適用於物聯網的記憶體式資料處理技術
In-memory Data Processing for IoT

林宏軒　張鎮庭　徐國峰　闕志克　陳玉倫
Hung-Hsuan Lin, Jhen-Ting Jhang, Kuo-Feng Hsu, Tzi-Cker Chiueh, Yu-Lun Chen

中文摘要

近年來物聯網(Internet of Things, IoT)興起，IDC市場調查發現2014年物聯網的市場規模已達到6,558億美元，並預計至2020年將擴大到1兆7,000億美元[1]。在物聯網中所有智能裝置都可以上網，因此所帶來的訊息總量非常大，但每個東西所傳送的訊息卻很小，因此需要新技術快速地處理或儲存這些大量的小訊息。本論文基於批次處理循序送出（BOSC）技術，結合記憶體式處理（in-memory processing）技術、快速封包處理技術（fast packet processing）與遠端機器儲存日誌（logging）的方式，提供快速接收並處理物聯網訊息的方法，並以硬碟效能最大化的方式批次循序更新至硬碟。

Abstract

In recent years, the IoT (Internet of Things) rises. IDC market survey showed that the IoT related market in 2014 reached 655.8 billion US dollars, and is expected to rise to 1,700 billion in 2020 [1]. In IoT, all smart things can connect to the Internet and bring about large amount messages – and the size of each message is very small. Therefore new techniques are needed to process and store the large amount of small messages. This paper, based on Batch mOdifications with Sequential Commit (BOSC) technique, combined with in-memory processing, fast packet processing, and remote logging techniques, provides a solution for fast receiving and handling messages, and then flushing messages with optimal hard disk performance.

關鍵詞(Key Words)

批次更新循序送出(Batch mOdifications with Sequential Commit；BOSC)
記憶體式處理(In-memory Processing)
快速封包處理技術(Fast Packet Processing)
物聯網(Internet of Things；IoT)

1．前言

在物聯網裡，所有設備的裝置都可以上網，或是裝置與裝置之間可以透過訊號方式彼此溝通。可以上網的裝置如穿戴式手環，手環會定時偵測人體健康資訊，並將所得資訊透過網路的方式傳到主機儲存，使用者可以隨時登入並查看人體健康相關資訊。裝置與裝置之間溝通的例子如e-tag，使用者的車子裝設e-tag，高速公路上架設門架，當車子經過時門架可以讀取e-tag上的訊息達到偵測車輛通過的目的。

而隨著愈來愈多的裝置可以上網，伺服器端在同樣的時間內將必須處理比以往更多的資料，因此伺服器的處理效能就顯得更為重要。

幸運地是，由於記憶體的價格降低，記憶體式運算逐漸變為主流。記憶體的存取速度比硬碟的存取速度快了一個數量級，因此在記憶體內存取資料能比傳統在硬碟上存取資料有更高的效率。而當記憶體運算變為主流的時候，會有三個問題需要克服：

- 對硬碟的存取方式
- 對網路的存取方式
- 日誌(log)更新方式

第一個問題是由於目前記憶體的揮發特性無法永久保存資料。當資料只存在於記憶體不存在於硬碟時，若遇到特殊狀況如斷電的時候資料會揮發，因此資料最終要寫進如硬碟的非揮發性裝置。而由於硬碟的存取速度比記憶體存取速度慢很多，因此必須配合硬碟的特性，採用硬碟能最大化效率的方式來將資料寫進硬碟。本論文透過批次更新循環送出技術(BOSC)[2][3]，利用記憶體作為是一種快取儲存，將要對硬碟所做的更新暫存於記憶體中，再以循序批次的方式將記憶體中的更新寫進硬碟。在這樣的更新方式下對硬碟的操作都是循序的，因此可以用最大化硬碟的效率將資料寫進硬碟。

第二個問題是由於物聯網時代很多裝置都可以連網，但傳輸的資料通常是非常小的。比如說穿戴式手環偵測人體心跳，將時間戳記、使用者、心跳數的資料往伺服器傳送，這樣的資料大小通常只需要幾十個位元組(byte)。當這樣的資訊傳送給伺服器的時候，伺服器收到的將會是大量但小型的封包。由於每個封包的處理在不管大或小的情況下都有基本固定的負擔(overhead)，在以往封包較大的狀況下這種基本負擔所造成的影響相對較小。而在物聯網中這種大量小封包的環境下這種基本負擔相對影響較大，因此需要解決這種基本負擔才能提升封包處理效能。本論文透過結合mTCP[4]與DPDK[5]的方式，採用epoll的架構，來解決此問題。

第三個日誌的處理主要是為了解決伺服器的資料持續性(durability)問題。當資料儲存於記憶體而尚未更新至硬碟的期間，若有任何意外狀況發生導致程式重啟，則需要有日誌來提供其所需還原的資訊。而日誌的記錄若儲存於記憶體則一樣會有資料在斷電後遺失的困擾，若記在硬碟則仍會限制住伺服器處理資料的速度。本論文透過網路傳輸，將日誌寫於本機與遠端機器的記憶體，不管哪一台機器有毀損的情況發生，都可以透過另一台的日誌記錄還原資料，來保證資料的持續性。

在本論文中，我們將針對上述三點問題及其解決方法作討論，並量測效能。本論文的其餘章節分配如下：第二章描述相關的研究背景，第三章講解我們的演算法及架構，第四章為實驗及數據，在第五章給出本篇論文的結論。

2 ．研究背景

本論文基於批次更新循序送出技術[2][3]，提供了將資料快速更新至硬碟的能力；結合mTCP[4]與DPDK[5]技術，提供了快速的網路封包處理技術。此章節將介紹這兩種技術。

2.1 批次更新循序送出

批次更新循序送出[2][3]提供了一個更新感知硬碟存取介面(update-aware disk access interface)，以區塊為單位對硬碟做存取。對硬碟區塊所需要做的更新會被重新排程，這些更新會被暫存起來，等到累積了多筆更新後再以批次的方式將這些更新寫入硬碟。更新硬碟的步驟包括：

1. 讀取硬碟區塊
2. 對此硬碟區塊做更新
3. 寫入硬碟區塊

由於區塊資料存於硬碟的實體位置為循序的，因此第1步驟及第3步驟對硬碟讀取或寫入區塊的時候可以用循序的方式來存取，保證了效能。而第2步驟由於更新是在記憶體內完成，因此效能非常快。其中，由於更新具有時間順序，因此在將更新寫至區塊時，會依照發生的時間順序做更新，來保證資料正確性。整個過程由於對硬碟的操作都是循序完成，因此可以擁有很高的硬碟每秒有效更新次數。

除了批次更新的特性之外，[2][3]也提供了

資料的持續性(durability)。而為了提供此特性，效能將會被限制在寫入日誌(log)的速度，因此採用低延遲(latency)[6]的技巧將日誌寫入硬碟來降低此效能瓶頸。BOSC採用一個獨立的背景執行緒(thread)來做日誌寫入的動作，該執行緒會等待一小段的時間，取得這段時間要寫入的多筆日誌，將多筆日誌以一次性的方式將日誌寫入硬碟。

2.2 mTCP

mTCP是一個可擴展的用戶級別(user-level)傳輸控制協議(TCP)，可以很容易地實作在軟體應用(application)層，因為它提供了與傳統一樣的網路端口(socket)介面，並且可以不用修改到內核(kernel)。在mTCP的論文[4]中提到了幾個傳統傳輸控制協議的幾個效率低落的地方：

●缺乏網路連接的區域性

在多核心的系統中，會使用多個執行緒來增加系統整體效能，但這些執行緒必須分享同一個監聽的網路端口，因此導致系統效能降低。

●共享文件描述符(file descriptor)空間

在一個伺服器擁有很多連線的情況下，多個執行緒在共用一個檔案描述符空間時所需存取的鎖(lock)會影響效能。

●低效率的單一封包處理機制

當處理小封包的時候，每次封包存取時的負擔(overhead)的影響相對增加。這些負擔包括記憶體的空間分配、記憶體的存取、較大的資料結構所造成的負擔等。

●沉重的系統調用(system call)負擔

此負擔是來自於每次切換於用戶級別(user-level)與系統級別(system-level)之間，尤其在物聯網都是小封包的時候，這種頻繁的切換會帶來相對較大的負擔以致於拖慢系統效能。

該論文作者亦宣稱大部份的先前研究都需要更改內核來改善封包處理的效能問題，也因此較不適用於多數的應用程式，比如說MegaPipe[7]有考慮了上述幾項問題但需要改動內核，而且在中央處理器(CPU)上的使用率也不佳。

相對而言，mTCP架構在用戶級別中的封包處理技術，解決了上述的所有效能問題，包括：

●如Affinity-Accept[8]與MegaPipe[7]，每一個中央處理器核心都有一個專屬對應的網路端口，並實作負載平衡，讓接收端有多個網路接收端口可以平分負載。

●採用用戶級別的封包處理程式庫，如DPDK[5]與PSIO[7]，來減少低效能封包處理所帶來的影響。

●採用用戶級別的封包處理技術來降低來回切換於系統級別的負擔，並實作封包層與端口層的批次處理技術以減少上下文切換(context switch)的負擔。

同時，由於物聯網時代會有大量的同時連接數(concurrent connections)，mTCP採用了epoll架構，讓中央處理器一直輪詢並處理事件。假設一個程式採用了mTCP，那麼在一台擁有M個核心的機器上，程式在初始時會設定一個數值N (N ≤ M)，表示希望程式與mTCP內部分別產生N個執行緒，每一個程式內部的執行緒都會與一個mTCP內部的執行緒一同綁定執行在一個機器的核心上，藉此降低上下文切換的負擔，讓中央處理器可以有效率地做事而不是頻繁地在執行緒之間切換。

由於已經採用了epoll架構，並實作了平行處理與負載平衡機制，因此mTCP不提供執行緒安全性(thread-safe)，避免因為平行控制(concurrency control)的處理而降低了效能。

3．系統架構與方法

本論文以BOSC技術為主，首先延伸並實作了記憶體式資料處理技術，可以接收前端使用者對資料的讀取、寫入、更新、刪除等請求，讓資料的處理在記憶體中完成。

在記憶體式資料處理技術中，採用執行緒集區(thread pool)技術，產生出大量的工作執行緒，每一個工作執行緒一次負責一筆前端送來的請求，以阻塞(blocking)的方式依序執行以下步驟：讀取前端送來的請求、寫入日誌、等待日誌完成、於記憶體中處理請求、將請求結果回傳至前端等。

透過執行緒的集區技術，可以達到平行處理的效果，當某一個工作執行緒因為正在等待

輸入/輸出(IO)的時候，可以很自然地透過系統的上下文切換技術改為處理其他等待處理的執行緒，不會有中央處理器閒置的狀態。

在工作執行緒要寫入日誌時，BOSC的技術中會有一個獨立的硬碟日誌執行緒處理日誌的寫入。每個工作執行緒會將需要記錄的日誌資訊先寫到一個記憶體的暫存空間，等待硬碟日誌執行緒批次將日誌寫入硬碟後才算是完成日誌。

而當工作執行緒寫完日誌後接下來會在記憶體中處理該筆請求。記憶體中採用的是最基本的哈希表(hash table)來處理及儲存資料，包括：讀取、寫入、更新、刪除資料等。每筆請求進來時都會含有資料的欄位內容，針對欄位的訊息會使用哈希函數(hash function)計算出該資料的一個索引(index)，表示該資料位於哈希表中格子，然後於該格中執行請求所對應的操作行為。

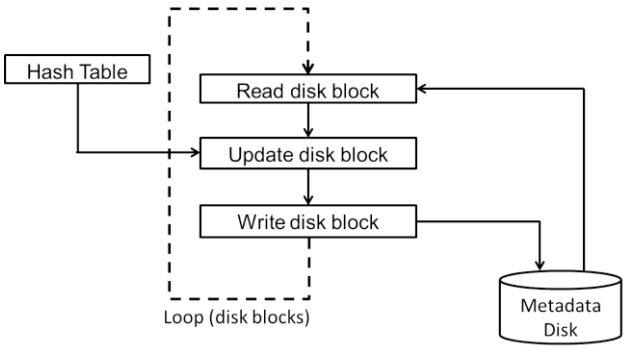

圖 1 以批次方式更新硬碟區塊

這些所做的執行請求會對記憶體中的內容做更新，然後由BOSC的批次更新循環送出技術，將記憶體中所儲存的這些變更，批次更新至硬碟中，如圖1所示，包括了章節2.1所述的三個步驟：讀出區塊、更新區塊與寫入區塊。

在這樣的基礎架構下，我們為了改善整體的效能，並讓系統擁有處理物聯網時代小型封包的能力，我們針對了幾個重要架構部份做修改，這些修改將於子章節中分別討論。

3.1 日誌處理

為了資料的持續性，需要實作日誌的記錄。每一個工作執行緒所執行的工作，皆需要確定其日誌記錄完成後才算真正完成。這是為了確保當工作執行緒已完成記憶體的資料變更後，但在更新至硬碟前若發生了意外狀況導致程式重啟時，能有日誌可以還原其工作內容。

BOSC的原本作法是將日誌寫入硬碟，雖然BOSC有提出改善的方法能降低寫入的延遲，但對硬碟的操作勢必會有一定的延遲時間。

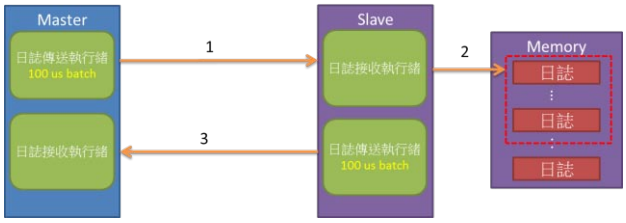

圖 2 以背景執行緒將日誌寫入遠端機器

相對的，本論文針對這部份改用寫日誌到另一台機器記憶體的方式，來增快效能，如圖2所示。程式會被安裝到兩台機器，每台程式會新增兩個執行緒來處理遠端機器的日誌溝通，一個為日誌傳送執行緒，負責將日誌相關的訊息傳送至遠端機器，另一個為日誌接收執行緒，負責接收來自遠端機器的日誌相關訊息。

當第一台程式的工作執行緒在處理前端的請求的時候，除了寫一份日誌到自己的記憶體之外，也必須等待日誌傳送執行緒將同樣的日誌傳送到第二台的程式，並收到第二台程式的日誌回應，才算完成。日誌傳送執行緒會每隔100微秒的時間，收集這段時間的所有日誌，一次傳送至遠端的程式。而對面的程式會使用日誌接收執行緒做接收，並在日誌儲存好之後日誌傳送執行緒回覆確認訊息。當兩邊的日誌都寫入至記憶體之後，即視為完成了日誌的寫入。不管哪一台機器毀損，都會有另一台機器所儲存的日誌可以提供足夠的還原資訊。

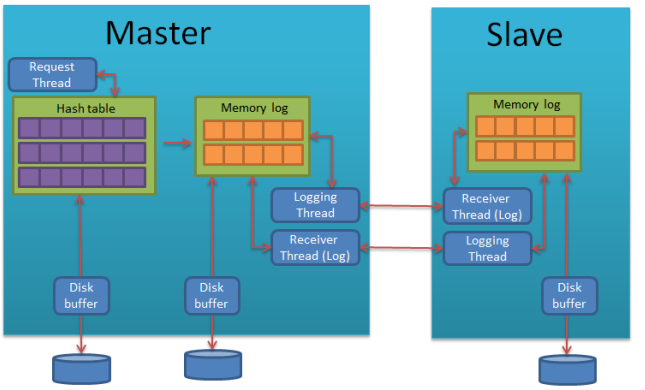

圖 3 系統架構圖

整體架構圖如圖3所示，程式中仍保留了原本BOSC的硬碟日誌執行緒，但硬碟日誌執行緒改為非同步(asynchronous)的方式處理，也就是說工作執行緒只需要確認在兩台機器的記憶體都有寫入日誌，就算完成日誌的寫入，而不用等待硬碟日誌執行緒將日誌寫入硬碟。硬碟日誌執行緒會以背景執行的方式每隔幾秒鐘就將目前記憶體的日誌批次寫入硬碟，並且由於日誌的寫入是多筆同時寫入，因此也不會產生出太多的輸入/輸出(IO)而影響整體效能。

3.2 結合mTCP/DPDK網路封包處理技術

為了能有效處理物聯網時代大量的小型資料，發揮出網路卡的全速效能，本論文採用了mTCP與DPDK技術。DPDK提供了能與硬體直接溝通的的用戶級別介面，mTCP使用了DPDK的函式庫(library)實作了傳輸控制協議層以下的用戶級別封包處理，來提供應用層的快速封包處理能力。

對應用層的程式而言，因為應用層只看得到傳輸控制協議層，因此應用程式只需要與mTCP溝通，而mTCP會再與底層的DPDK溝通。並且由於mTCP提供了與傳統網路端口類似的應用程序接口，因此應用程式可以很容易地使用mTCP。

然而，如章節2.2所述，應用程式內在使用mTCP時較佳的方式為採用epoll的架構，在應用程式內會產生N個執行緒以epoll的方式處理網路的輸入/輸出，本論文假設這N個應用程式內的執行緒為輪詢執行緒。同時，相對於這N個輪詢執行緒，mTCP內也會產生N個執行緒，每一個皆對應到一個應用層的輪詢執行緒，本論文假設mTCP內這N個執行緒稱為mTCP執行緒。需要注意的是，由於mTCP並不支援執行緒安全性(thread-safe)，因此N個輪詢執行緒所對應的N個網路端口不能同時讓應用程式內別的執行緒操作，否則會有資料處理錯誤的狀況發生。

本論文讓輪詢執行緒取代工作執行緒，輪詢執行緒相當於是一個工作分配者身兼工作執行者，將屬於輸入/輸出的部份分配給其他執行緒，而自己主要處理中央處理器的工作處理。由於程式中大部份需要中央處理器的工作皆讓輪詢執行緒執行，因此可以避免中央處理器頻繁在多個執行緒間上下文切換所造成的負擔。然而，要注意的是，由於輪詢執行緒的數量較少，因此輪詢執行緒不應該被任何輸入/輸出(IO)阻塞(blocking)住，所有有關輸入/輸出的處理皆要改為非阻塞(non-blocking)或非同步的方式。

輪詢執行緒需要分配出去的輸入/輸出的工作總共有兩類。其中一類是與前端使用者的溝通，這部份負責幫忙處理的是mTCP執行緒。另一類是日誌的處理，主要由日誌的相關執行緒來負責，包括硬碟日誌執行緒、日誌傳送執行緒與日誌接收執行緒。資料會在輪詢執行緒與輸入/輸出的執行緒溝通，比如說輪詢執行緒將要寫入日誌的資料傳給日誌傳送執行緒，而日誌處理完之後的回應就由日誌接收執行緒交回給輪詢執行緒。

輪詢執行緒與前端使用者的溝通是透過mTCP執行緒，透過mTCP與DPDK技術的幫忙，可以快速地處理來自使用者的大量小型封包。而需要注意的是，由於日誌的產生是在程式內部，雖然每筆日誌的資料都很小，但程式內部已經將多筆日誌批次處理，當日誌數量多的時候會集合成大封包在網路上做傳送，因此日誌的處理部份可以不使用mTCP或DPDK技術。

3.3 批次更新硬碟

本論文採用BOSC的批次更新循序送出技術，將多筆更新暫存於記憶體的哈希表中，對應寫入硬碟的部份會有一個背景執行緒做資料

更新至硬碟的動作。

該執行緒會先選定一個硬碟區塊，檢查對應的記憶體區塊是否有更新的資料，若有的話會對此區塊做更新，更新步驟包括如章節2.1所述的讀取硬碟區塊、對此硬碟區塊做更新與寫入硬碟區塊。

當前一個硬碟區塊做完更新之後，會依照硬碟區塊順序做下一個區塊的更新，以保證循序更新。而若該區塊於記憶體中沒有新的更新資料，則會跳過該區塊。

批次更新的效率會跟前端送進來請求的速度與記憶體的大小有關。當前端送進來的請求速度很低的時候，更新區塊時可能只更新到少少幾筆資料。而由於同一區塊中不管暫存了多少筆更新資料，在不超過一個硬碟區塊的儲存大小情況下，對硬碟做更新時所花的時間是差不多的。因此若每次做硬碟的更新時都只有少數幾筆資料可更新的情況下，更新效率自然也快不起來。因此當前端送進來的請求愈快時，硬碟的每秒有效更新資料次數也會隨著提升。

然而，當前端請求變得非常快的時候，記憶體的大小即成為了下一個影響更新效率的因素。當記憶體愈大的時候，每個區塊就能在記憶體中暫存較多的更新資料，因此更新硬碟區塊時也能有較多的更新資料可以寫進硬碟，可以有較高的每秒有效更新資料次數。反之，當記憶體很小時，就無法享受到批次更新的效果。

4 · 實驗

在本論文中使用了四台機器做測試，其中兩台為Intel(R) Xeon(R) CPU E5-2620 v2 @ 2.10GHz，為24核心的機器，另外兩台為Intel(R) Xeon(R) CPU E5540 @ 2.53GHz，為8核心/超執行緒(HT)16核心的機器，每台配置一張10Gb的網路卡，並以10Gb的網路交換器相連。

本論文的實驗利用E5540 @ 2.53GHz的其中一台當主機，另外三台當測試用戶端，而日誌記錄主機安裝在E5540 @ 2.53GHz的測試用戶端機器，也就是說此台機器身兼日誌記錄及測試用戶端的角色，其中兩個中央處理器核心給日誌記錄使用，另外六個中央處理器核心由測試用戶端使用。

由於物聯網像穿戴式手環這種應用，大多是偵測使用者的即時資訊並將資訊寫進伺服器內，幾乎都是新增資料的請求，因此本實驗主要專注在測試新增資料的數據。在此章節中將會有兩個子章節分別討論寫入日誌的效能與新增資料的效能。

4.1 日誌處理效能

在此子章節測試的是日誌的處理效能。日誌的處理是採用日誌傳送執行緒將一段短時間內的日誌收集起來，透過網路傳送將日誌寫入遠端機器。此實驗目的是想知道此方式是否會有效能上的瓶頸，而由於日誌為批次處理，因此不會有小封包在網路上傳送導致效率不佳的問題。

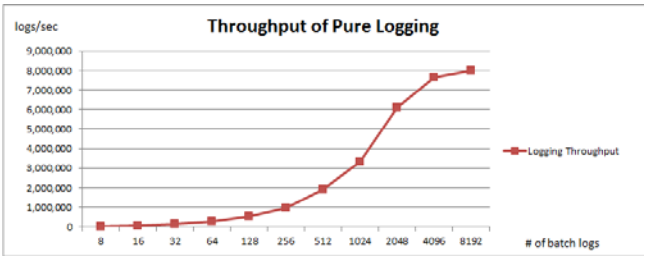

圖 4 批次寫入日誌的效率

實驗時每筆日誌的大小約為85位元組(bytes)，由程式內部自己產生，而不是由接收前端的使用者請求來產生，這是為了精確測量日誌的處理效率。

實驗結果如圖4所示，橫軸為日誌寫入執行緒每次傳送日誌的最大數量，也就是說每一次最多寫幾筆日誌到另一台機器，而縱軸為每秒處理成功的日誌數量。由圖中可以看出，隨著每次傳送的日誌數量增多，每秒處理成功的日誌數量也跟著增加。

假設希望整體效能可以達到每秒處理100萬個請求的情況下，寫入日誌效能也需要在每秒100萬以上，則以圖4的實驗來看，只要前端進來請求的速度夠快，寫入日誌的效能也能隨之提升，最快可以到每秒處理八百萬筆日誌，因此效能瓶頸不會在寫入日誌的效能。

此實驗同時也觀察了中央處理器的用量，以超執行緒16核的機器而言，平均每個核心只使用了4%至7%的用量，表示執行日誌的處理不會對中央處理器有太大的負擔。

4.2 整體效能

在此小節中將會加入mTCP與DPDK的技術，並比較效能數據。實驗時每筆資料大小一樣約為85位元組，測試的皆是新增資料的效能數據。實驗時假設硬碟空間足夠且不會有硬碟滿的狀況，記憶體預設最多可以儲存1億6千萬筆資料。

表 1 效能比較

Req/Sec	N	NH	NHD	NHL	NHDL
LinuxTCP	799,706	540,093	417,006	300,796	250,405
mTCP	2,208,262	897,920	634,251	740,663	624,477

我們用兩種程式版本比較，一種是原始的程式架構，該架構使用的是執行緒集區技術，每一個執行緒每次處理一筆前端送進來的請求，利用多執行緒達到平行處理的目的。此版本使用的是原本的作業系統提供的網路處理介面，我們將此版本在表1中標記為LinuxTCP。

另一個版本是如第三章所述的方式，使用mTCP與DPDK並將程式內部使用輪詢執行緒取代工作執行緒，此版本在表1中標記為mTCP。

首先，第一個實驗先觀察當前端送來請求時，在記憶體不做任何處理就直接回傳結果的速度，也就是說只有網路的輸入/輸出的效率，結果列於表1的第一欄N，很明顯可以看出mTCP及DPDK在網路方面效能的改良。再來，第二個實驗觀察當前端送來請求時，在程式記憶體內處理完就回傳結果的速度，此部份測試不包括日誌的處理也不需要將資料更新至硬碟，只會做記憶體內哈希表(hash)的處理，結果列於表1中的NH欄位。很明顯可以看出來因為多做了哈希表的處理，速度皆有下降，而mTCP的新版本速度下降更多，這顯示在記憶體的哈希表處理上可能還有改進的空間。

再來，考慮當記憶體滿的狀況的時候，在此情況下，每當一筆資料被更新至硬碟，記憶體釋放出一筆資料的空間，才能處理一筆前端新增資料的請求。當記憶體滿的狀況下的效能數據如表1中的NHD。接著，實驗測試當記憶體沒滿的情況下，每筆請求皆需要處理日誌，效能的結果列於表1的NHL。最後，實驗測試當記憶體已滿且每筆請求皆需要做日誌的時候的效能數據，結果列於表1中的NHDL。

由表1可以看出，在使用新架構後，在記憶體已滿及需要日誌的情況下，效能仍可以達到每秒處理60萬筆請求以上，而當機器記憶體夠大的情況下，在做日誌處理保證其資料持續性的情況下效能更可以達到每秒74萬筆請求。

5 · 結論

本論文以批次更新循序送出(BOSC)的技術為基礎，實作了記憶體式處理技術，使用遠端機器儲存日誌的方式提供資料的持續性，並結合mTCP與DPDK的封包處理技術，提供物聯網時代處理大量的小型封包的能力。

實驗結果顯示，在前端收送使用者的請求、每筆請求需要記錄日誌的情況下，若記憶體足夠時效能可以達到每秒處理74萬筆新增資料的請求；而記憶體已滿時效能則為每秒62萬筆請求。

參考文獻

[1] (2015) Explosive Internet of Things Spending to Reach $1.7 Trillion in 2020, According to IDC. [Online]. Available://www.idc.com/getdoc.jsp?containerId=prUS25658015

[2] D. N. Simha. Efficient Implementation Techniques for Block-Level Cloud Storage Systems. PhD thesis, Stony Brook University, 2014.

[3] D. N. Simha, M. Lu, and T.-c. Chiueh. An update-aware storage system for

low-locality update-intensive workloads. In ACM SIGPLAN Notices, volume 47, pages 375–386. ACM, 2012.

[4] E. Jeong, S. Wood, M. Jamshed, H. Jeong, S. Ihm, D. Han, and K. Park. mtcp: a highly scalable user-level tcp stack for multicore systems. In 11th USENIX Symposium on Networked Systems Design and Implementation (NSDI 14), pages 489–502, 2014.

[5] Intel dpdk: Data plane development kit. [Online]. Available://dpdk.org.

[6] T.-c. Chiueh and L. Huang. Track-based disk logging. In Dependable Systems and Networks, 2002. DSN 2002. Proceedings. International Conference on, pages 429–438. IEEE, 2002.

[7] S. Han, S. Marshall, B.-G. Chun, and S. Ratnasamy. Megapipe: a new programming interface for scalable network i/o. In Presented as part of the 10th USENIX Symposium on Operating Systems Design and Implementation (OSDI 12), pages 135–148, 2012.

[8] A. Pesterev, J. Strauss, N. Zeldovich, and R. T. Morris. Improving network connection locality on multicore systems. In Proceedings of the 7th ACM european conference on Computer Systems, pages 337–350. ACM, 2012.

作者簡介

林宏軒

國立交通大學資訊科學與工程研究所博士。現任工研院資訊與通訊研究所資料中心架構與雲端應用軟體組技術副理，專長於雲端儲存系統、記憶體式資料處理、棋類人工智慧。

張鎮庭

國立交通大學資訊科學與工程研究所碩士。現任工研院資訊與通訊研究所資料中心架構與雲端應用軟體工程師，專長於雲端儲存系統、記憶體式資料儲存處理、中間代碼優化與轉換。

徐國峰

國立台灣大學電機工程學研究所研究生。於陳銘憲教授所指導之網路資料庫實驗室進行研究，專長於記憶體式資料儲存處理、社群網路分析應用。

闕志克

工研院資訊與通訊研究所所長。美國柏克萊大學電腦科學博士。

陳玉倫

工研院資訊與通訊研究所資料中心架構與雲端應用軟體組副組長。美國威斯康辛大學麥迪遜分校電腦科學碩士。