國立臺灣大學電機資訊學院資訊網路與多媒體研究所

碩士論文

Graduate Institute of Networking and Multimedia

College of Electrical Engineering and Computer Science

National Taiwan University

Master Thesis

使用高效能遠端虛擬記憶體聚合未使用之記憶體空間

Aggregating Unused Memory with Efficient Remote Swapping

高至賢

Tzu-Hsien Kao

指導教授：洪士灝 博士

Advisor: Shih-Hao Hung, Ph.D.

中華民國 106 年 7 月

July, 2017

doi:10.6342/NTU201704017

國立臺灣大學碩士學位論文
口試委員會審定書

使用高效能遠端虛擬記憶體聚合未使用之記憶體空間

Aggregating Unused Memory with Efficient Remote Swapping

本論文係高至賢君（學號 R01944033）在國立臺灣大學資訊網路與多媒體研究所完成之碩士學位論文，於民國一百零六年七月廿一日承下列考試委員審查通過及口試及格，特此證明

口試委員：　洪士灝　（簽名）

（指導教授）

廖世偉　　涂嘉恒

所　長：　楊佳玲

doi:10.6342/NTU201704017

誌謝

硕士班的學習，我認爲是學習如何從接受、使用知識的角色轉變成如何站在巨人的肩膀上向前更加邁進一步的角色的過程。

在這個過程中，首先需要感謝洪教授的指導。因爲有老師從題目開始到研究完成這整個過程中的建議與指點，讓我可以有機會了解系統研究的方法，得以一窺這個領域的大門。也是因爲有老師在研究方法上的指導，我才終於可以完成碩士的學業。謝謝老師給我的這些機會和包容，讓我在碩士班的這段時間沒有白廢，能帶回在研究、學習、做事方法上的豐碩收穫。

其次要感謝的是博士班的阮渥豪學長與劉政岳學長。兩位學長在研究的過程中不厭其煩地回應我的問題，並且多次與我討論研究的方向與呈現的表現方式。學長們的經驗與著眼點不僅是我學習的榜樣，也幫助我減少研究過程中可能會遇到的狀況。少了這些協助，我是不可能完成這一切的。

再來，要感謝 PAS Lab. 的各位同學。和各位一起學習討論的過程中我得到相當多的良性刺激，同學們堅強的精神和開朗的個性也總是讓我得到相當大的鼓勵。

最後要感謝我的家人與女友的陪伴和支持，在我遇到低潮時你們總是在我身邊。學習做研究有很多需要克服的難題，少了你們作我的後盾、我走不到這一步。

doi:10.6342/NTU201704017

摘要

近年來因爲大數據分析的需求，資料中心與叢集內部高速網路效能朝向高頻寬、低延遲的方向急遽成長。比起過去，利用高速網路技術共享伺服器間的資源的效率變得前所未有地高。在這篇論文中，我們研究透過高速內部網路伺服器間遠端共享記憶體的技術，透過虛擬記憶體系統將遠端伺服器的記憶體作爲交換空間使用。我們提出以的高度可移植的交換記憶體架構，是使用產業標準的開放原始碼系統程式和硬體架構所完成。故只需要進行軟硬體的配置與安裝，不需要對作業系統加上特殊的修改，我們相信這樣的架構可以廣泛應用在各領域中。

我們用簡單的測試程式與實際的應用作爲此遠端記憶體機制的驗證，包含記憶體內快取系統（memcached）、訓練機械學習模型（Tensorflow）與基因定序（MUMmer）於 50Gbps 的高速乙太網路環境。實驗結果顯示我們的機制可以提供高效率的遠端虛擬記憶體存取：跟使用傳統硬碟的記憶體交換機制比起來，我們不需要爲了避免虛擬記憶體的猛移現象（thrashing）而付出高昂的記憶體採購成本。舉例來說，使用我們的方式，TensorFlow 訓練深度學習模型時，效能可以僅低於使用足夠放進所有資料的實體記憶體時 1.24 倍、並發揮比傳統硬碟快 16 倍的效能。而且因爲使用 RDMA over Converged Ethernet（RoCE）網路，我們的機制只造成雙方伺服器間極低的間接成本。

關鍵字： 交換空間、遠端交換空間、RDMA、RoCE

doi:10.6342/NTU201704017

Abstract

In recent years, the performance of interconnection networks in the datacenter have been vastly improved with higher bandwidth and lower latency, driven by the demand of big data analytics. With the high-speed network technologies, sharing of resources among different servers becomes more efficient than ever. In this thesis, we study remote swap memory technologies which allows one server to utilize the memory on a remote server as the swap memory for the virtual memory system via a high-speed interconnection network. We propose a portable remote memory swap mechanism with reliable open-source system software and industrial standard hardware components. The construction of the mechanism is done by configuring the software and hardware beyond the operating system without level vendor-specific modifications, so we believe the methodology is generic and is useful to a wide range of applications.

To evaluate the performance of our proposed mechanism, we carry out microbenchmarks as well as realistic applications, including in-memory cache (memcached), machine learning model training (Tensorflow), and genome sequence alignment (MUMmer), on a setup with two servers connected via 50Gbps Ethernet. The experimental results show the efficiency of the our mechanism. The remote memory swap mechanism is faster than traditional memory swap mechanism with hard disks and saves the cost of adding physical memory to avoid thrashing of virtual memory. In our experimental results, using our remote swap mechanism, TensorFlow training deep learning mod-

doi:10.6342/NTU201704017

els was accelerated by 16 times, compared to swapping using local disk. It ran only 1.24 times slower than running on a server with larger physical memory to hold the entire data set. Due to the use of RDMA over Converged Ethernet (RoCE), our remote memory swap mechanism caused little overhead to both servers.

Keywords: Swap, Remote Swap, RDMA, RoCE

doi:10.6342/NTU201704017

Contents

doi:10.6342/NTU201704017

List of Figures

doi:10.6342/NTU201704017

List of Tables

doi:10.6342/NTU201704017

doi:10.6342/NTU201704017

Chapter 1

Introduction

Modern systems process large amount of data in memory with an improved throughput. Such big-memory workloads seldom use the virtual memory because of the high cost of swapping[4]. Swapping to disk may severely impinge on the overall performance for these workloads. In recent years, the performance of interconnection networks in the datacenter continues to increase. Modern networking technologies such as *Infiniband* [10] features low latency (in the order of microseconds), high throughput (up to 200Gbps), and the support of Remote Direct Memory Access (RDMA) [17] operations to reduce CPU utilization.

The advent of high-speed networking leads us to fine-grain sharing of various resources in the datacenter, including the memories. In a traditional datacenter, certain resources become underutilized because of mismatching workload requirements. For example, compute cycles may be exhausted before memory capacity is reached, leaving a fraction of the memory unused. Data gathered from datacenters show that server memory is unused as much as 50% [2]. Therefore, disaggregating memories and placing them in pools will be beneficial.

In order to use high-speed interconnects to access remote memory, the clients can naively use remote memory pools as block pseudo-devices, and the server need to make local memory available for clients. However this kind of implementation, need to be modify both client and server side systems. These modifications make system unportable, and hard to use.

doi:10.6342/NTU201704017

In this thesis, we study remote swap memory technologies which allows one server to utilize the memory on a remote server as the swap space for the virtual memory system via a high-speed network. We designed a framework that enables one server to utilize the memory in another server to execute large-scale memory-intensive applications that demand more memory than the server can provide by its physical memory. Via a high-speed network, page faults in the virtual memory system can be satisfied quickly by swapping infrequently used pages into the memory on a remote server.

To maximize the applicability of the framework, we set the following goals:

- The framework should be as portable as possible. To achieve this goal, we choose to construct the framework with open source software and commodity hardware components.
- The framework should be as efficient as possible. The remote memory swap mechanism should outperform traditional swap mechanism with hard disks and be comparable to solid-state disks (SSD).
- The framework should enable unmodified applications to benefit from remote swapping over RDMA networks without operating system modifications.

Note that, while parallelizing the application and divide the data to run on multiple servers should be able to achieve better performance and scalability than our approach. Our framework aims to provide a convenient, transparent way to mitigate the problem of resource fragmentation.

Other storage devices like SSD may have comparable performance to remote memory system via high-speed network, but the price of high-performance SSD and the lifetime of SSD have been issues for adopting SSD as swap devices in practice [14, 3, 22].

As a result, our proposed framework provides a portable remote memory swap mechanism with reliable open-source system software and industrial standard hardware components. The construction of the mechanism is done by configuring the software and hardware beyond the operating system without level vendor-specific modifications, so we believe the methodology is generic and is useful to a wide range of applications.

doi:10.6342/NTU201704017

To evaluate the performance of our proposed mechanism, we carry out micro-benchmarks as well as realistic applications, including in-memory cache (*memcached*), machine learning model training (*Tensorflow*), and genome sequence alignment (*MUMmer*), on a setup with two servers connected via 50Gbps Ethernet. The RDMA over Converged Ethernet [20] (*RoCE*) protocol allows RDMA operations over an Ethernet network.

The experimental results show the efficiency of the our mechanism. First, the remote memory swap mechanism is faster than traditional memory swap mechanism with hard disks. The cost of additional physical memory for virtual memory thrashing avoidance can be saved. Our experiment results from our micro-benchmark reveal the worst case, where our remote swapping mechanism outperformed hard disk setup by 3.4 times and ran only 3 times slower than running on a big-memory server with larger physical memory to hold the entire data set.

From the proposed remote swap mechanism, applications are also benefited. For example, when using our remote swap mechanism, training time of TensorFlow models was accelerated by 16 times, compared to swapping using local disk. It ran only 1.24 times slower than running on a server with sufficient physical memory to hold the entire data set.

Finally, due to the use of *RoCE*, our remote memory swap mechanism caused little overhead to both servers. By using the low CPU utilization RDMA operations, our remote memory swap mechanism provide a new vision to utilize remote memories for local system performance improvement.

The rest of the thesis is organized as follows. Chapter 2 discusses the related work. Chapter 3 provides the relevant background. Chapter 4 present the design and implementation. An experimental evaluation of the remote memory pager is discussed in Chapter 5. We conclude our work in Chapter 6.

doi:10.6342/NTU201704017

Chapter 2

Background

In this chapter we overview necessary background and explain the benefits from remote memory swapping. Remote memory swapping is composed of three main components: high-speed interconnect network, Linux memory swap mechanism, and network block storage device.

The latency of swapping is determined by software processing time and hardware transfer time. First, high-speed interconnect network provides low response time and high bandwidth which can transfer data many times faster than traditional disks, and using RDMA operations can reduce the CPU processing time of network protocol, which is discussed in Chapter 2.1 Then, in Chapter 2.2 we introduce the Linux memory swap mechanism and compare the pros and cons between the traditional swapping methods like swapping memory to disks, and swapping memory to remote memory. Finally we overview the *Linux SCSI target*, a block storage networking standard which supports a rapidly growing number of fabric modules, and all existing Linux block devices as backstores. We explain how and why we use this network block storage for remote memory swapping in Chapter 2.3.

2.1 High-Speed Network and RDMA

Over the years, Ethernet speed has increased from 10 megabits per second (Mbps) to 200 gigabits per second (Gbps) and researchers are already planning to scale up to 1 ter-

doi:10.6342/NTU201704017

abits per second (Tbps). Since 10 gigabit Ethernet (10GbE) or faster interconnections has become an inexpensive commodity with a growing installation base, servers with 10GbE network interfaces have been around. Due to more efficient processors, a faster PCI Express (PCIe) bus, and more sophisticated transfer protocols, the capability to have a single file transfer saturate a link become more common.

As centralized services, such as backup servers, often need to handle parallel data transfers contending for bandwidth, there is a need to deploy faster technologies to accommodate the aggregation of those flows[16, 7, 11]. Dell'Oro Group predicts in its recently released *Ethernet Switch —Data Center Five Year Forecast Report*[6] that 400-Gbps switch ports will begin to ramp strongly in 2019. Meanwhile, more than half of data center switch ports will operate at either 25Gbps or 100 Gbps in 2020, the market research firm believes. While it is possible to get TCP to saturate 40Gbps or faster connections, it is very sensitive and requires careful tuning. Even then requires significant CPU power to achieve.

RDMA protocol is a well-proven data center technology offering high performance and efficiency. RDMA protocol supports *zero-copy* networking. It is designed for communication within interconnected compute nodes, I/O nodes and devices in a system area network. RDMA operations allow one side of the communication parties to exchange information directly with the remote memory without the involvement of the remote host. This enables better computation and communication overlap, thus provide potentials for performance improvements.

Latency of RDMA operations between two nodes is quite comparable to local memory access latency[12]. Thus using RDMA operations provides the potential of significant performance improvement for remote memory swapping.

2.2 Linux Memory Swap Mechanism

Linux virtual memory system manages all physical memory resources. When free pages available to virtual memory system fall below a threshold, pageout requests are

doi:10.6342/NTU201704017

triggered by the kernel thread **kswapd** to swap pages out to swap devices. *Page-in* requests are invoked on demand as page faults occur. *Page-out* data are placed to these devices based on their priorities[5].

Any block device can be used as a swap space, e.g., Hard Disk Drives (HDDs) and Solid State Drives (SSDs). Swapping devices such as HDDs or even SSDs operate at several orders of magnitude slower than main memory modules. Excessive paging activity to and from the swapping device renders a system crawling as the CPU is mostly waiting for I/O activity.

For remote memory swapping, pre-allocated RAM disks on remote nodes perform this role. Although swapping to RAM disks performs worse than running with enough local memory, it still performs much better than swapping to HDDs. The CPU utilization can downgrade to 20% while swapping to a local RAM disk, 8% while swapping to remote memory, and less than 1% while swapping to local disk[21].

2.3 Linux SCSI Target

iSCSI (Internet Small Computer System Interface) provides *SCSI* accesses over IP networks. *iSCSI* Extensions for RDMA [18] (*iSER*) is a network protocol that extends *iSCSI* to use RDMA. *iSER* permits data to be transferred directly into and out of remote *SCSI* computer memory buffers over *InfiniBand* and Ethernet networks without intermediate data copies by using RDMA.

LinuxIO [13] (LIO) is the standard open-source *SCSI* target in Linux. It supports almost all prevalent storage fabrics. *LIO* supports kernel level RAM disk as backstore, which implement the methods of accessing data on disk. A backstore subsystem plugin is a physical storage object that provides the block device underlying a *SCSI* endpoint. Using the kernel level RAM disk can avoid the high memory protection overhead of using the user level RAM disk.

By using the kernel level RAM disk supported by *LIO* and *iSCSI* with the *iSER* protocol, the high CPU utilization of using the remote RAM disk as swap space can be avoided.

doi:10.6342/NTU201704017

Chapter 3

Related Work

Studies of memory disaggregation deal with the remote memory as an extension to the local memory space. Remote memory swapping, a disaggregation approach which considers remote memory as swap space, have demonstrated the ability to be deployed transparently with little/no modification to the OS or the running applications.

However, the performance and promises were often limited by slow networks and high CPU overheads. Several works have studied the remote memory swapping for different purposes as shown in Table 3.1. We differentiate them by network technology, bandwidth and portability.

Nswap [25] proposed a block device driver which combines networks and swap devices to allow cluster nodes with over-committed memory to use idle memory and to swap its pages over the network. They used slow 100Base-T Ethernet and hard disks to examine their remote memory swap mechanism. Performance of the *Nswap* is limited due to the low hard disks and the lack of high-speed network. Also, operating system on both client and server sides need to be modified.

HPBD [12] proposed the design and implementation of a high performance networking block device (*HPBD*) over 10Gbps *InfiniBand* fabric, which serves as a swap device of kernel Virtual Memory (VM) system for page transfer to/from remote memory servers. To deploy *HPBD*, installation of the kernel module and the management daemon are needed. The applications they used for experiments like **qsort**, which access remote memory not so frequently. *Nuzura* [8] measured the performance of some HPC applications instead,

doi:10.6342/NTU201704017

	Network	Bandwidth	Modification of OS
Nswap[25]	TCP	100Mbps	needed
HPBD[12]	InfiniBand	10Gbps	needed
Nuzura[8]	RoCE	10Gbps	needed
INFINISWAP[9]	InfiniBand	56Gbps	needed

Table 3.1: Modern work in designing remote memory system

which required several times of local memory on their system. *Nuzura* proposed an approach similar to *HPBD*. They implemented a remote memory swap system and examined the availability of remote memory swapping via their own *UZURA* 10Gbps *RoCE* interconnects.

INFINISWAP [9] implemented a remote memory swap system on a 56Gbps, 32-machine *InfiniBand* interconnected RDMA cluster and evaluated it using multiple unmodified memory-intensive applications. It shows that when working sets do not fit in memory, applications performance degrade linearly using *INFINISWAP* instead of experiencing a super-linear drop. *INFINISWAP* implemented their mechanism by implementing a kernel module and a user space daemon, which should be installed when deploying the cluster environment.

Our work are tested on the 50Gbps Ethernet which is becoming around in datacenters. The framework we proposed is different from the previous works in that operating system and system software we used are not modified. When deploying servers, using our remote memory swap mechanism is not necessary to install special device driver or modified kernel to support applications which require pages from remote memory. Service providers can use the stable version operating system and system software to serve big-memory workloads reliably.

doi:10.6342/NTU201704017

Chapter 4

Remote Memory Swap Framework

The main goal of remote memory swapping is to efficiently expose remote memories to user applications without any modifications to those applications or the OSes of individual machines. It must also be reliable and low operation overhead so that application performance on remote machines remains unaffected. Thus, the purpose of our work is to design an efficient, portable framework for swapping memory to remote memory, which means we should use existing reliable software to contribute our framework instead of implementing new kernel modules or system daemons.

In this chapter, we present the design and analysis of the performance issues in our framework. We propose the main design and discuss the design issues in Chapter 4.1. In Chapter 4.2 we discuss the potential performance issues and the trade-offs.

4.1 Designing the Remote Memory Swap Framework

As shown in Figure 4.1, our remote memory swap mechanism serves the kernel's paging requests by communicating with remote memory servers. The client is direct connected to server via 50Gbps *RoCE* network without network switch.

The client is an *iSCSI* initiator, which serves I/O requests stream from the virtual memory system by sending requests to the remote memory servers. The server is a RAM disk *iSCSI* target, which provides its own local memory for paging store and push/pull pages from client using RDMA operations with the *iSER* protocol.

doi:10.6342/NTU201704017

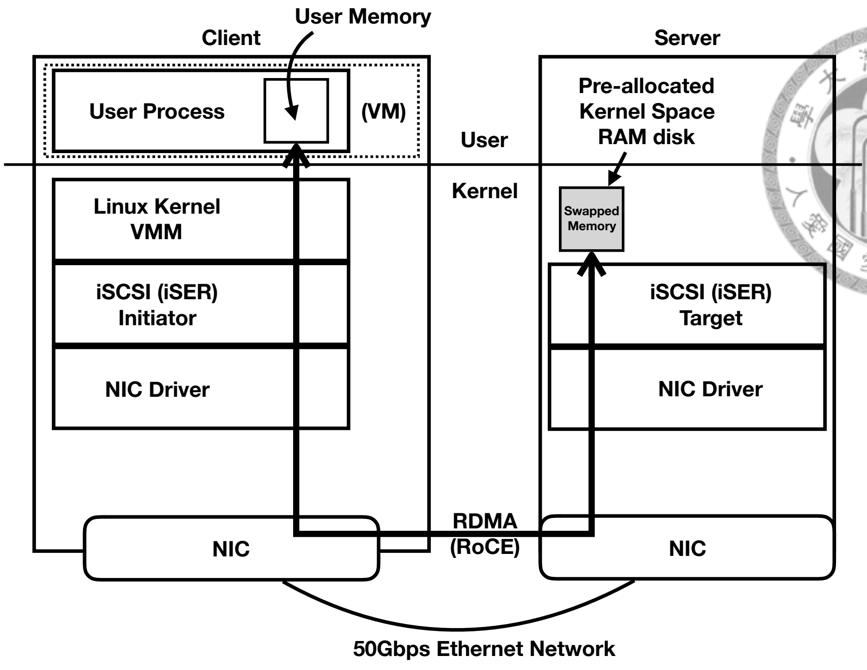


Figure 4.1: Remote Memory Swap Framework Architecture

Using the *iSCSI*, a reliable software emulation for local block storage, remote resources at the block level can be used efficiently. We use *iSCSI* because it is widely used in file systems for data storage. Introduced in Chapter 2.3, when using the remote RAM disk as swap space, low memory protection overhead is promised by the kernel level RAM disk supported by *iSCSI*, and low operation overhead is promised by using *iSER* protocol.

In our framework, user processes can run natively or in virtual machine (VM) environments. User applications can access remote swap space as the extension of local memory, and VMs can use remote swap space as their VRAM by Linux Kernel-based Virtual Machine (*KVM*) memory overcommiting[19]. Remote memory swapping capability allows VMs to take the benefits of remote memory transparently by allocating more virtualized memory than there are physical memory on the system. Therefore, cloud computing environments and data centers can utilize their unused memory in *RoCE* interconnected clusters.

doi:10.6342/NTU201704017

4.2 Performance Issues

Asynchronous Communication: In a client-server architecture, the swapping process sends out paging requests to remote memory swap server and waits to be served. In the swap-in (page-in) case, the client user process/thread which requires remote pages is blocked until the server responses. RDMA operations can reduce asynchronous communication overhead by its low CPU utilization and fast response time.

Remote Memory Overhead: RDMA operations implement *zero-copy* remote DMA, but virtual memory system copy pages to/from remote swap space. Linux kernel uses 4KB page size for x86/x64 architecture. In 50Gbps interconnected cluster, a client may requests up to 1600K pages per second to remote memory servers, but due to the RAM disk emulation overhead, a memory server can only serve 160K pages requests by our measurement. We trade off portability of applications and OSes against the utilization of RDMA operations for performance in this work.

doi:10.6342/NTU201704017

Chapter 5

Performance and Evaluation

In this chapter, we evaluate the performance of the proposed framework comparing to a traditional system with HDDs swapping, to a system with our remote memory swap mechanism.

We use 4 different test programs. One is a micro-benchmark, which allocates a 8 GB array and sequentially write integers into this array. Because writing memory sequentially performs traditional HDDs the best due to the locality, this test reveal the worst case of our mechanism compared to the disk swapping. The second application is *memcached*. *Memcached* is a frequently used in-memory cache for various applications. Improvements of in-memory cache performance from remote swapping may provide a ground for benefits of other workloads using *memcached*. The third application is *TensorFlow*. Training large deep learning models using TensorFlow may demand more memory than the server can provide by its physical memory. The last application is *MUMmer*, which is a system for rapidly aligning entire genomes, whether in complete or draft form. Aligning entire genomes sequence data will require very high-performance computers, of the type currently available only at the largest sequencing and bioinformatics centers.

For each of the tests, we run these applications 10 times and report the average performance number.

doi:10.6342/NTU201704017

5.1 Experimental Setup

The experiments are conducted on two *Intel Core i7-5930K* servers with 32GB DDR3-1333 RAM on each server. Server and client nodes are connected to *RoCE* network using *Mellanox MT27708A0* with opensource *OFED* Linux driver. The HDD configuration on each node is *WESTERN DIGITAL WD10EZEX 1TB SATA600*. The operating system is *CentOS 7.3* with Linux kernel version is 3.10.0.

To compare the performance impact of remote paging with local memory performance, we change the total local memory size available to the OS and vary the swapping devices. Three testing applications and one micro-benchmark test are used in our evaluation. In each test, we test our mechanism with 3 setups: enough local memory, remote memory swapping and local disk swapping. We use the performance of applications running with enough memory as the baseline for evaluation. We use all of the 32GB memory physically available for the local memory setup test.

5.2 Performance Results of Micro-benchmark

Our micro-benchmark program allocates a 8 GB array and sequentially write integers into this array. In this test, we set the local memory size as 4GB and a RAM disk at remote server as swap area.

As shown in Table 5.1, the performance of remote-swap setup is only 3 times slower than the local memory setup, and is 3.4 times faster than the disk-swap setup even in the worst case. Also, the low CPU utilization of remote-swap setup is benefited from our remote memory swap mechanism. The results of micro-benchmark show that our remote memory swap mechanism can perform much better than traditional disk swapping even in the worst case.

Due to the asynchrony of different components in the operating system, an accurate measurement is not possible without thorough analysis of the swapping mechanism of the kernel, which is beyond the scope of this work. We have measured and profiled the behavior of the proposed mechanism in another work.

doi:10.6342/NTU201704017

	CPU Wait Time	User CPU Usage	Elapsed Time
Real	0s	100%	3.11s
HDD	22.28s	32%	32.82s
RDMA	0.32s	96%	9.63s

Table 5.1: Results of the Micro-benchmark Test

5.3 Application Performance Results

5.3.1 Memcached

Memcached is an in-memory object caching system that provides a simple key-value interface. We use *twemperf*[26], a tool to measure *memcached* server performance. Our experiments request the local *memcached* server for 19.5GB data in different block size from 32 to 1024 KB. We use 200 clients to generate requests for each block size test. In this test, we set the local memory size as 12GB and a RAM disk at remote server as swap area.

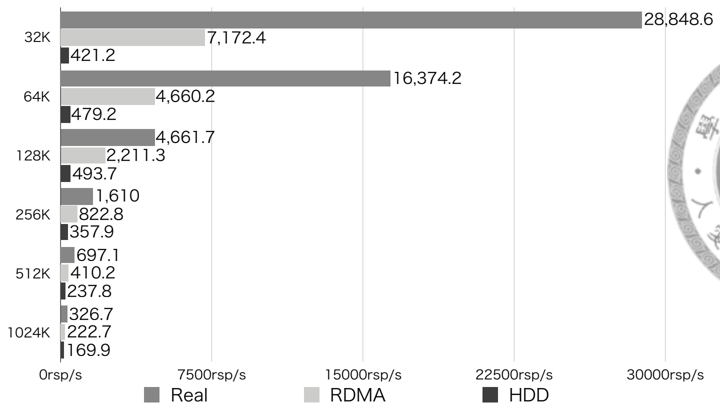
As shown in Figure 5.1, the results of the response rates of the remote-swap setup are lower from only 1.46 times to 4 times than local memory while the response rates of using disk setup are lower from 2 times to 68.5 times than local memory setup. The results of P99 response time and duration time also show that our remote memory swap mechanism performs tolerably in some cases for in-memory caching system.

5.3.2 TensorFlow

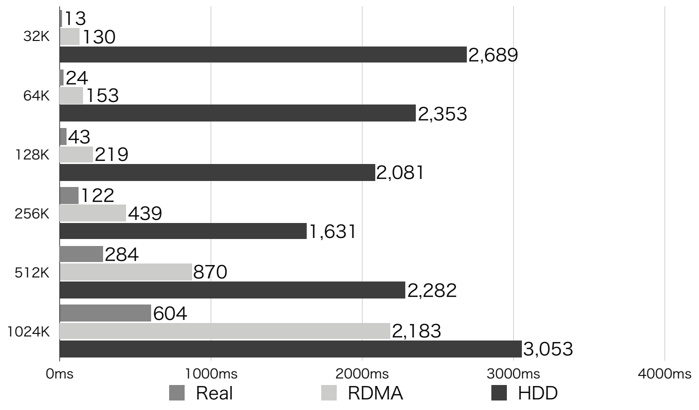
TensorFlow is an open source software library for machine learning using data flow graphs. Even though the deep learning on GPU runs much faster than on CPU, for the tasks which are not critical, CPU may be a smart, cost-effective choice. In cloud computing environment, some deep learning tasks on the cheaper CPU instances instead of GPU instances run only slightly slower with about 2/3rds of the cost of the GPU instance[15]. With remote swapping, throughput of *TensorFlow* may be improved in an interconnected cluster.

We choose 2 famous deep learning applications: image classification and text summa-

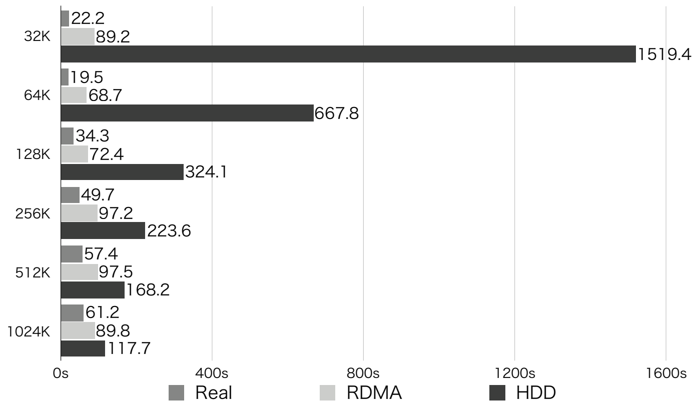
doi:10.6342/NTU201704017



(a) Response Rates



(b) P99 Response Time



(c) Duration Time

Figure 5.1: Results of the Memcached Tests

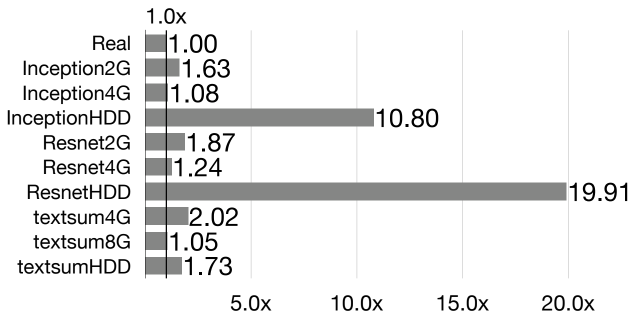
rization. For image classification test, we use two different models in *TensorFlow* models: *Resnet* and *Inception*, which are two Convolutional Neural Network (CNN) image classification models. We uses the *textsum* model to test the text summarization traingng performance. *Textsum* is a sequence-to-sequence with attention model for text summarization, for our tests.

doi:10.6342/NTU201704017

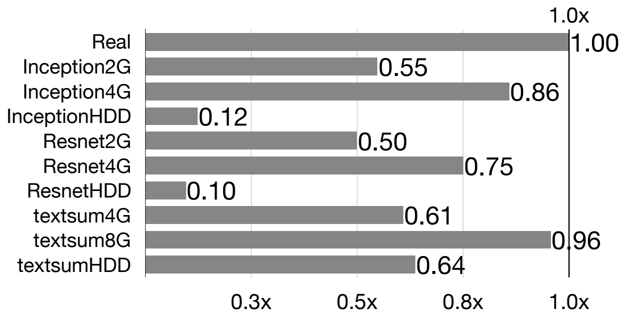
We follow the example code for training the models[1]. The *ImageNet*[24] dataset is used for image classification tests, and we use the official toy data and vocab examples to run the *textsum* training. Training runs 50 steps for each test.

Inception	Resnet	Textsum
6.8GB	8.5GB	9.6GB

Table 5.2: The Memory Usage of TensorFlow Models Training



(a) Duration Time Relative to Local Memory Setup



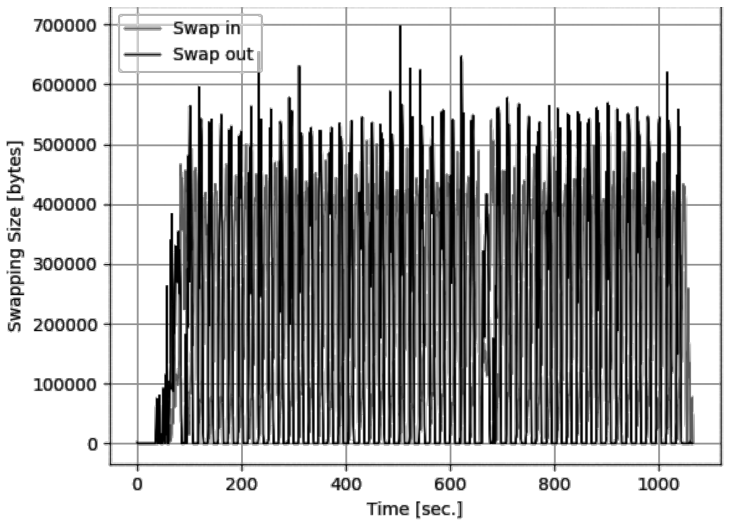
(b) CPU Usage Relative to Local Memory Setup

Figure 5.2: Results of the TensorFlow Model Training Tests

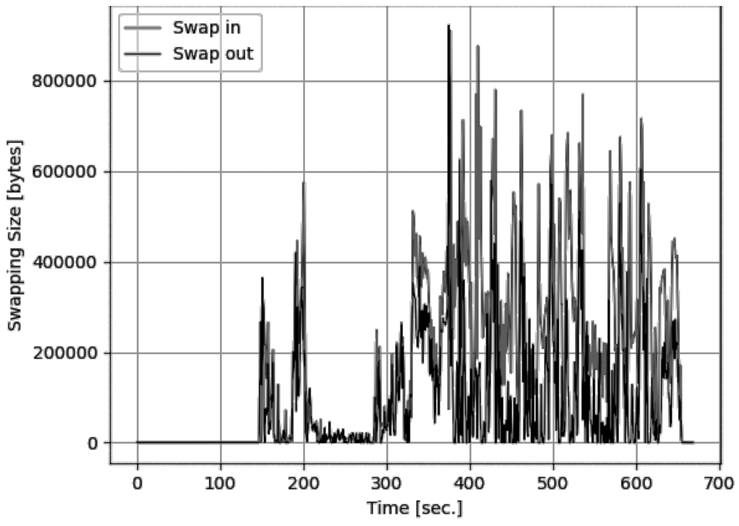
In the tests, we set the local memory size as 2GB, 4GB for image classification tests and 4GB, 8GB for text summarization to examine the performance of different swapping usage. Our machine freezes when we run the 2GB for classification and the 4GB for text summarization using disk setup, so we only have the 2GB image classification and 8GB text summarization results for disk setup. Table 5.2 shows the memory usage measured when using the local memory setup.

As shown in Figure 5.2, the performance of remote-swap setup is 16 times faster than disk-swap setup and is only 1.24 times slower than local memory setup in *Resnet* training test. Training models is memory-intensive as shown in Figure 5.3(a). When training the

doi:10.6342/NTU201704017



(a) Resnet



(b) Textsum

Figure 5.3: Swap Usage Comparison of Training Different Models

big models like *Resnet*, the slow swap device like HDD impinges on the overall performance.

The performance downgrade is not significant in training the *textsum* model when using a slow disk swap because the concentrative swap usage of *textsum* model training as shown in Figure 5.3(b). The result showed that the overhead was varied depended on the application memory access pattern.

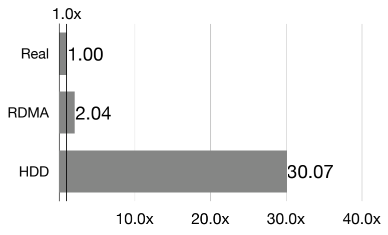
doi:10.6342/NTU201704017

5.3.3 MUMmer

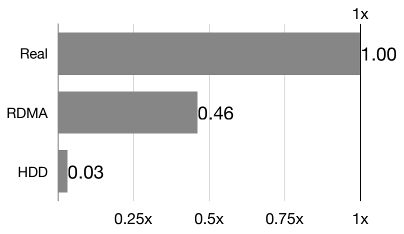
In bioinformatics, a sequence alignment is a way of arranging the sequences of DNA, RNA, or protein to identify regions of similarity that may be a consequence of functional, structural, or evolutionary relationships between the sequences. One of the most successful algorithms for computing alignments between genome sequences is *MUMmer*[23].

The test reads 100-character substrings sampled from the *Bacillus anthracis* genome (GenBank ID: NC_003997.3). Thus, each read exactly aligns to the genome end-to-end at least once, and possibly more depending on the repeat content of the genome. In this test, we set the local memory size as 8GB and a RAM disk at remote server as swap area.

On average, the local memory setup for our genome sequence aligning test uses average 94% CPU and requires 13GB memory space which is accessed frequently. As shown in Figure 5.4, performance of the disk-swap setup is 30 times slower than the local memory setup. The experimental results show that disk-swap setup is too slow to satisfy page faults from the application. Performance of the remote-swap is only 2 times slower than local memory setup, which means our remote memory swap system can provide tolerable virtual memory performance for a high performance application like sequence alignment.



(a) Duration Time Relative to Local Memory Setup



(b) CPU Usage Relative to Local Memory Setup

Figure 5.4: Results of the MUMmer Sequence Alignment Test

doi:10.6342/NTU201704017

Chapter 6

Conclusion and Future Work

In this thesis, we study the availability of utilizing remote memory in RoCE interconnected environment. We proposed a portable, efficient remote memory swap mechanism and evaluate the performance of 3 big-memory memory-intensive applications. Our experimental results showed that using our remote memory swap mechanism, TensorFlow training deep learning models can run up to 16 times than swapping using local disk, and runs only 1.24 times slower than local memory system. Our solution allows remote memory swapping to enhance local memory hierarchy by the virtual memory system via a high-speed network by configuring the software and hardware, without operating system modifications.

Because of the hardware limitations, we experimented our remote memory swap framework in a direct connected network environment without network switches. Therefore, performance of swapping memory to multiple remote memory pools is not available, and fault-tolerance of remote memory swapping is also unavailable with only one swap device.

In our future work, we plan to enable the remote memory swap mechanism with multi-client/multi-server architecture with fault-tolerance. We also intend to investigate designs that can automatically set up and manage the remote memory swap mechanism in a RoCE cluster effectively without modifications of system software and operating system to utilize remote memory and interconnected high-speed network bandwidth.

doi:10.6342/NTU201704017

doi:10.6342/NTU201704017

Bibliography

[1] TensorFlow Models. `https://github.com/tensorflow/models`.

[2] B. Abali, R. J. Eickemeyer, H. Franke, C. Li, and M. Taubenblatt. Disaggregated and optically interconnected memory: when will it be cost effective? *CoRR*, abs/1503.01416, 2015.

[3] A. Badam. *Bridging the Memory-Storage Gap*. PhD thesis, October 2012.

[4] A. Basu, J. Gandhi, J. Chang, M. D. Hill, and M. M. Swift. Efficient virtual memory for big memory servers. In *Proceedings of the 40th Annual International Symposium on Computer Architecture*, ISCA '13, pages 237–248, New York, NY, USA, 2013. ACM.

[5] David A Rusling. Linux memory management. `http://www.tldp.org/LDP/tlk/mm/memory.html`.

[6] Dell'Oro Group. Ethernet Switch —Data Center Five Year Forecast Report. `http://www.delloro.com/products-and-services/ethernet-switch-data-center`.

[7] Gilad Shainer. 100 Gbps Headed For The Data Center. `http://www.networkcomputing.com/data-centers/100-gbps-headed-data-center/407619707`.

[8] M. GOTO, M. SATO, K. NAKASHIMA, and K. KUMON. Implementing remote swap memory using rdma over 10gb ethernet. *IEICE technical report. Computer systems*, 106(287):7–12, oct 2006.

doi:10.6342/NTU201704017

[9] J. Gu, Y. Lee, Y. Zhang, M. Chowdhury, and K. G. Shin. Efficient memory disaggregation with infiniswap. In *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17)*, pages 649–667, Boston, MA, 2017. USENIX Association.

[10] InfiniBand Trade Association. The InfiniBand Architecture. `http://www.infinibandta.org/specs`.

[11] E. Kissel, M. Swany, B. Tierney, and E. Pouyoul. Efficient wide area data transfer protocols for 100 gbps networks and beyond. In *Proceedings of the Third International Workshop on Network-Aware Data Management*, NDM '13, pages 3:1–3:10, New York, NY, USA, 2013. ACM.

[12] S. Liang, R. Noronha, and D. K. Panda. Swapping to remote memory over infiniband: An approach using a high performance network block device. In *2005 IEEE International Conference on Cluster Computing*, pages 1–10, Sept 2005.

[13] Linux SCSI Target. LinuxIO. `http://linux-iscsi.org`.

[14] K. Liu, X. Zhang, K. Davis, and S. Jiang. Synergistic coupling of ssd and hard disk for qos-aware virtual memory. In *2013 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, pages 24–33, April 2013.

[15] Max Woolf. Benchmarking TensorFlow on Cloud CPUs: Cheaper Deep Learning than Cloud GPUs. `http://minimaxir.com/2017/07/cpu-or-gpu/`.

[16] H. S. S. Nichole Boscia. Comparison of 40g rdma and traditional ethernet technologies. *NAS Technical Report: NAS*, 01 2014.

[17] RDMA Consortium. An RDMA Protocol Specification (Version 1.0). `http://www.rdmaconsortium.org/home/draft-recio-iwarp-rdmap-v1.0.pdf`.

[18] RDMA Consortium. iSCSI Extensions for RDMA Specification (Version 1.0). `http://www.rdmaconsortium.org/home/draft-ko-iwarp-iser-v1.PDF`.

doi:10.6342/NTU201704017

[19] Red Hat, Inc. Overcommitting with KVM. https://access.redhat.com/documentation/en-US/Red_Hat_Enterprise_Linux/6/html/Virtualization_Administration_Guide/chap-Virtualization-Tips_and_tricks-Overcommitting_with_KVM.html.

[20] RoCE Initiative. RoCE Introduction. http://www.roceinitiative.org/roce-introduction/.

[21] A. Samih, R. Wang, C. Maciocco, T. Y. C. Tai, R. Duan, J. Duan, and Y. Solihin. Evaluating dynamics and bottlenecks of memory collaboration in cluster systems. In *2012 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (ccgrid 2012)*, pages 107–114, May 2012.

[22] M. Saxena and M. M. Swift. Flashvm: Virtual memory management on flash. In *Proceedings of the 2010 USENIX Conference on USENIX Annual Technical Conference*, USENIXATC'10, pages 14–14, Berkeley, CA, USA, 2010. USENIX Association.

[23] M. C. Schatz, C. Trapnell, A. L. Delcher, and A. Varshney. High-throughput sequence alignment using graphics processing units. *BMC Bioinformatics*, 8:474 – 474, 2007.

[24] Stanford Vision Lab. ImageNet. http://image-net.org/.

[25] K. G. Tia Newhall, Sean Finney and M. Spiegel. Nswap: A network swapping module for linux clusters. In *Proceedings of Euro-Par'03 International Conference on Parallel and Distributed Computing (Klagenfurt, Austria, August 2003)*, volume 2790 of *Lecture Notes in Computer Science*. Springer, 2003.

[26] Twitter. twemperf (mcperf). https://github.com/twitter/twemperf.

doi:10.6342/NTU201704017