫（圖表 22、圖表 23），*sys_select* 用來等待下一筆的資料，第一次 *sys_read* 讀取資料的長度，第二次 *sys_read* 則根據第一次 *sys_read* 讀取到的資料長度去將資料從 socket buffer 複製到 user buffer，*sys_pwrite64* 則將處理完的資料寫入到磁碟，接著 *sys_write* 回應訊息給客戶端（51bytes），*sys_gettimeofday* 取得時間的資訊用於設定 *sys_select* 等待資料的時間，以結束一次處理資料的循環。

4576	30.273851	192.168.239.1	192.168.239.131	SMB	Write AndX Request, FID: 0x143a, 65536 bytes at offset 0
4622	30.274817	192.168.239.1	192.168.239.131	SMB	Write AndX Request, FID: 0x143a, 65536 bytes at offset 65536
4668	30.276361	192.168.239.1	192.168.239.131	SMB	Write AndX Request, FID: 0x143a, 65536 bytes at offset 131072
4677	30.284679	192.168.239.131	192.168.239.1	SMB	Write AndX Response, 65536 bytes
4714	30.285242	192.168.239.1	192.168.239.131	SMB	Write AndX Request, FID: 0x143a, 65536 bytes at offset 196608
4760	30.287285	192.168.239.1	192.168.239.131	SMB	Write AndX Request, FID: 0x143a, 65536 bytes at offset 262144
4805	30.287675	192.168.239.1	192.168.239.131	SMB	Write AndX Request, FID: 0x143a, 65536 bytes at offset 327680
4851	30.290648	192.168.239.1	192.168.239.131	SMB	Write AndX Request, FID: 0x143a, 65536 bytes at offset 393216
4875	30.292268	192.168.239.131	192.168.239.1	SMB	Write AndX Response, 65536 bytes
4897	30.292669	192.168.239.1	192.168.239.131	SMB	Write AndX Request, FID: 0x143a, 65536 bytes at offset 458752
4943	30.293827	192.168.239.1	192.168.239.131	SMB	Write AndX Request, FID: 0x143a, 65536 bytes at offset 524288
4989	30.295212	192.168.239.1	192.168.239.131	SMB	Write AndX Request, FID: 0x143a, 65536 bytes at offset 589824
4990	30.295772	192.168.239.131	192.168.239.1	SMB	Write AndX Response, 65536 bytes
5036	30.297023	192.168.239.1	192.168.239.131	SMB	Write AndX Request, FID: 0x143a, 65536 bytes at offset 655360
5037	30.298083	192.168.239.131	192.168.239.1	SMB	Write AndX Response, 65536 bytes
5082	30.299513	192.168.239.1	192.168.239.131	SMB	Write AndX Request, FID: 0x143a, 65536 bytes at offset 720896
5084	30.300258	192.168.239.131	192.168.239.1	SMB	Write AndX Response, 65536 bytes
5129	30.301165	192.168.239.1	192.168.239.131	SMB	Write AndX Request, FID: 0x143a, 65536 bytes at offset 786432
5131	30.302073	192.168.239.131	192.168.239.1	SMB	Write AndX Response, 65536 bytes
5177	30.302931	192.168.239.1	192.168.239.131	SMB	Write AndX Request, FID: 0x143a, 65536 bytes at offset 851968
5178	30.304024	192.168.239.131	192.168.239.1	SMB	Write AndX Response, 65536 bytes
5223	30.304764	192.168.239.1	192.168.239.131	SMB	Write AndX Request, FID: 0x143a, 65536 bytes at offset 917504
5225	30.305858	192.168.239.131	192.168.239.1	SMB	Write AndX Response, 65536 bytes
5271	30.306798	192.168.239.1	192.168.239.131	SMB	Write AndX Request, FID: 0x143a, 65536 bytes at offset 983040
5272	30.307768	192.168.239.131	192.168.239.1	SMB	Write AndX Response, 65536 bytes
5273	30.309583	192.168.239.131	192.168.239.1	SMB	Write AndX Response, 65536 bytes
5275	30.310829	192.168.239.131	192.168.239.1	SMB	Write AndX Response, 65536 bytes
5276	30.312088	192.168.239.131	192.168.239.1	SMB	Write AndX Response, 65536 bytes
5278	30.313303	192.168.239.131	192.168.239.1	SMB	Write AndX Response, 65536 bytes
5279	30.314648	192.168.239.131	192.168.239.1	SMB	Write AndX Response, 65536 bytes
5282	30.317788	192.168.239.131	192.168.239.1	SMB	Write AndX Response, 65536 bytes
5284	30.319015	192.168.239.131	192.168.239.1	SMB	Write AndX Response, 65536 bytes

圖表 21 WireShark（1MB）



圖表 22 Behavior of 256KB Write



圖表 23 Samba 系統呼叫循環圖

4.5 追蹤流程

為了進一步分析 smbd 跟核心的行為，我們利用 Dtrace 在使用者空間追蹤到的資料與 Systemtap 在核心空間追蹤到的資料，得到了一個包含 TCP 網路協定、Samba 服務常式、核心、網卡驅動程式等完整的追蹤資料，我們可由這資料得到函式與函式之間前後的關係。

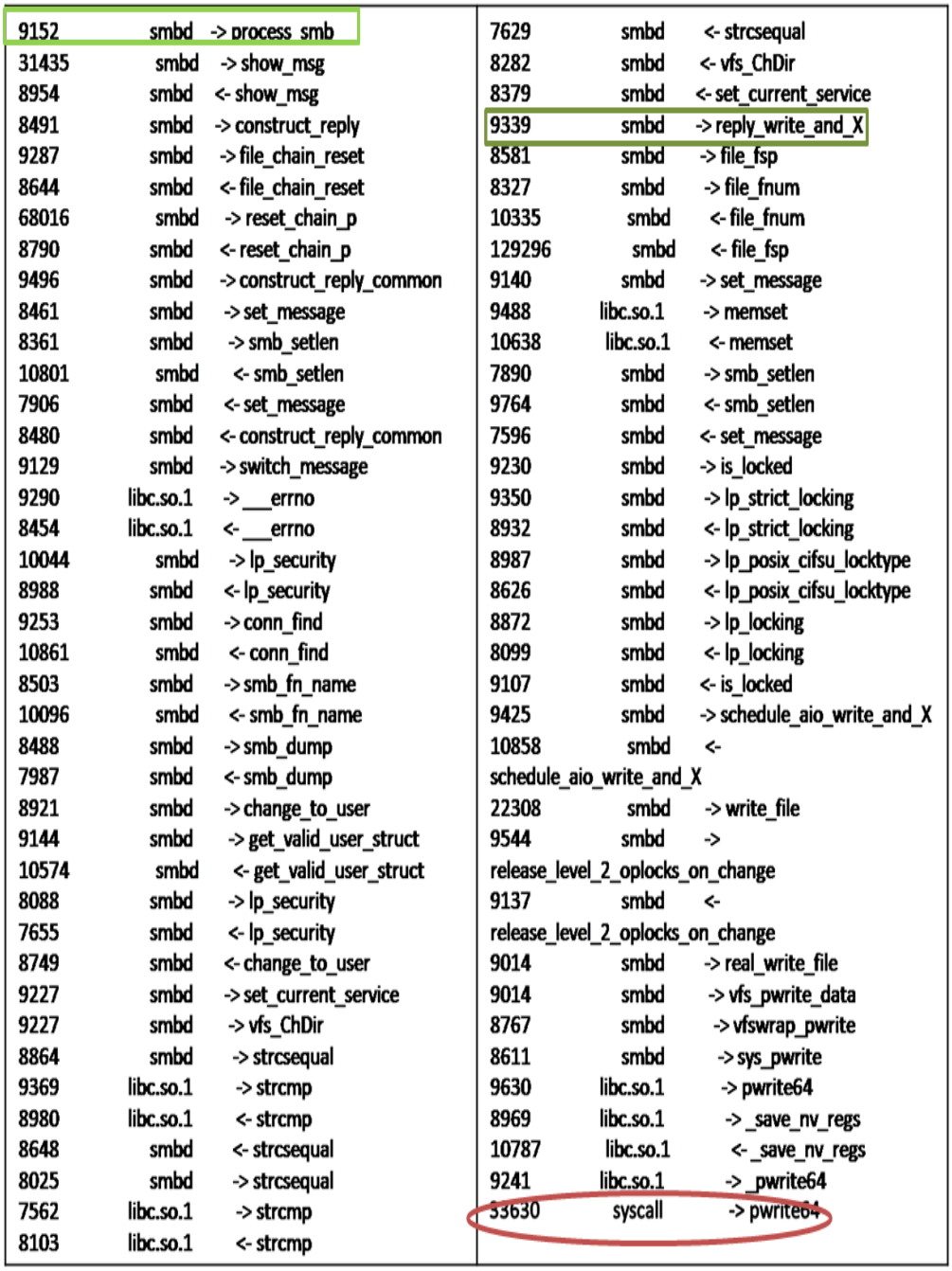
4.5.1 Dtrace 之追蹤資料

我們使用 Dtrace 追蹤到 smbd 於執行時的行為，如圖表 24，可以知道 smbd 用了兩次 *sys_read* 去讀取 socket buffer 的資訊，第一次讀取要寫入的資料長度，讀取後交由 *read_smb_length_return_keepalive* 函式來處理長度的資訊。接著由 *receive_smb* 函式呼叫第二次系統呼叫讀取寫入的資料到 user buffer，得到資料後於圖表 25 交由 *process_smb* 函式來做處理。分析來自 client 端的命令，再根據不同的 client 要求提供不同的方法，若來自 client 的命令為寫入，則交由 *reply_write_and_x* 這函數來處理，接著呼叫 *sys_pwrite64* 將資料從 user buffer 搬入 file system cache，另外於圖表 26 *send_smb* 函式處理回應給客戶端的資料。此即為一次 512b 寫入的 smbd 處理路徑。

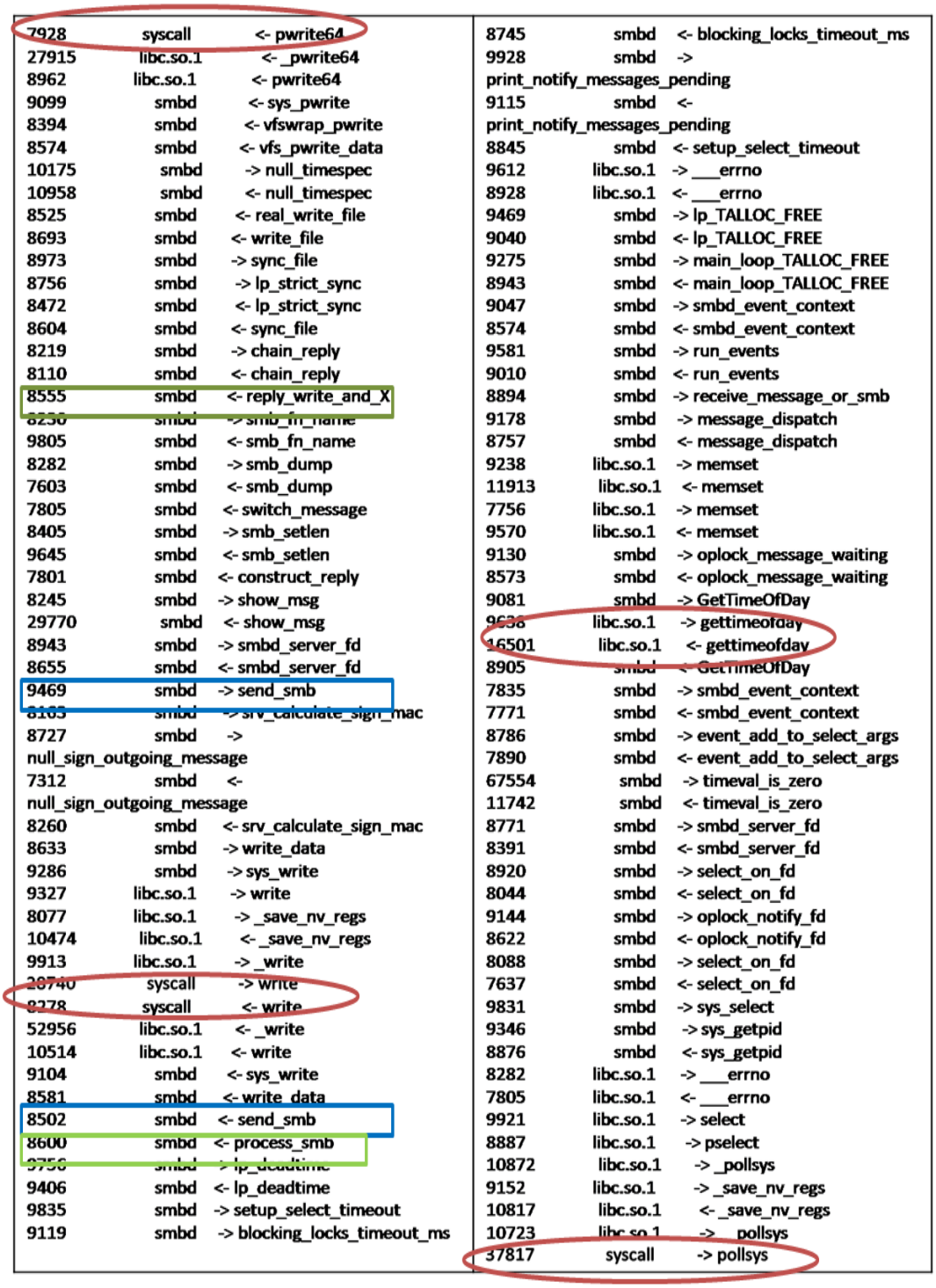
```
9510    smbd    -> receive_smb
9134    smbd    -> receive_smb_raw
8797    smbd    -> read_smb_length_return_keepalive
9223    smbd    -> read_data
8790    smbd    -> sys_read
9391    libc.so.1    -> read
9286    libc.so.1    -> _save_nv_regs
11410   libc.so.1    <- _save_nv_regs
10040   libc.so.1    -> _read
31088   syscall      -> read
7637    syscall      <- read
31819   libc.so.1    <- _read
10850   libc.so.1    <- read
9335    smbd    <- sys_read
8879    smbd    <- read_data
9063    smbd    <- read_smb_length_return_keepalive
9047    smbd    -> read_data
```

```
8835    smbd    -> sys_read
8439    libc.so.1    -> read
8084    libc.so.1    -> _save_nv_regs
10974   libc.so.1    <- _save_nv_regs
9070    libc.so.1    -> _read
30005   syscall      -> read
7778    syscall      <- read
29248   libc.so.1    <- _read
66624   libc.so.1    <- read
8525    smbd    <- sys_read
8122    smbd    <- read_data
8573    smbd    <- receive_smb_raw
11194   smbd    -> srv_check_sign_mac
10719   smbd    -> null_check_incoming_message
12250   smbd    <- null_check_incoming_message
8876    smbd    <- srv_check_sign_mac
8778    smbd    <- receive_smb
```

圖表 24 Dtrace 追蹤 smbd 讀取寫入資料之結果



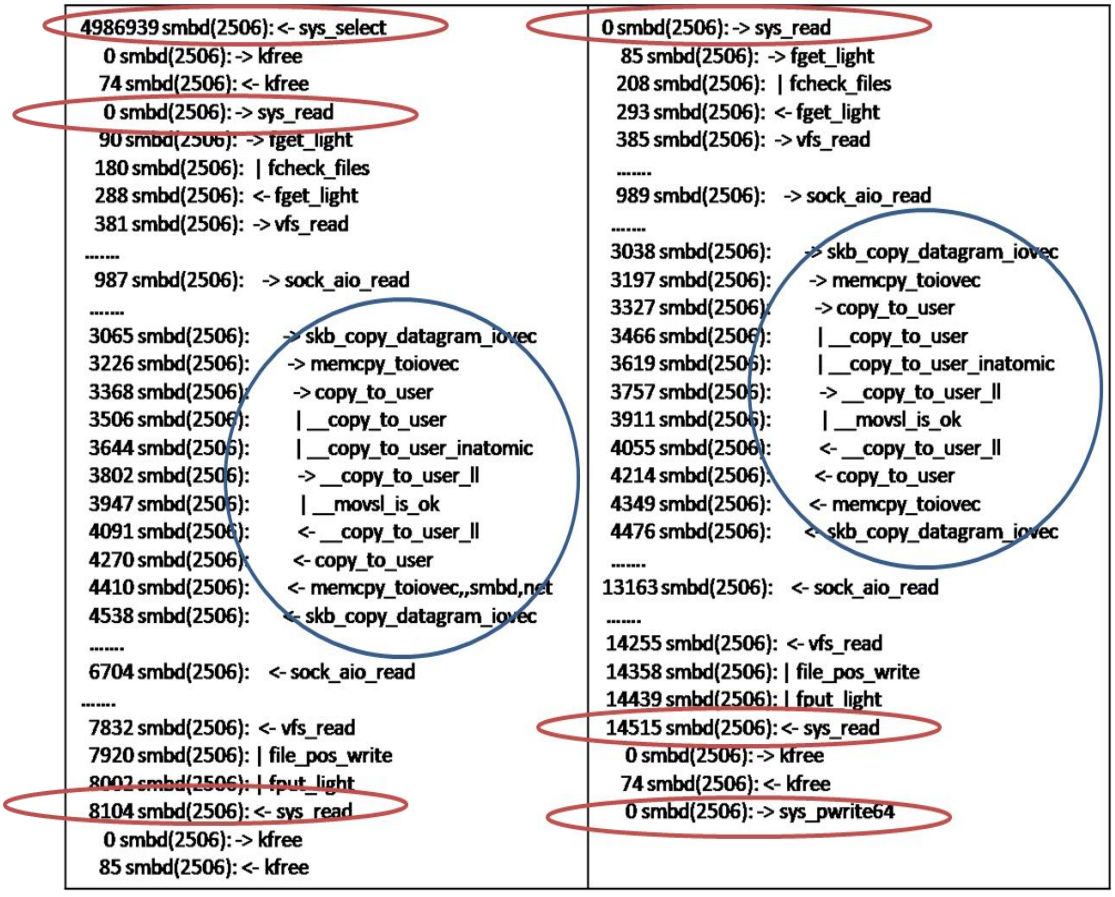
圖表 25 Dtrace 追蹤 smbd 處理寫入資料之結果



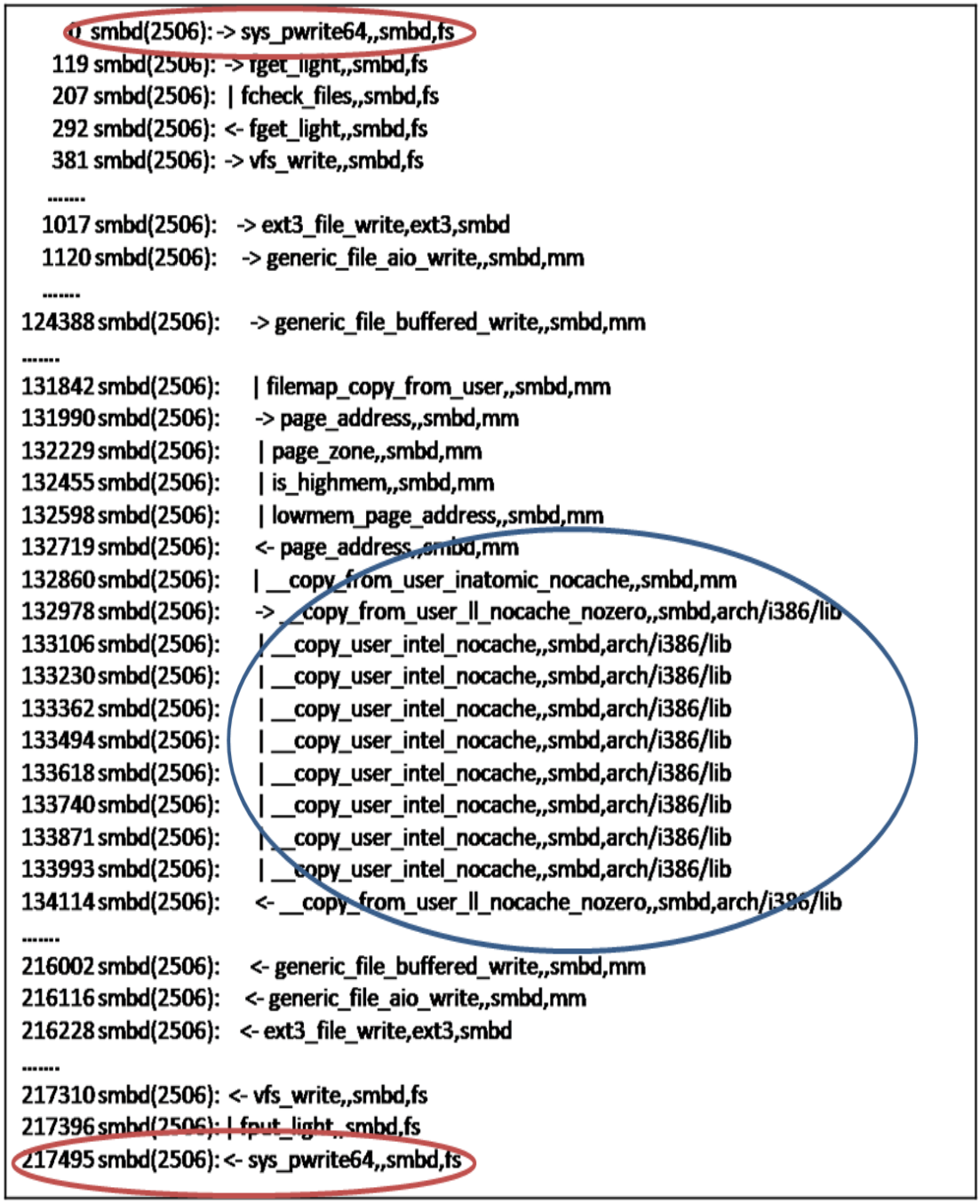
圖表 26 Dtrace 追蹤 smbd 送出回應資料到客戶端之結果

4.5.2 SystemTap 之追蹤資料

接著我們為了能得到 smbd 在系統呼叫下核心的行為，我們利用了 SystemTap 進一步得知 smbd 在系統呼叫下核心的行為，找出資料搬移的路徑。圖表 27 為兩次 *sys_read* 中核心所做的行為，第一次 *sys_read* 複製 4byte 的資料長度到 user buffer，在使用者空間判斷寫入的長度後，第二次將長度大小的資料從 socket buffer 複製到 user buffer，接著處理從客戶端來的寫入命令（圖表 28）。如果為 512byte 的資料寫入命令，則將 512byte 的資料寫入 file system cache，而寫入 file system cache 又多了一次資料的搬移從 user buffer 到 file system cache。



圖表 27 SystemTap 追蹤核心對於 smbd 讀入系統呼叫下之結果



圖表 28 SystemTap 追蹤核心對於 smbd 寫入系統呼叫下之結果

第5章 效能改善機制

根據前一章的探測資料，我們得知每次一個客戶端的要求寫入，都會花費 6 次的系統呼叫，第一次 *sys_read* 讀取資料長度，第二次 *sys_read* 讀取資料，第三次 *sys_pwrite* 寫入 disk buffer，第四次 *sys_write* 回應客戶端，第五次 *sys_gettimeofday* 取得時間資訊，第六次 *sys_select* 等待下一次要求。如果要求寫入的檔案大小較小，每次要求都需要 6 次的 system call，context switch 的增加，造成了 CPU 的負擔。

其次對於要求較大的檔案而言，context switch 造成的影響就沒那麼大，但是資料複製（data copy）卻是另外一個值得探討的問題，每一次的寫入要求 CPU 都必須將資料從 socket buffer 複製到 user buffer，Samba 處理過後再將資料從 user buffer 複製到 disk buffer。然而對於嵌入式 Samba 伺服器而言，這複製的動作是可以省略的，因為資料並不會遭到更動，大量的複製反而造成了 CPU 的負擔。

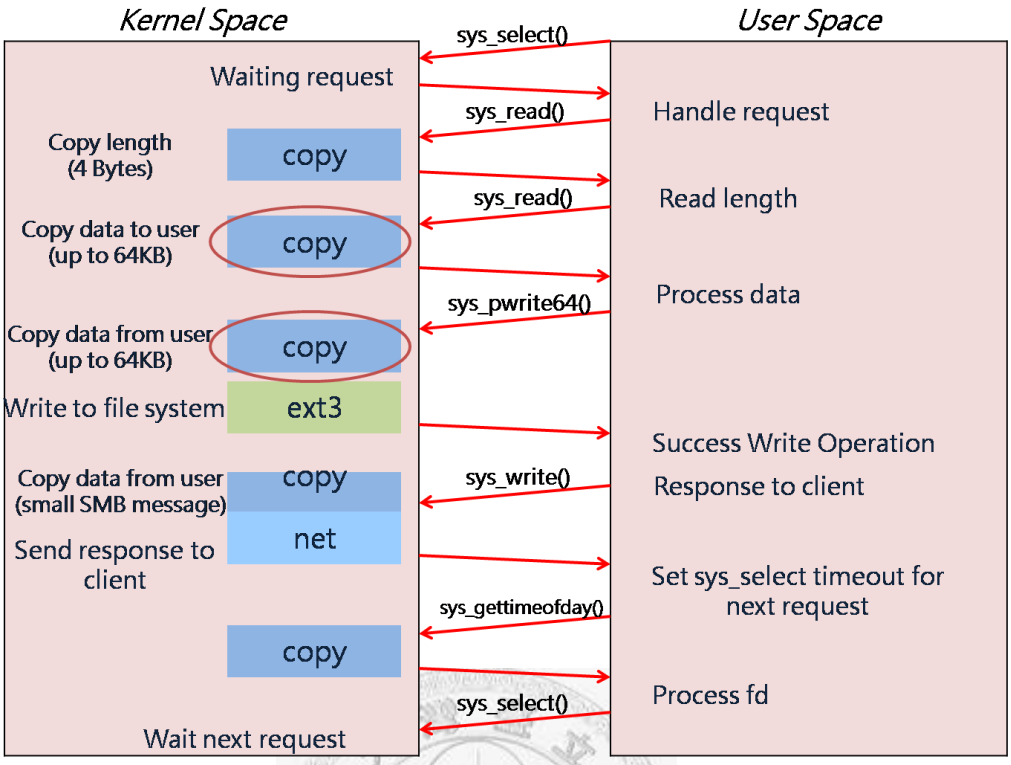
為了解決上述這兩點問題，我們決定將 Samba 處理資料的主要程式遷移到核心空間（kernel space），以達到 zero-copy 解決 context switch 以及資料複製的問題。

5.1 評估遷移路徑

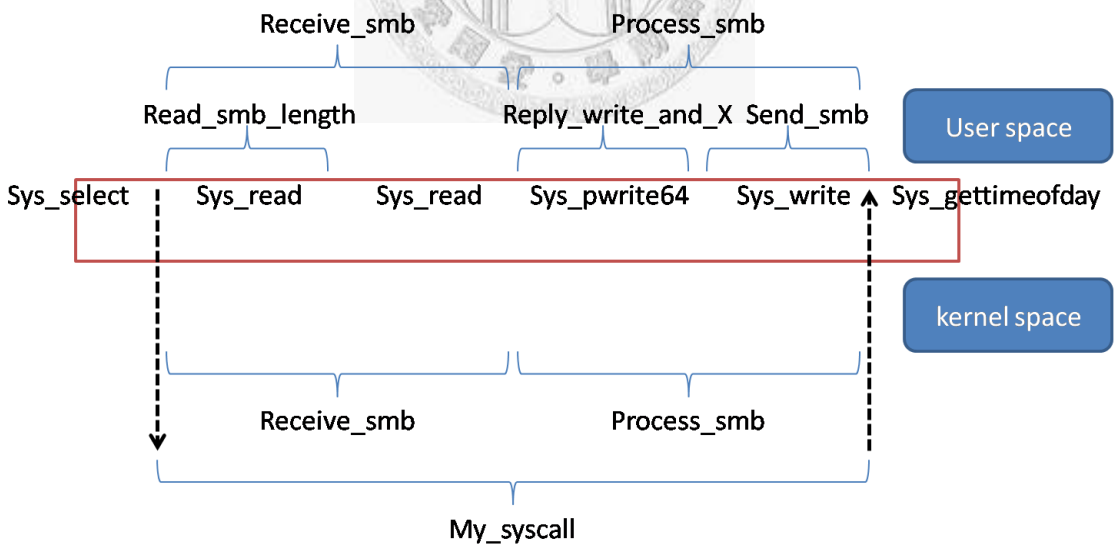
經由上述的工具我們整理出 SMB 寫入流程圖（圖表 29）。Samba 初始化完後，呼叫 *sys_select* 的系統呼叫，等待來自於客戶端的要求；收到要求後，*sys_select* 回傳，Samba 開始接收資料，先呼叫第一個系統呼叫 *sys_read*，從 socket buffer 複製 4byte 的資料到 user buffer，並根據這資料得到資料的長度並加以判斷是否長度大於零；接著進行第二次系統呼叫 *sys_read*，複製經由前一次系統呼叫所取得的長度的資料，因 TCP receive buffer 及 send buffer 的限制，故一次最多處理 64KB 的資料；接著資料複製到 user buffer 後，Samba 進行資料的處理，分析客戶端的命令，並實行不同的運作方式，若客戶端運行寫入，則資料處理完後即呼叫 *sys_pwrite64* 將資料寫入到檔案系統，成功寫入後接著呼叫 *sys_write* 回報訊息給客戶端；接著設定 *sys_select* 的等待時間，進入等待下一個來自客戶端的要求。若客戶端要求寫

入的大小為 1MB 則依照上述的運作，重複 16 次。

為了將 Samba 處理資料的主要程式遷移到核心空間，我們需要知道 Samba 及核心之間相互的行為，因此我們藉由 Dtrace 及 SystemTap 所追蹤到的資料，進一步探討 Samba 寫入的行為。我們將 Samba 與核心間的行為，以系統呼叫為分界，做遷移的評估。Samba 寫入行為中，有兩次系統呼叫為 *sys_read*，讀取 socket buffer 的資料到 user buffer，及 *sys_pwrite64*，將 user buffer 資料寫入到 disk buffer。為了將這兩次的資料複製消除掉，我們必須知道 Samba 是如何處理這些系統呼叫的。因此我們利用 Dtrace 加上 SystemTap 得知 Samba 與核心追蹤的資料，並加以分析，如圖表 30 所示較為重要的兩個函式為 *receive_smb* 與 *process_smb*，這兩個函式就包含了 4 次的系統呼叫，而其中兩次系統呼叫更是我們所關心的資料複製。Samba 中最主要的 user buffer 部分包含於這兩個函式，若我們將這兩個函式遷移到核心，可免除 Samba 與核心間資料同步的問題。於是我們決定創造一個新的系統呼叫將最主要的兩個函式遷移到核心，以減少 6 次的 context switch 以及 2 次的資料複製 。



圖表 29 SMB 寫入之流程圖



圖表 30 遷移流程圖

5.2 評估效能改善的程度

在將 Samba 的主要程式遷移到 kernel space 之前，我們對於減少資料複製，做一個簡易的評估。為了能快速的評估，我們利用 Strace，紀錄所有系統呼叫所花費的時間。由於牽涉資料複製的兩個系統呼叫分別為 *sys_read* 與 *sys_pwrite64*，我們由此概略估計資料複製所佔整個資料處理的比例，再以下列算式評估移除資料複製過後可能的效能改善：

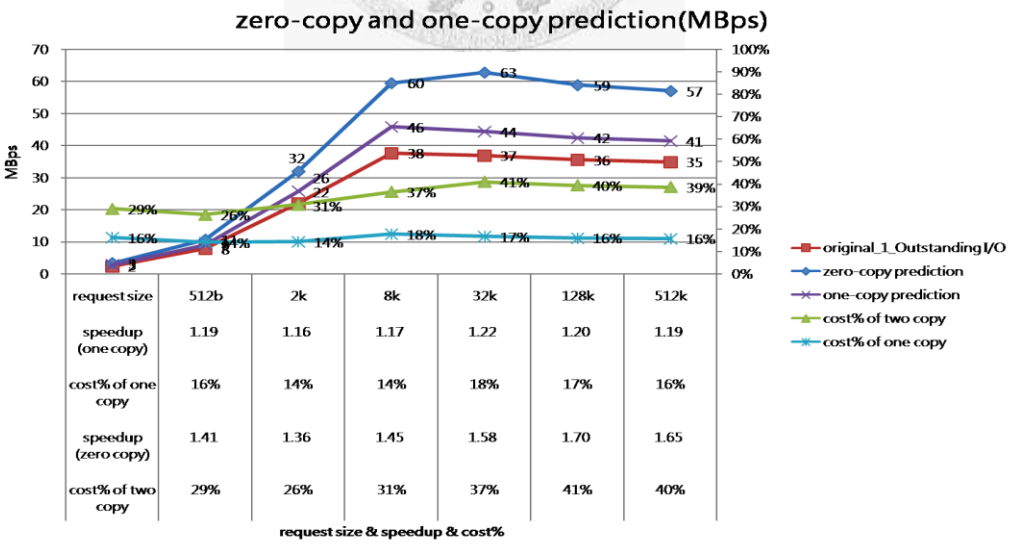
*bandwidth = request size * IOPS = request size / response time*

cost% = cost time / response time

bandwidth_improved = bandwidth_original / （*1 - cost%*）

就小檔案而言，因為大部份的時間並不是花在資料複製的時間，所評估出來的時間為不含資料複製之系統呼叫所花的時間；就大檔案而言，則概略評估系統呼叫的時間為資料複製時間。

由圖表 31 顯示，於大檔案傳輸的情況下，若是利用 zero-copy，最高可達到 ~60MBps 的輸出，將近 65%的效能改善，於是我們開始著手進行遷移。



request size	512b	2k	8k	32k	128k	512k
speedup (one copy)	1.19	1.16	1.17	1.22	1.20	1.19
cost% of one copy	16%	14%	14%	18%	17%	16%
speedup (zero copy)	1.41	1.36	1.45	1.58	1.70	1.65
cost% of two copy	29%	26%	31%	37%	41%	40%

圖表 31 zero-copy 與 one-copy 輸出預測

第6章 實作方式與實驗結果

為了只將 Samba 一部分的程式碼，*process_smb* 與 *recvive_smb* 遷移到核心，我們實作一個新的系統呼叫，作為進入核心及退出核心的介面。但系統呼叫最多只能傳入 6 個參數，無法做過多的傳值呼叫，且過多的傳值呼叫會造成額外的負擔，因此我們將 Samba 於 user space 中 debug 的一些全域變數移除，減少傳入核心的資料。此外系統呼叫需要傳入 fd number，而不同的系統呼叫所需的 fd number 也不同，故在進入我們系統呼叫下一併將 fd number 的位址傳入核心。

因 Samba 使用到 gcc library，所以我們必須將使用到的 gcc library 遷移到核心。gcc library 中針對不同的平台有不同的方法，因方法過多，我們只遷移實驗平台 i386 所用的 library。

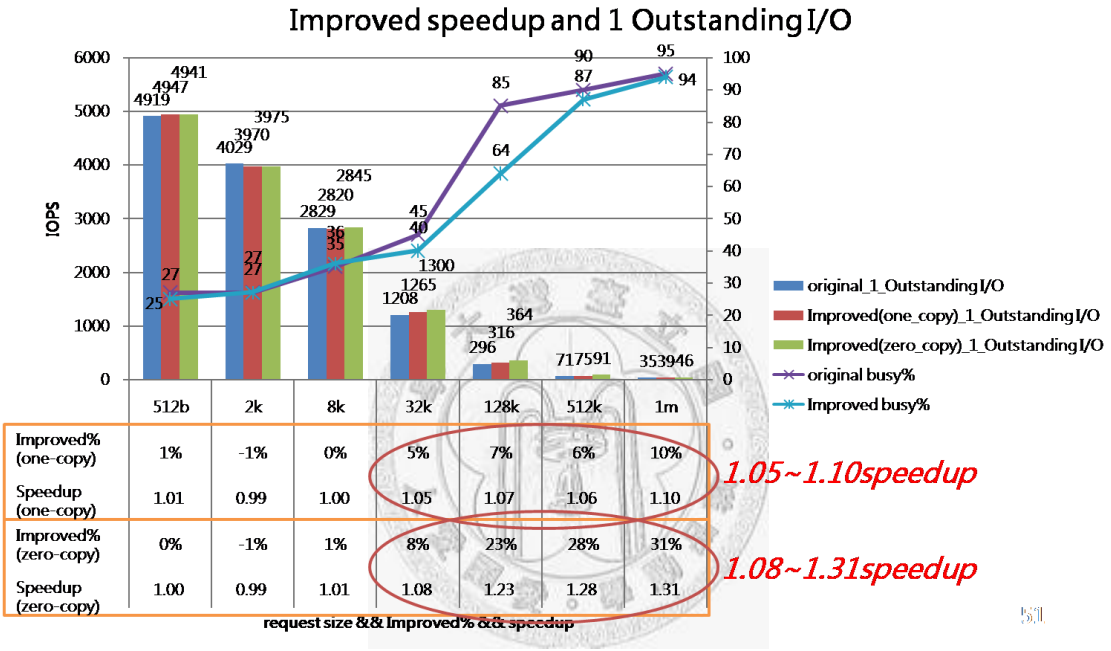
將 Samba 遷移進核心後，還存在者兩次的資料複製，接者我們針對於影響效能較大的資料複製做修改。資料複製於核心中主要由兩個函式控制，*copy_to_user* 及 *copy_from_user*，我們將資料複製改以指標的方式，將 user buffer 及 file system cache 指向 socket buffer。

遷移到核心後，我們利用 Iometer 於客戶端產生不同大小的寫入要求，來進行測試。由於增進效能的部分是以資料複製為主，所以當要求寫入的檔案大小越大，效能才越顯著。接著我們分別以 1 個 outstanding I/O 與 32 個 outstanding I/O 來對於改善後的 Samba service 做測量，並觀察系統中的狀態。

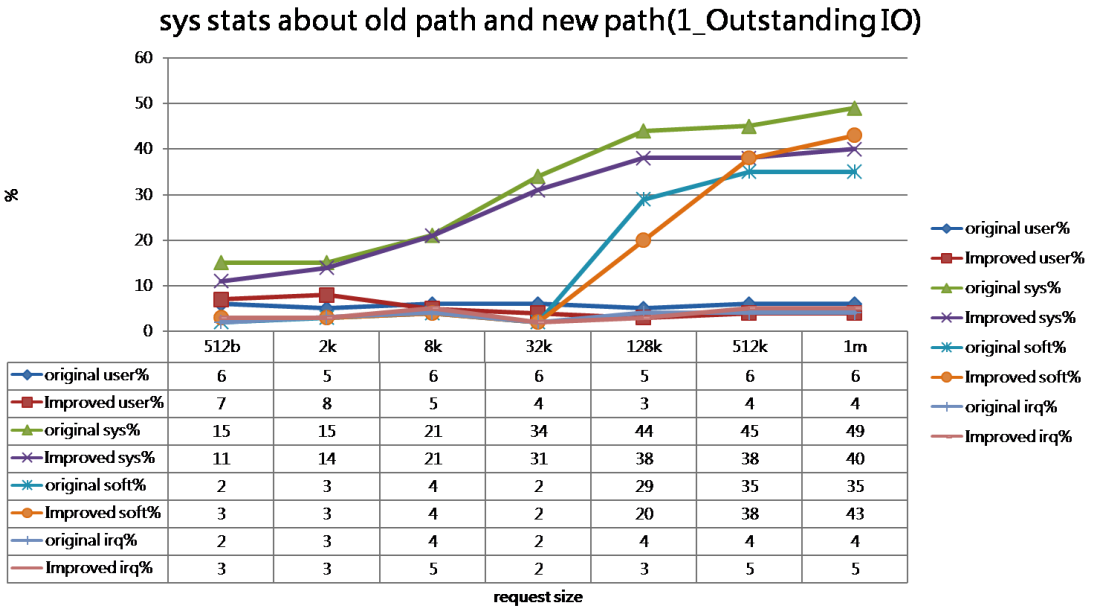
6.1 1 Outstanding I/O 時系統效能的改善

於 1 個 outstanding I/O 的情況下，我們看到於 one-copy 的情況下 32KB 的 request file size 到 1MB 的 request file size 我們可以有 1.05~1.10 的 speedup，而 zero-copy 則可達到 1.08~1.31 的 speedup，接著我們再看到改善前與改善後的 CPU utilization，結果我們看到只有在 128KB 時 CPU utilization 有明顯的下降，大於 128KB 的 request file size 卻沒有明顯的減少，於是我們進一步分析改善後系統資源

分布的狀況，於圖表 33 我們得知，就 sys 所佔的時間而言，隨著 request file size 越大所降低的比例也越多，zero-copy 確實減少了 CPU 的負擔，但 CPU utilization 卻沒有隨 request file size 越大而越少，於圖表 33 得知 softirq 經過我們改善後反而比改善前花費了較多時間，因而造成了 CPU utilization 沒有如預期的隨 request file size 越大而越少。但就 sys 所花的時間來看，我們成功的將 sys 所花的時間從 49% 降低到了 40%。有效降低了 CPU 於資料複製的負擔。



圖表 32 one-copy 與 zero-copy 的 speedup（1 outstanding I/O）



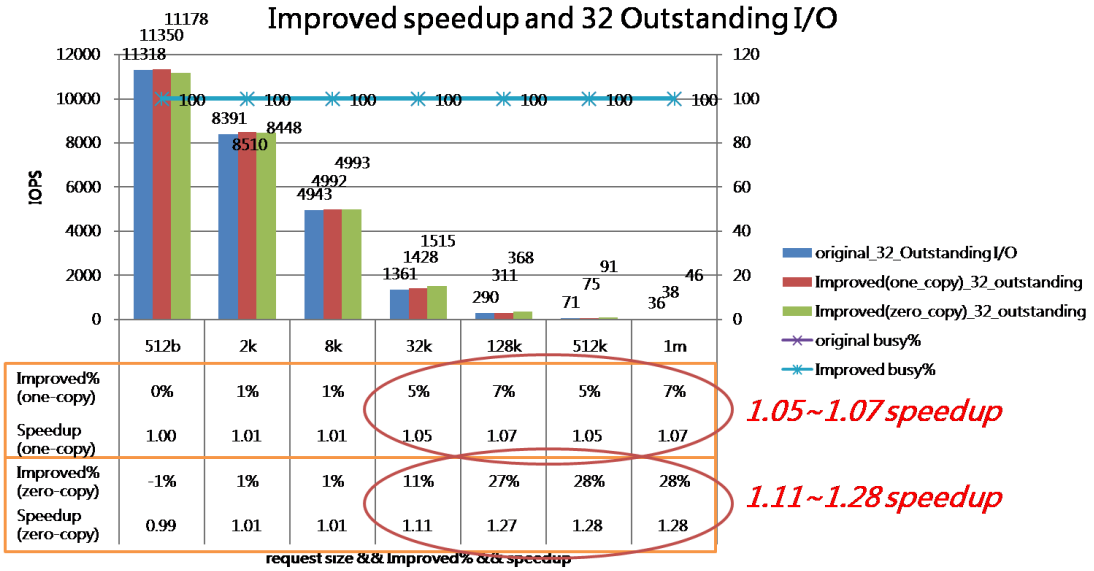
	512b	2k	8k	32k	128k	512k	1m
original user%	6	5	6	6	5	6	6
Improved user%	7	8	5	4	3	4	4
original sys%	15	15	21	34	44	45	49
Improved sys%	11	14	21	31	38	38	40
original soft%	2	3	4	2	29	35	35
Improved soft%	3	3	4	2	20	38	43
original irq%	2	3	4	2	4	4	4
Improved irq%	3	3	5	2	3	5	5

圖表 33 改善前後之系統狀態（1 outstanding I/O）

6.2 伺服器滿載的情況系統效能的改善

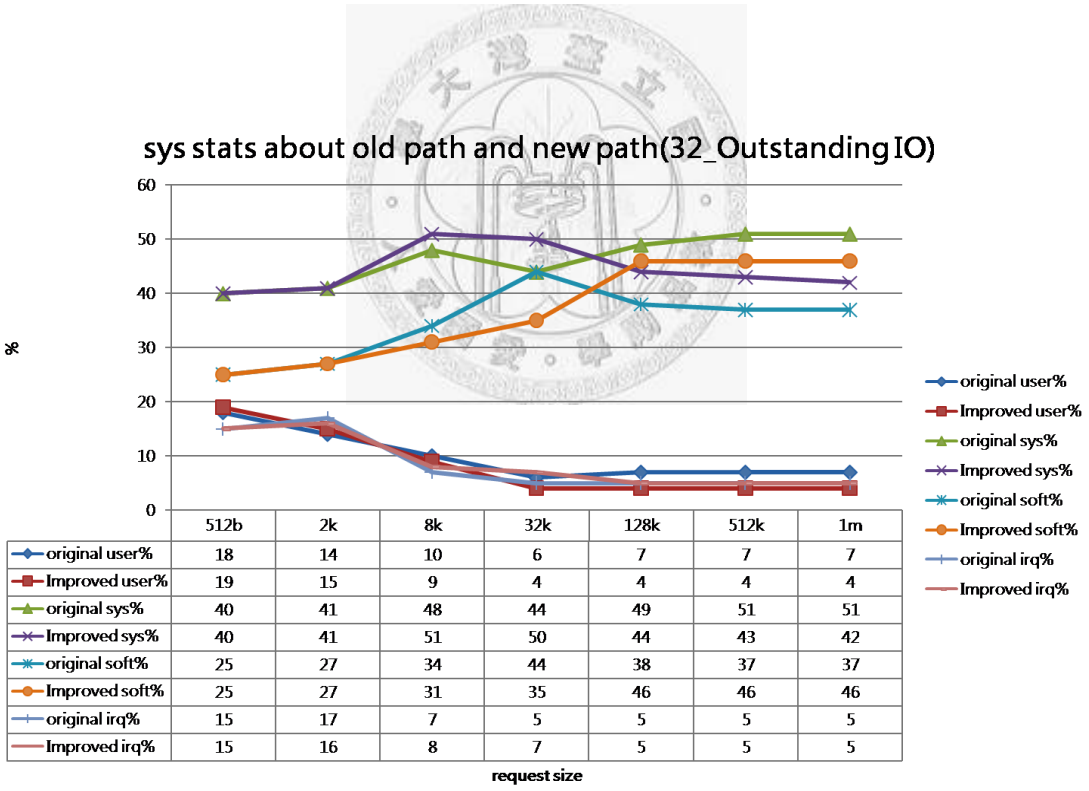
接著由圖表 34 我們得知在 CPU 滿載的情況下，因所處理的資料量增加，如有多餘的 CPU 資源可用，輸出也可相對的提高。由於我們改善了資料複製使得 CPU 使用率於 1 個 outstanding I/O 下較原本的低，於 128KB 可明顯的看到 CPU 資源完全利用的時候改善的幅度從原本的 23%上升到了 27%，而 1MB 的情況因為原本於 1 個 outstanding I/O 的情況下就已沒餘下多少 CPU 資源，故於 32 outstanding I/Os 的情況下沒辦法有明顯的成長。從 32KB request file size 到 1MB request file size，one-copy 可以有 1.05~1.07 的 speedup，而 zero-copy 更可達到 1.11~1.28 的 speedup。

接著我們再進一步分析伺服器滿載的情況下資源的分配情形，於圖表 35 可看出於較大的 request file size 下，我們成功將 sys 所花的時間從 51%降低到了 42%，有效降低了 CPU 負荷滿載情況下資料複製的負擔。並發現 softirq 與效能成正比的一個關係。



	512b	2k	8k	32k	128k	512k	1m
Improved% (one-copy)	0%	1%	1%	5%	7%	5%	7%
Speedup (one-copy)	1.00	1.01	1.01	1.05	1.07	1.05	1.07
Improved% (zero-copy)	-1%	1%	1%	11%	27%	28%	28%
Speedup (zero-copy)	0.99	1.01	1.01	1.11	1.27	1.28	1.28

圖表 34 one-copy 與 zero-copy 的 speedup（CPU 滿載的情況）



	512b	2k	8k	32k	128k	512k	1m
original user%	18	14	10	6	7	7	7
Improved user%	19	15	9	4	4	4	4
original sys%	40	41	48	44	49	51	51
Improved sys%	40	41	51	50	44	43	42
original soft%	25	27	34	44	38	37	37
Improved soft%	25	27	31	35	46	46	46
original irq%	15	17	7	5	5	5	5
Improved irq%	15	16	8	7	5	5	5

圖表 35 改善前後之系統狀態（CPU 滿載的情況）

第7章 預測加入 TOE 後的效能

由之前的測試得知 CPU 所花費的時間中有 80%都花費在系統上，也就是說扣除資料複製所佔的時間，還有 50%的時間是我們值得去探討的，根據相關的研究顯示，於處理 TCP 網路通訊協定的部分，也花費了系統不少時間。因此我們針對網路的部分來進行評估。

為了就網路處理的部分深入評估，我們先對核心中 net 的部分插入 probe，發現處理網路部分最上層的函式為 *net_rx_action*，故我們針對於 *net_rx_action* 函式的進入點與退出點分別插入 probe，加入時間的資訊，並加以收集（圖表 36）（圖表 37）。接著評估於 512B 到 1MB 於網路處理所花的時間。以下為我們所評估的主要數學式，由 Iometer 我們可以得知 bandwidth、response time、IOPS，其中 IOPS 又與 response time 成反比，因此我們求出網路所花的時間後，再與 response time 相除得到網路所佔的比例（cost%），接著原本的 bandwidth 再除以改善後所需花費的時間比（1-cost%），所得的結果即為加入 TOE 後所能達到的 bandwidth，由以下的算式做出預測：

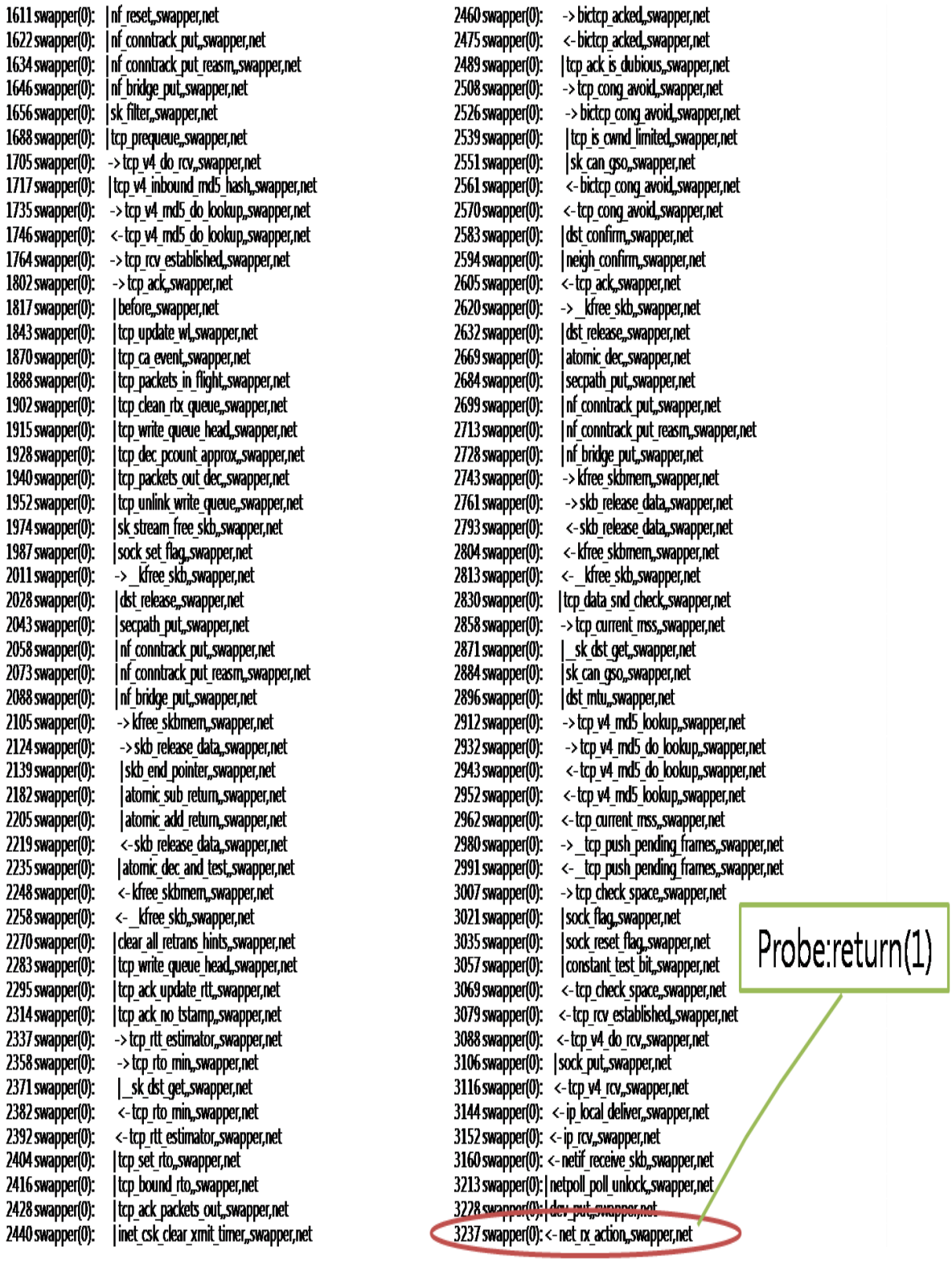
*bandwidth = request size * IOPS = request size / response time*

network cost% = cost of netwrok / response time

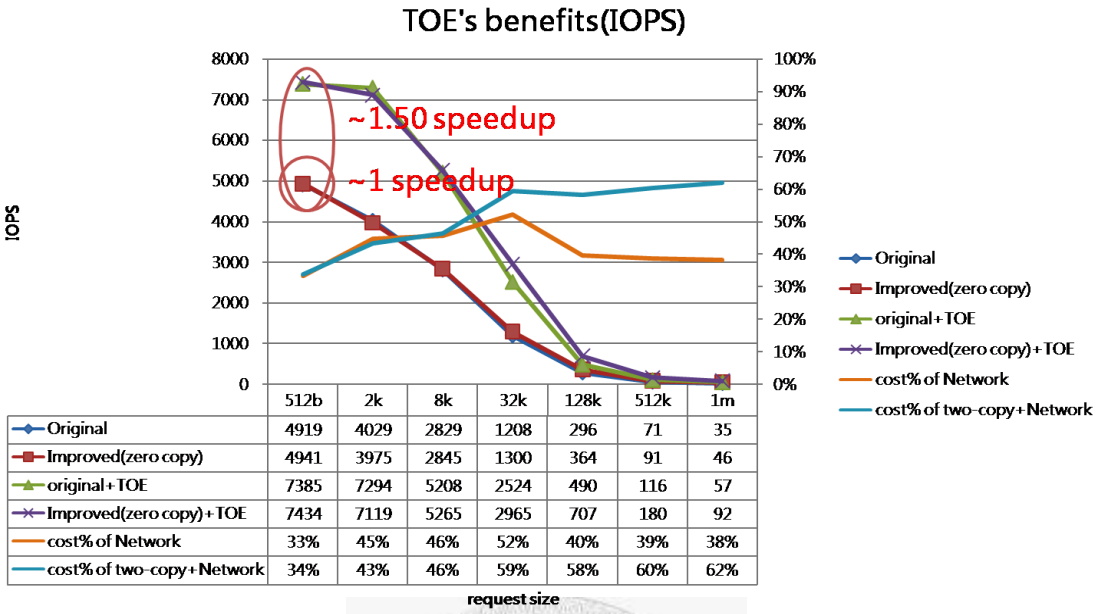
bandwidth with TOE = bandwidth_original /（1 - network cost%）

根據這預測，我們假設 TOE 能解決所有網路處理的時間，並加以評估。於圖表 38 可看到 zero-copy 對於較小的 request file size 而言並沒有多大的改善，因此預測若加上 TOE 將只有 1.5 的 speedup。另外圖表 39 可看到單就只有 zero-copy 只能達到 1.31 的 speedup，且單只有 TOE 的話也只能達到 1.62 的 speedup，但若兩者整合 TOE 加上 zero-copy 的話，則可達到將近 92MBps 的頻寬，改進為原本的 2.62 倍。

圖表 36 網路處理之追蹤資料

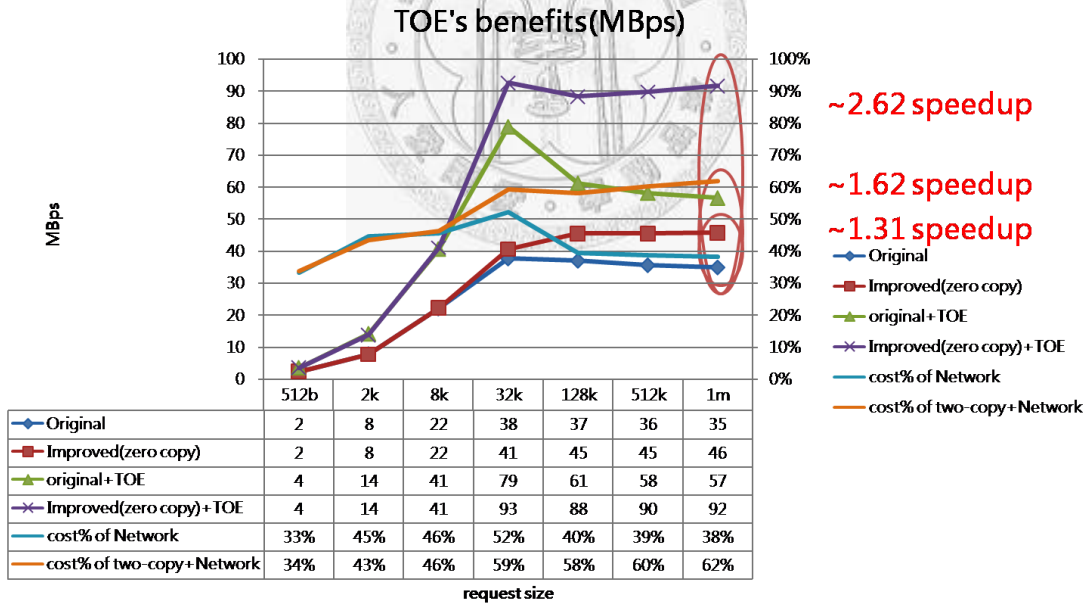


圖表 37 網路處理之追蹤資料



	512b	2k	8k	32k	128k	512k	1m
Original	4919	4029	2829	1208	296	71	35
Improved(zero copy)	4941	3975	2845	1300	364	91	46
original+TOE	7385	7294	5208	2524	490	116	57
Improved(zero copy)+TOE	7434	7119	5265	2965	707	180	92
cost% of Network	33%	45%	46%	52%	40%	39%	38%
cost% of two-copy+Network	34%	43%	46%	59%	58%	60%	62%

圖表 38 預測 TOE 的理想 IOPS



	512b	2k	8k	32k	128k	512k	1m
Original	2	8	22	38	37	36	35
Improved(zero copy)	2	8	22	41	45	45	46
original+TOE	4	14	41	79	61	58	57
Improved(zero copy)+TOE	4	14	41	93	88	90	92
cost% of Network	33%	45%	46%	52%	40%	39%	38%
cost% of two-copy+Network	34%	43%	46%	59%	58%	60%	62%

圖表 39 預測 TOE 的理想頻寬

第8章 結論和未來展望

為了達到高效能低成本的嵌入式網路儲存系統，必須就整個系統做全面設計與測試以找出系統中的瓶頸，並加以評估硬體的設計應使用多少的資源才能達到最好的成本效益（cost-performance）。我們針對 Samba 網路儲存伺服器，提供一套完整的測試及設計流程，完整追蹤出 Samba 於處理資料時，函式間的相互關係。我們追蹤包含系統呼叫下核心函式的完整路徑，以幫助設計嵌入式儲存系統時軟硬體的整合;我們測量出資料複製於較大的 request file size 下佔了系統約 25%的時間，接著利用追蹤到的完整路徑將 Samba 處理資料的部分遷移到核心，以達成 zero-copy，及減少 context switch。其中較大的檔案有將近 1.31 的效能改善，有效的減輕了 CPU 的負擔。

此外就系統所花費的另 70%時間裡，經過我們的測量，結果顯示約有 40%左右的時間是花在處理網路部分，接著進一步評估，若 zero-copy 加上 TOE 將可達到約 2.62 的效能改善，因此未來若能將網路處理部分以 FPGA 上自行設計的 offload engine 來處理，並加上軟體部分的 zero-copy，即可達成一個高效能且低成本的網路儲存式伺服器。

參考文獻

[1] Fall, K.R., and Pasquale, J., "Exploiting in-Kernel Data Paths to Improve I/O Throughput and Cpu Availability", in *Proceedings of the 1993 USENIX Winter Technical Conference*, 1993, pp. 327-334.

[2] Hewlett-Packard Company, "Linux Programmer's Manual - Sendfile", http://devresource.hp.com/STKL/man/RH6.1/sendfile_2.html, 2001.

[3] Buddhikot, M., "Project Mars: Scalable, High Performance, Web Based Multimedia-Ondemand (Mod) Services and Servers", in *Departmetn of Computer Science, Washington University, St. Louis, MO, USA*, 1998

[4] Kim, H.-y., and Rixner, S., "Tcp Offload through Connection Handoff", in *European Conference on Computer Systems*, 2006, pp. 279-290.

[5] Dalessandro, D., Wyckoff, P., and Montry, G., "Initial Performance Evaluation of the Neteffect 10 Gigabit Iwarp Adapter", in *Cluster Computing, 2006 IEEE International Conference*, 2006, pp. 1-7.

[6] Balaji, P., Jin, H.W., Vaidyanathan, K., and Panda, D.K., "Supporting Iwarp Compatibility and Features for Regular Network Adapters", in *Cluster Computing, 2005. IEEE International*, 2005, pp. 1-10.

[7] Dalessandro, D., Devulapalli, A., and Wyckoff, P., "Iwarp Protocol Kernel Space Software Implementation", in *Parallel and Distributed Processing Symposium, 2006. IPDPS 2006. 20th International*, 2006, pp. 8 pp.

[8] Engel, J., Meneskie, J., and Kocak, T., "Performance Analysis of Network Protocol Offload in a Simulation Environment", in *Atlantic Coast Marketing SE*, 2006, pp. 762-763.

[9] Halvorsen, P., Jorde, E., Skevik, K.A., Goebel, V., and Plagemann, T., "Performance Tradeoffs for Static Allocation of Zero-Copy Buffers", in *Proceedings of 28th Euromicro Conference*, 2002, pp. 138-143.

[10] Kang, D.-J., Kim, Y.-H., Cha, G.-I., Jung, S.-I., Kim, M.-J., and Bae, H.-Y., "Design and Implementation of Zero-Copy Path for Efficient File Transmission", *High Performance Computing and Communications*, vol. 4208/2006, 2006.

[11] J. Tranter. Exploring the sendfile system call. http://ldp.dvo.ru/LDP/LG/issue91/tranter.html

[12] Andrew Tridgwell. Samba resources. http://us1.samba.org/samba/, 1992.

[13] Tom's Guide web resources. http://www.tomsguide.com/us/dlink-medialounge-dsm-g600-wireless-g-network-st

orage-enclosure,review-676-9.html
[14] Senapathi, S., and Hernandez, R., Introduction of Tcp Offload Engines (Dell Power Solution, 2004), pp. 103-107
[15] Gupta, P., Light, A., and Hameroff, I., Boosting Data Transfer with Tcp Offload Engine Technology on Ninth-Generation Dell Poweredge Servers (Dell Power Solutions, 2006), pp. 18 - 22
[16] Tianhua, L., Hongfeng, Z., Guira, C., and Chuansheng, Z., "Research and Implementation of Zero-Copy Technology in Linux", in *Sarnoff Symposium, 2006 IEEE*, 2006, pp. 1-4.
[17] InfiniBand Trade Association, "Infiniband Architecture Specification", 2004.
[18] Borkar, S., Cohn, R., Cox, G., Gleason, S., Gross, T., Kung, H.T., Lam, M., Moore, B., Peterson, C., Pieper, J., Rankin, L., Tseng, P.S., Sutton, J., Urbanski, J., and Webb, J., "Iwarp: An Integrated Solution to High-Speed Parallel Computing", in *Supercomputing '88. [Vol.1]. Proceedings.*, 1988, pp. 330-339.
[19] Wu, Z.-Z., Chen, H.-C., and Huang, C.-M., "The 10gbit Hba Hardware Design for Iwarp Offloading Engine", *CCL TECHNICAL JOURNAL*, 2005.
[20] CR, H., Implementing Cifs: The Common Internet File System (2004)
[21] Samba resources. http://us6.samba.org/samba/
[22] Wang, C.W., Performance Optimization of the Samba Read Service on Linux-Based Network-Attached Storage Systems (2008)
[23] Dong-Jae, K., Chei-Yol, K., Kang-Ho, K., and Sung-In, J., "Design and Implementation of Kernel S/W for Tcp/Ip Offload Engine(Toe)", in *Advanced Communication Technology, 2005, ICACT 2005. The 7th International Conference on*, 2005, pp. 706-709.
[24] Intel Corporation. IOmeter resources. http://www.iometer.org/, 1998.
[25] Sysstat resources. http://pagesperso-orange.fr/sebastien.godard/index.html
[26] Intel. VTune Performance Analyzer resources. http://www.intel.com/cd/software/products/asmo-na/eng/239144.htm
[27] Strace resources. http://linux.die.net/man/1/strace
[28] Valgrind resources. http://valgrind.org/
[29] IBM. Pvtrace resources. http://www.ibm.com/developerworks/library/l-graphvis/

[30] Graphviz resources.
http://www.graphviz.org/
[31] Sun. Dtrace resources.
http://www.sun.com/bigadmin/content/dtrace/
[32] Red Hat, IBM, Intel, and Hitachi. SystemTap resources.
http://sourceware.org/systemtap/