A Survey on the Challenges of Implementing Physical Memory Pools[*][†]

Heather Craddock[‡], Lakshmi Prasanna Konudula, and Gökhan Kul
Division of Physical and Computational Sciences
Delaware State University, 1200 N. DuPont Hwy, Dover, Delaware, 19901, USA.
{hcraddock, lkonudula, gkul}@desu.edu

Abstract

Cloud computing has been rapidly expanding in the last decade, and has become one of the most heavily researched topics in computing; yet despite significant hardware developments, server architecture maintains a monolithic structure that limits the capabilities of cloud-based systems. Memory limitations force cloud providers to add more monolithic servers to their data centers every day, and demanding software systems may require specially designed servers. In this article, we identify enabling technologies for physical memory pools such as OS design, distributed shared memory structures and virtualization with regards to their relevance and impact on eliminating memory limits, and we discuss the challenges for physical memory pools which can be used by multiple servers.

Keywords: cloud operating systems, hardware virtualization, memory pools, memory protection

1 Introduction

Big data and growing global inter-connectivity requirements are fuelling rapid growth in cloud computing. The need for on-demand resource and data sharing requires cloud systems that can reduce reliance on local resources alone. The Data Never Sleeps report [10] predicts that 1.7 MB of data will be generated every second per person on earth by 2020, and such immense quantities of data require software and hardware architectures that are capable of managing it [14]. It is crucial that these architectures are updated and modified to keep up with changing demands.

The cloud, unlike personal computers, is more akin to public utilities such as water and electricity. The resources are collected together to be used when they are needed by whoever needs them, which allows the resources to be shared and the system to be scaled at need. In a similar way, the cloud enables the the use of shared resources to accomplish tasks that could not easily be completed with local resources alone, and allows for the storage of more data than local storage would allow and enables access to that data by a large number of people. Cloud providers such as Google, Amazon and Microsoft along with many others run and maintain the physical or virtual servers which can be rented for a defined timeframe. Corporations, private individuals, academic institutions, small businesses, governments, and others utilize this service in order to help improve efficiency, performance and accessibility instead of investing large sums of money to create their own computing centers.

Virtualization is a key component of the proper utilization of cloud computing. Virtualization allows for servers to host multiple operating systems (OS) on a number of different virtual machines (VM),

[*]This paper is an extension of "The Case for Physical Memory Pools: A Vision Paper" that appeared in the proceedings of International Conference on Cloud Computing (CLOUD) 2019, San Diego, California, USA.

[†]Both Heather Craddock and Lakshmi Prasanna Konudula should be considered joint first authors.

[‡]Corresponding author: Division of Physical and Computational Sciences, Delaware State University, Dover, Delaware, USA. 19901, Tel: +1(302)857-6640, Web: `http://delawaresec.com/`, Email: hcraddock@desu.edu

allowing for enhanced data migration, recoverability, and improved network productivity. With virtualization, hardware resources can be shared among various VMs. However, running unmodified OSs on VMs generates significant performance cost due to numerous context switches. To combat this, techniques were developed to combine virtualization methods or modify the guest OS to enable the system to use resources more securely and efficiently [5].

Memory hardware improvements have been developing swiftly in terms increased memory for decreased cost [11, 2]. Yet, memory use is often limited by OS policies and OS resource management limitations. Managing the distributed shared memory resources is even more difficult, as the lack of methods and systems developed to coordinate large pools of distributed memory hampers progress [12].

The prevalence and growing importance of the cloud means it is imperative to improve cloud technologies and design flexible system components. The main contribution of this paper is to provide an examination of the challenges of implementing physical memory pools while additionally investigating recent innovations in virtualization, OS design, and distributed memory management. We also discuss improvements that can be made to these proposals, and suggestions as to the future work in and the direction of the field of cloud computing. This paper is an extension of our own work presented in [6].

We begin this paper by defining Physical Memory Pools in Section 2. We then discuss some enabling technologies and techniques for physical memory pools in Section 3; this includes some proposed OS models and approaches to improving virtualization and memory management of such systems. Section 4 presents some security implications and possible methods for memory protection. We discuss the direction of cloud memory models and propose where potential work can be found in OS design, distributed memory, and virtualization, with a brief discussion of the future of the cloud computing field in Section 5. We conclude our study in Section 6.

2 Physical Memory Pools

A *physical memory pool* (PMemP) is a memory unit cluster where each unit does not belong to a single monolithic server, but can instead be used by connected monolithic machines on a need-to-use basis, as shown in Figure 1. We argue that cloud systems should adopt a hardware disaggregation scheme that employs these pools of memory units. This system component is controlled by a *governor* that can prioritize the resource requests from participating servers and allocate the memory to different machines accordingly. This approach is supported in literature by the networked *mComponent* memory units in the LegoOS [16] system. In this instance, the suggested mComponents can act as a physical memory pool. There are several challenges implicit in the hardware resource disaggregation scheme: (1) Limitations of current OS architecture (Sections 3.1 and 5.1), (2) Plug-n-play use of the component (Sections 3.2 and 5.2), (3) Adoption to existing virtual systems (Section 5.3), and (4) Networked memory device speed limitations (Section 5.4).

A Java Virtual Machine (JVM) heap memory allocation scheme can be logically compared to physical memory pools. A JVM uses an allocated memory space when running a Java program. The user may define both a minimum and maximum heap memory allotment, and the program is initialized with the minimum memory quantity. If more memory is required, memory allocation can be increased up to (but not exceeding) the defined maximum heap amount.

Example. We expect that physical memory pools operate in a similar logical manner to JVMs. We designed the following simulation to demonstrate the how a JVM functions:

Firstly, we created a `List` where the following occurred in an infinite loop: 100 elements are added to the list, and random searches are made for 100 elements by list index until we exhaust our memory. Each part of the list is utilized, preventing the use of an efficient swapping function. After running out of memory, a further block of memory must be allocated up to the maximum, in similar behavior to the

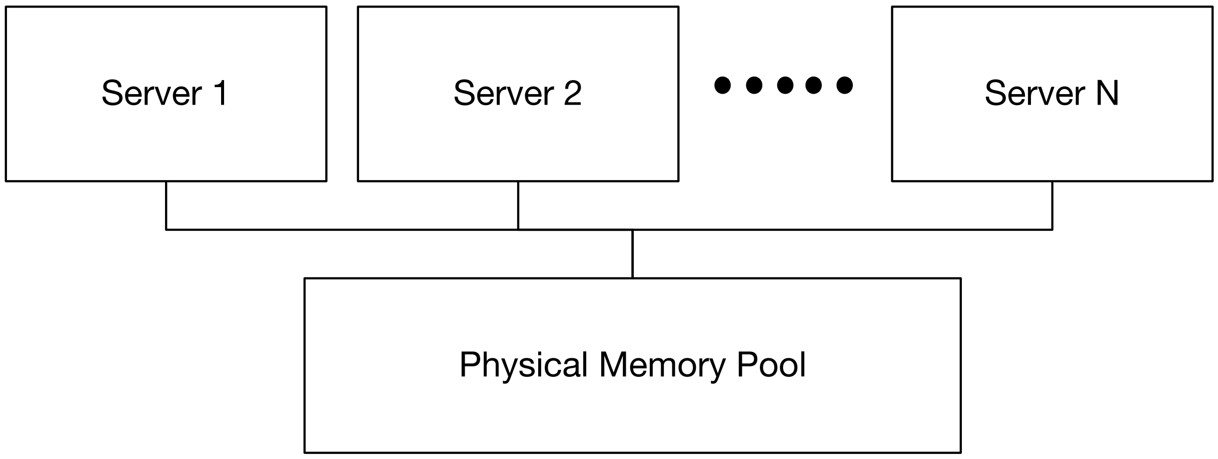


Figure 1: Physical memory pool overview

PMemPs described in Section 2.

The simulation was run on an Apple MacBook Pro (MacOS High Sierra 10.13.6) with 16 GB 2400 MHz DDR4 RAM, a 2.2 GHz Intel Core i7processor, and a 256 GB SSD, and over Java 1.8.0. We varied initial memory allocations of 128 MB, 256 MB, 512 MB, and 1024 MB, respectively, with a maximum allocation of 1024 MB, in order to show the mechanism efficiency and to compare and contrast block allocation and directly using maximum heap memory. The results of this simulation can be seen in Figure 2. We share all the code and documentation on GitHub [1]

Figure 2a shows List growth over time. Predictably, runs using a larger initial heap allocation reach the maximum number of elements faster; however, there is a comparable increase pattern and the time that the JVM spends on garbage collection and adding additional memory blocks must be considered. When approaching the initial heap size the garbage collector operates more aggressively, but new memory blocks must still be added to ensure continuous operation. This behavior can be better seen in Figure 2b. This is the ideal behavior of a memory resource disaggregation system as it provides the advantage of sharing a common memory pool rather than necessitating the building of monolithic servers with large memories.

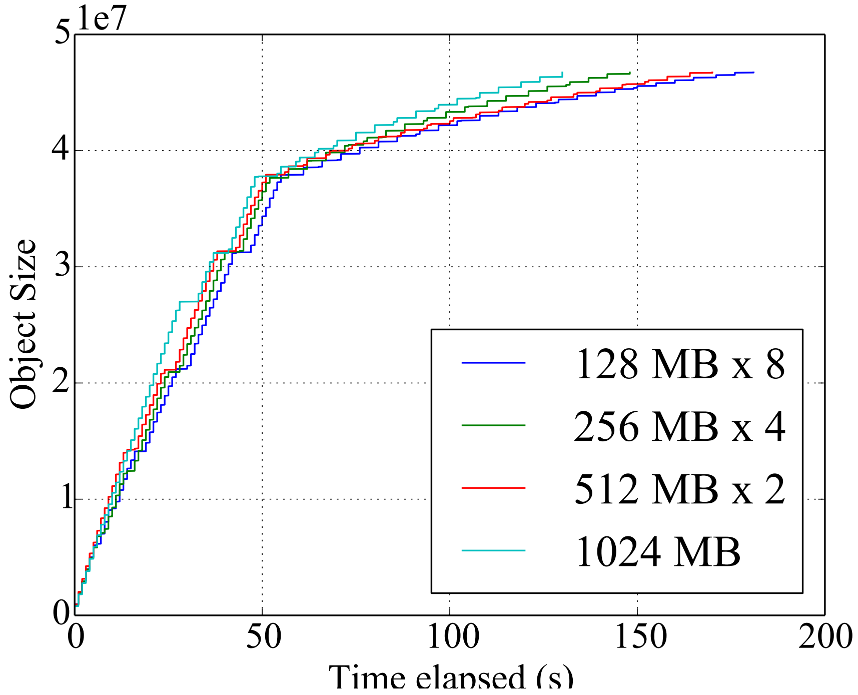
3 Enabling Technologies

Physical memory pools can be implemented with a combination of existing enabling technologies. In this section, we present these methods and discuss how they can be used as an enabling technology.

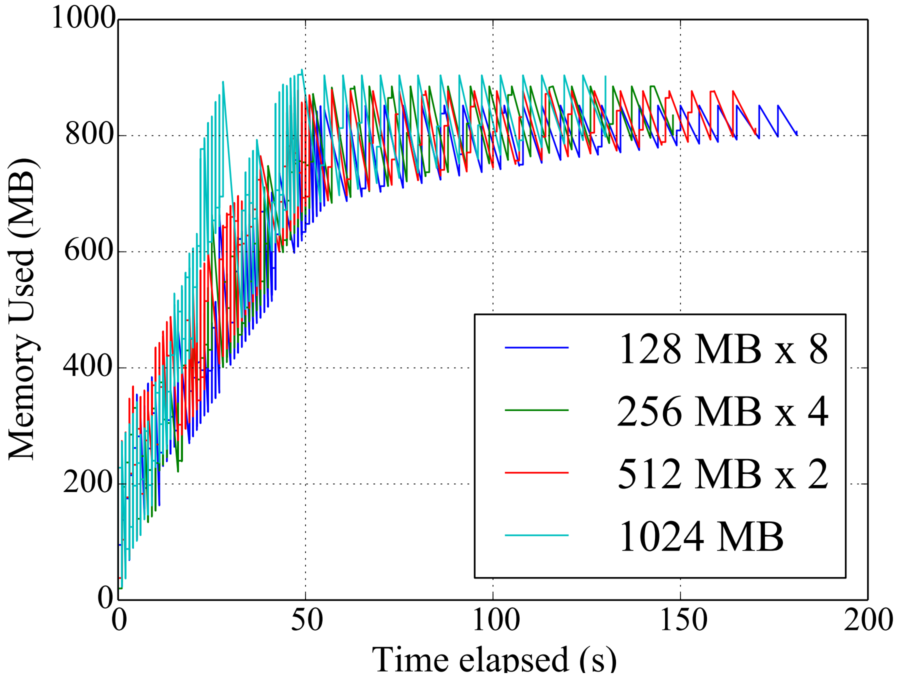
3.1 Operating System Design

Operating systems are an integral part of any computing system, and the increasingly popular cloud environment requires operating systems that conform to the different and ever-changing needs of the platform [14, 20, 17]. Despite this, there has been limited focus on designing operating systems that improve upon models that are outdated for cloud computing [16]. In the literature, several new operating systems have been proposed – Vasilakis *et al.* [19] proposed a new method for OS design for the cloud, and Shan *et. al* [16] proposed LegoOS, a revolutionary OS directly designed to run on disaggregated

[1]Code repository located at https://github.com/PADLab/MemorySwapExperiment



(a) Object count growth in the List



(b) Memory utilization growth

Figure 2: Comparison of allocation under varying memory configurations

hardware components. Swift [18] suggested a method improving upon current operating systems by focusing on improving OS memory management techniques.

3.1.1 Revamping OS Design

The foundations of all popular modern operating systems have been laid out by the works of Hansen [9], Dijkstra [7], and Ritchie & Thompson's UNIX system [15]. As suggested by Vasilakis *et al.* [19], the current OS model is overly complex. They suggest that the current OS design is not supportive of scaling or decentralization – rather than developing a new OS design, newer features have been stacked on the original UNIX model, making it unwieldy, particularly for a system designed for simplicity. They found four specific issues:

- **Data Cataclysm:** The current model is broken, so the development of distributed systems require more than simple changes or updates to current operating systems
- **Reliance on Commodity Hardware:** Inexpensive, unreliable hardware nodes require software-based assurance
- **Rise of the Personal Cloud:** Users own micro-clouds, not accounted for in the original model
- **Data Processing Shift:** Data processing is not exclusive to experts any more, and can be done by anyone

Vasilakis *et al.* [19] try and combat these issues by proposing a distributed system design with some key components; it requires a scaling-tolerant file system, an innovative execution primitive, and the use of sandboxing to shift assurance to the software layer. The system they proposed is reliant on the programming language that it would be implemented in, much like UNIX is based on C. However, the language has not yet been created, so the design must be practically implemented with a new or adapted programming language for the design's quality to be substantiated. In addition, the authors suggested that using automatic memory management for garbage collection would limit the vulnerabilities inherent in manual memory management. However, garbage collection comes with other issues; it can be harder to judge performance and memory usage, and it is important to note that the garbage collection technique only deals with memory resources and not other resources. These issues would also need to be addressed in a practical implementation. Even so, this system design is important in bolstering the discussion about OS design for the cloud.

Furthermore, Shan *et. al* [16] designed LegoOS. This OS is built with *splitkernel architecture* which appoints key hardware-software interfaces to hardware monitors to support disaggregated components. The authors stated that these disaggregated components would become more important as cloud computing moved away from the use of monolithic servers. As can be seen in Figure 3, each hardware component is monitored separately, reducing the dependency on strict coupling of hardware, and improving failure resistance.

3.1.2 OS Design for Near Data Processing (NDP)

The development of NDP architecture called for improvements to be made upon more traditional memory models. Through NDP, close proximity to memory models was shown to reduce power consumption while enhancing throughput, but OS support is required to deal with low-latency, protection, and data locality. Barbalace *et al.* [4] introduced Shadowgraphy, an OS designed on the following principles: (1) Multi-kernel design optimizing communication through shared memory, allowing for interaction between IO devices and the OS while running on separate processing units; (2) Enforced cross-kernel user

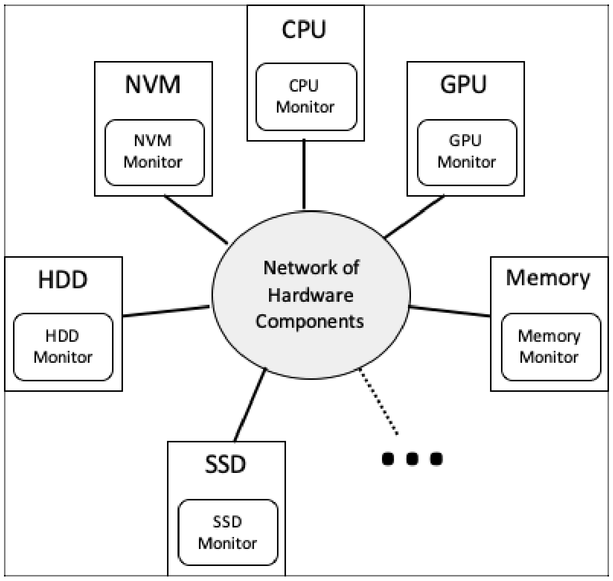


Figure 3: LegoOS design

privilege protections, whereby different applications in the NDP processing unit can maintain differing levels of user privilege irrespective of where the application is running; (3) Local scheduling at each kernel based on data access patterns; and (4) Data migration enabled by efficient hardware and software memory caching.

The authors affirm that system software must be developed for NDP, beginning with the OS. They discuss the disadvantages of offloading while multiple applications are running, but there is no mention of how Shadowgraphy can present a solution for this problem. The authors also discuss how transparency for application developers can be achieved through asymmetry in the processing units and multiple levels of OS interface, and it is therefore important to propose suitable multi-server, multi-kernel designs than allow multiple users to concurrently access the system. Any new OS designed should also support NDP and CPU architecture to allow for efficient code migration.

3.1.3 Distributed Memory Techniques

Distributed Shared Memory (DSM) is a memory architecture whereby computing tasks can be shared among multiple separate hardware components. Through DSM, this distributed physical address space can be logically addressed as if it was all local space. In this manner, the different memory components of separate processing nodes can be networked together to create a larger memory pool. Recent literature attempts to address the issues with current Distributed Shared Memory techniques.

3.1.4 Scalability in Data-Intensive Applications

Data-intensive applications are commonly run on the current common hardware configuration where a high-bandwidth network connects many different nodes; this configuration inhibits the scalability of the systems. Nelson *et al.* [12] propose the Grappa system, a cluster-based DSM designed to improve performance on data-intensive applications. Grappa's primary contribution is derived from implementing parallelism in order to maximize process resource use while disguising message-sending latency and communication costs as can be seen in Figure 4. Implementing distributed computing in parallel allows the system to bypass some of the flaws of traditional scaling methods where locality of data and caching

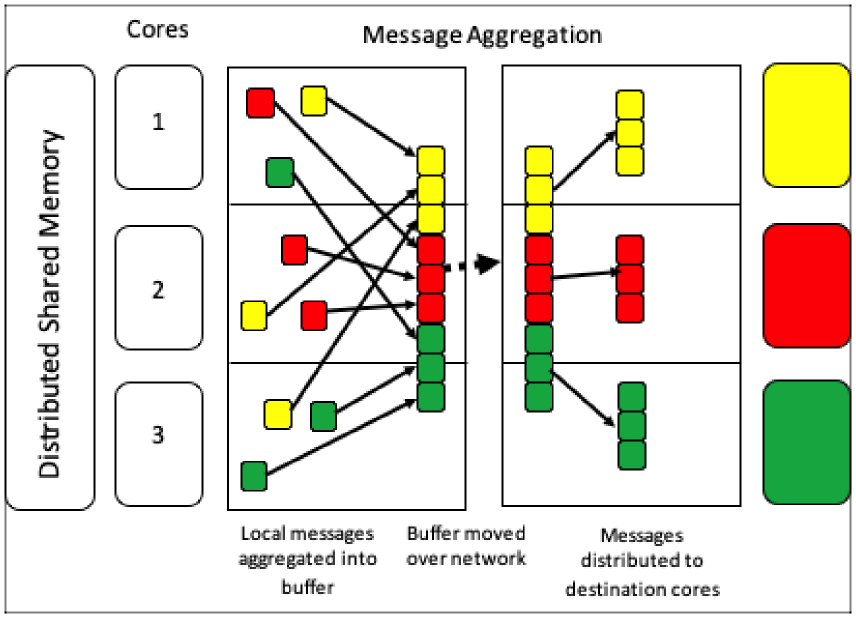

Cores
Message Aggregation
Distributed Shared Memory
1
2
3
Local messages aggregated into buffer
Buffer moved over network
Messages distributed to destination cores

Figure 4: Grappa design

were required to scale effectively. The authors' proposal allows the high-bandwidth network costs of traditional hardware systems to be disguised. There are three key components of Grappa:

- Distributed Shared Memory - local data can be exported to the global address space in order to be accessed anywhere in the system. Unnecessary retrieval or sending of data over the network is limited by performing operations at the data's home node, also guaranteeing global order and memory consistency.
- Tasking System - Parallelism and load-balancing is enabled through multi-threading and work-stealing to allow for better utilization of system resources. Worker threads execute individual tasks, and longer-latency operations yield the processing core to maintain maximum processor utilization.
- Communication Layer - Network bandwidth use is limited through small message aggregation.

There are potential improvements to be made in the Grappa system, particular in the area of failure recovery. Currently, recovering from failure is more costly than restarting the system entirely, sacrificing fault-tolerance for scalability. It may be worth considering how to improve the system so that it becomes more useful to recover the system than restart. Furthermore, current hardware limits the sending of small messages through Grappa, but this system should be revisited as hardware innovations could lead to further improvements to network latency.

3.1.5 Improving Memory Access Speed

Distributed memory relies heavily on fast memory access, but although decreasing hardware costs means that constructing clusters with large memory resources is cheaper than ever, memory access speed is limited by the network latency between the separate hardware elements of a cluster. The authors of [8] propose methods that mitigate this issue by improving memory access speeds over distributed hardware components.

Countering the slow memory access caused by the use of TCP/IP protocol in cluster systems is a vital component of improving memory access speed. Remote Direct Memory Access (RDMA) is an existing technique allowing one machine to directly access the memory of another without involving the operating system by allowing the NIC to circumvent the remote CPU and kernel to allow direct data access. FaRM builds upon this system by enabling direct access to data through one-sided RDMA reads. This technique improves message-passing speed, enhancing application performance. FaRM also ensures in-order transactions through the use of lock-free reads. The RDMA technique achieves high throughput and low latency, but it achieves remote meory access using the NIC rather than the CPU. The I/O operations will not therefore go through the CPU, which leaves potential for the CPU to lose control of the data. Data loss or error may occur if a transaction occurs and there is a failure to determine if there is sufficient storage space. This issue could be mitigated by reserving adequate storage space in the preparation; however, there is still potential for improvement.

3.1.6 Server Load Imbalance

Novakovic *et al.* introduced in [13] the RackOut memory pooling technique designed to improve access speed in clusters through direct access. Most large-key value stores maintain data in the server memory to provide high throughput and low latency, but skew can limit performance. Skew can cause load imbalances, leading to poor datacenter utilization; there were no solutions to the skew issue that did not incur further large overheads, and so the authors state it was paramount to create a system that could manage this load imbalance while alose meeting its other objectives.

RackOut minimized server load imbalance by allowing nodes in a rack to access the memory of other nodes without having to use the remote CPU. This was possible through the implementation of RackOut on a server group with the following qualities: (1) High internal bandwidth; (2) Low-latency communication fabric; and (3) Direct memory access of distributed nodes through one-sided operations. With this technique, data is only replicated outside the rack when necessary, and there is fast memory access within the rack. Operations can be balanced more evenly across the rack through the increased speed of memory access and operation-sharing between nodes. A limitation of this technique is the latency of the communication fabric; this may be mitigated by the technological trend towards lower-latency fabrics. While the RackOut system seemed to have a lot of potential, the scalability is in doubt without further exploration, as the scalability tests were limited by research resources. A more practical impact might be generated if this system could be tested on larger-scale datacenters.

3.1.7 Limited Discussion on Remote Memory

The authors of [3] list a various number of different cloud computing areas that could present challenges for researchers. It can be difficult to establish a direction for future work in a field without updating the current body of knowledge and opening the topic for discussion. The authors believed that the discussion on remote memory was out of date, as when remote memory was proposed twenty years ago there were more hardware and software limitations in implementing remote memory solutions. Despite the technological advances in the intervening time, discussion on efficient realization of remote memory is limited, and a number of challenges and potential solutions were proposed:

- Remote host crashes: 1) Allow the application to provide failure handlers; 2) Mask failure through redundancy with replication or erasure coding

- Slow or congested network: 1) Pre-allocate remote memory network bandwidth and prioritize the network traffic; 2) Assign different regions of memory to applications

- Virtual memory overheads: Check and rebuild the subsystem of the virtual memory
- Virtual machine indirection: Extract information about applications using the hypervisor
- Transparency level: Design the remote memory in different cases.
- Sharing model: When data sharing is prohibited (e.g. private data), limit the remote memory to the private data.
- Lack of write ordering across hosts: 1) Appropriate protocols and DSM can enforce costly ordering; 2) Applications to use the memory of remote hosts; 3) Allow for re-ordering
- Non-uniform latency: 1) Use the existing operating system mechanisms for NUMA; 2) Allow applications to view memory speed and adjust themselves accordingly
- Remote host compromised: 1) Encrypt remote memory data; 2) Compensate for larger attack surface with improved whole-system security
- Local vs. cluster memory: Statically reserve local memory while leaving the rest free as cluster memory
- Remote memory allocation: Centralize allocations through a memory-management host
- Memory placement: Centralize the memory placement mechanism
- Local memory management: The physical memory host should complete local memory management, with awareness that there may be overheads on RDMA-based NICs
- Control plane efficiency: Experiment using off-the-shelf control plane solutions
- Memory metadata overhead: manage remote memory in slabs larger than pages to limit metadata requirements

A number of issues and potential solutions are introduced in [3], but there are some key issues that were overlooked. Security problems were not covered in any detail, yet when considering remote memory we have to consider the security of data at the local machine level as well as at the network and remote machine level. Suitable security measures may increase cost but can potentially be implemented in a number of ways: (1) Prioritizing the machines storing the bulk of the data; and (2) Secure and protect network access. Ensuring that individual machines are secured correctly would limit the ability of an attack on a single host compromising the entire system.

3.2 Improving Memory Virtualization

In modern computing, virtual memory plays an important role. It increases the programmer's productivity and provided additional security benefits. Current workloads suffer from virtual memory contiguity overheads requiring high execution time. Jayneel Gandhi *et al.* [3] as an attempt to solve this problem, proposed Redundant Memory Mapping(RMM).

The paper proposes a hardware/software co-design, Redundant Memory Mapping which can map contagious virtual pages to physical pages. RMM implementation is robust and transparent, overcoming the alignment restrictions and overheads while the flexibility of the paging is retained. Authors attempt to address the principal problem of using page table in the paper. Using 124 range translations from the range table, a translation look-aside buffer miss can be overcome. Using 124 range translations from the range table can overcome a TLB mistake. The paper presented few evaluations showing that for all configurations and workloads RMM works.

Although RMM eliminates the vast majority of page walks, using eager paging may increase latency, which in turn can induce fragmentation. Performance is heavily impacted by latency and fragmentation. Also, additional hardware and software are required in implementing RMM, which may involve future development. The potential solution to the stated problem is retrieving data in parallel during translation. This provides a way to store low-latency huge data sets for real-time data analysis.

3.3 Efficient Memory Management

In the system design, memory capacity is a key limitation. With giant strides in-memory technology, the systems have an abundance of large memories at much lower costs. Existing operating system designs, however, are ill-equipped to deal with large memories. In order to manage these large memories, a guiding design principle of Order (1) operations [18] is proposed. With Order (1), memory operations will be performed in constant time independent of the size of the operand. The basic principle is to handle memory management using file-system techniques. Order (1) performance is provided by enabling the operations on the whole file instead of individual pages. Exposing the data directly to the programs reduces the complexity of the system as the memory already has data residing in it. By using file systems for memory management is much convenient as they are capable of handling large memories, maintaining metadata and large address translation. The memory layer above the file is removed, allocating the user memory as files with the backup of tmpfs in O (1) memory. In this design, they count references to files while the references to pages are ignored. When a process terminates or unmaps the memory can be reclaimed. In order to improve the efficiency in memory mapping, pointers are used. With Order (1) memory, there is no need to swap between the disks which in turn eliminates the need for overhead and clean bits tracking. Range translations are used in Order (1) memory in order to curtail the memory access cost.

There are many limitations for implementing O (1) memory. Operations that rely upon page level mappings cannot be simply supported and are troublesome to optimize. Utilizing persistent memory to store volatile data makes the system complex. This, in turn, can cause leakages due to breaking in the isolation between user and kernel space. The only way to avoid the situation is by zeroing the memory before it is being reused.

4 Security Implications

The development of extensive main memories which are non-volatile and running large scale services on rack-scale computers creates significant challenges based on MMU mechanisms for memory protection. Protection is put into stake while trying to optimize for performance. Some challenges may include memory corruption due to stale locations, nested pages, and hypervisor calls.

LegoOS [16] assumes that, threads belonging to different processes will not have access to writable shared memory, which follows the general OS principles of memory protection. This assumption simplifies scheduling decisions and the overall architecture. Considering that writable shared memory is rarely needed if message passing among processes is possible, this limitation is not a major flaw. However, in terms of performance, message passing may take significantly more time depending on the system architecture.

To address the above challenges, Achermann *et al.* [1] proposed “Matching Key Capabilities(MaKC)”, an architecture capable of scaling memory protection at both the user and kernel level. MaKC is associated with a Block Protection Key (BPK) and Execution Protection Key (EPK) to divide memory hierarchies into equal size blocks. When there is a match between the BPK and the EPK then memory access is allowed, else it is blocked. With HMACs (Hash Message Authentication Codes) it is guaranteed that in the transit messages are not forged nor manipulated. Protection tables containing BPKs are implemented with a key matching strategy using hardware read and cache. MaKC has the capacity to handle protection in complex memory hierarchies. The MaKC approach also allows enabling huge pages without compromising the security of small page sizes.

MaKC design is not completed, the decision has to be made on the implementation of features on CPU-side or memory-side. MaKC implementation could lead to some potential issues:

- There is a space overhead caused by storing a large number of HMACs and keys. Also, it is expensive to access the keys and HMACs by entering into supervisor state.
- Complexity is added with the usage of MaKC.
- Using cryptographic keys in MaKC to authenticate the fingerprints could increase security management complexity .
- The model results in poor memory management as it uses fixed-size blocks for inefficient memory use.

5 Research Directions for Disaggregated Memory in the Cloud

In this section, we discuss how improvements to OS design, distributed shared memory, and virtualization can all contribute to the furtherance of the research on physical memory pools, while also considering the direction of cloud computing in general.

5.1 OS Design

OS design can be restricted by the limited availability of a suitable programming language for implementation, as posed in [19]. The OS should allow users to inspect topology of the CPU and NDP, as well as other information about the platform, by providing a transparent environment to the application developer.

In the future, a cloud OS must be dynamic and multi-functional in order to adjust to expanding requirements. Designed models should allow concurrent access for multiple users, in addition to supporting asymmetricity in the processing units; this can be implemented through a multiple-kernel, multi-server design.

5.2 Distributed Memory

Distributed system bottlenecks can occur in a number of places, including during address translation and due to networking hardware components. Continued hardware developed must be made to limit this bottleneck on new system proposals. Current constraints could be diminished in a number of ways; By increasing both the network and processing capacity of distributed nodes, developing larger and more capable nodes, using aggregation to create larger entities from individual nodes, and performing in-memory computing in order to execute the translation and data-fetch together.

Models are more frequently employed on fault-prone hardware. The importance of fault-tolerance in systems such as Grappa proposed in [12] requires that progress must be made in ensuring systems are able to recover from faults rather than a fault necessitating a restart.

5.3 Virtualization

Virtualization is advantageous in cloud computing as it allows for flexibility, scalability, and cost-effective implementation. It is beneficial as it adds a layer between the distributed hardware components and the applications running on them; however, virtualization can also add processing overheads, slash application performance, and introduce security vulnerabilities into the system. These limitations can be countered short-term by gravitating toward in-memory computing and allowing for isolation between the kernel and user space, with an eye towards developing operating systems designed for virtualization.

In the future, growing workloads should utilize range translation and in-memory computing in order to better employ the increasing physical memory that is available. In this way, large, stored data sets can leverage resources for low-latency and real-time analysis.

5.4 Network

There are still certain difficulties inherent to current network technology that are still being researched. The first issue is that network components and protocols carry communication overheads that can limit distributed performance. This can be simply addressed in a hardware communication environment by isolating the hardware network from the communication network. The second concern is the speed limitation engendered by network bandwidth constraints. This is trending toward being a minor issue due to the drastic and consistent developments to network technology over the course of the past decade.

5.5 Direction of Cloud Computing

There are several questions to consider regarding the direction of cloud computing:

- Complexity management: How is it possible to manage ever-complex, increasingly interconnected systems of microservices? The microservices are often provided by a variety of service providers, and it is both important and difficult to maintain performance and security through diverse structures. Furthermore, many of the services are dependent on vital infrastructure that is run by other teams or companies which demands a considerable level of trust; methods must be developed that ensure trust and reliability from outsource services.

- Failure tolerance: When systems are heavily involved in vital societal functions such as health, business, productivity, and safety, it is paramount that the systems are failure-tolerant. Many of these key functions utilize the cloud, and thus ensuring the failure-tolerance of networked distributed systems must be a priority; how can it failure-tolerance be guaranteed?

- Virtualization: Virtual systems can allow for better resource-sharing between distant, distinct hosts. How can operating systems be refined, adapted, or developed to further support the secure and efficient virtualization of services?

- Environmental and financial impact: Increased distribution prompts the rise of giant, always-on data centers that provide the cloud services. The energy consumption of such data centers leads to significant cost to both the environment and the service provider. How can these costs be limited? Virtualization may help as it allows for the resources to be geographically distributed, and thus the impact isn't centralized to one location. Future work may involve research improving workload distribution, not just to ensure the efficiency of operations but to ensure that demanding tasks are completed as energy-efficiently as possible on the most energy efficient hardware components.

6 Conclusion

This paper considered the contributions of recent literature and their impact concerning innovations in the cloud. The works included papers introducing specific models, such as RackOut, LegoOs, Shadowgraphy OS, and Grappa, and also more theoretical contributions to cloud computing as presented in [3]. It is important to gain a thorough grounding in the current state of research before significant advancements can be made, and thus the core contribution of this paper is to provide potential future research directions around independent physical memory pools that can be shared, allocated, and de-allocated by datacenter servers at need. It is important to note that this survey is by no means comprehensive. Although there were suggestions made as to improvements to be made or future work to be completed, the primary theme of the reviewed works was that the discussion on cloud computing is really only just beginning, leaving so much more still to explore in the field.

Acknowledgments

This material is based in part upon work supported by the funding provided by the State of Delaware to Delaware State University. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the University or the State.

We would also like to thank Kun Cheng from Delaware State University for his valuable contributions throughout the survey process, his reviews of related papers, and his role in preparing the CLOUD 2019 paper that laid out the foundations of this article.

References

[1] R. Achermann, C. Dalton, P. Faraboschi, M. Hoffmann, D. Milojicic, G. Ndu, A. Richardson, T. Roscoe, A. L. Shaw, and R. N. M. Brasche. Separating translation from protection in address spaces with dynamic remapping. In *Proc. of the 16th Workshop on the Hot Topics in Operating Systems (HotOS'17), Whistler, British Columbia, Canada*, pages 118–124. ACM, May 2017.

[2] N. Agarwal and T. F. Wenisch. Thermostat: Application-transparent page management for two-tiered main memory. In *Proc. of the 22nd International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS'17), Xi'an, China*, pages 631–644. ACM, April 2017.

[3] M. K. Aguilera, N. Amit, I. Calciu, X. Deguillard, J. Gandhi, P. Subrahmanyam, L. Suresh, K. Tati, R. Venkatasubramanian, and M. Wei. Remote memory in the age of fast networks. In *Proc. of the 2017 Symposium on Cloud Computing (ACM SoCC'17), Santa Clara, California, USA*, pages 121–127. ACM, September 2017.

[4] A. Barbalace, A. Iliopoulos, H. Rauchfuss, and G. Brasche. It's time to think about an operating system for near data processing architectures. In *Proc. of the 15th Workshop on the Hot Topics in Operating Systems (HotOS'17), Whistler, British Columbia, Canada*, pages 56–61. ACM, May 2017.

[5] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield. Xen and the art of virtualization. *ACM SIGOPS Operating Systems Review*, 37(5):164–177, December 2003.

[6] H. Craddock, L. P. Konudula, K. Cheng, and G. Kul. The Case for Physical Memory Pools: A Vision Paper. In *Proc. of the 2019 International Conference on Cloud Computing (CLOUD'19), San Diego, California, USA*. Springer, June 2019.

[7] E. W. Dijkstra. The structure of the "the" multiprogramming system. *Communications of the ACM*, 11(5):341–346, May 1968.

[8] A. Dragojević, D. Narayanan, O. Hodson, and M. Castro. FaRM: Fast Remote Memory. In *Proc. of the 11th USENIX Symposium on Networked Systems Design and Implementation (NSDI'14), Seattle, Washington, USA*, pages 401–414. USENIX Association, April 2014.

[9] P. B. Hansen. The nucleus of a multiprogramming system. *Communications of the ACM*, 13(4):238–241, April 1970.

[10] J. James. Data Never Sleeps 6.0. Technical report, Domo, Inc., June 2018.

[11] S. Mittal and J. S. Vetter. A survey of software techniques for using non-volatile memories for storage and main memory systems. *IEEE Transactions on Parallel and Distributed Systems*, 27(5):1537–1550, May 2016.

[12] J. Nelson, B. Holt, B. Myers, P. Briggs, L. Ceze, S. Kahan, and M. Oskin. Latency-Tolerant Software Distributed Shared Memory. In *Proc. of the 2015 USENIX Annual Technical Conference (USENIX ATC'15), Santa Clara, California, USA*, pages 291–305. USENIX Association, July 2015.

[13] S. Novakovic, A. Daglis, E. Bugnion, B. Falsafi, and B. Grot. The Case for RackOut: Scalable Data Serving Using Rack-Scale Systems. In *Proc. of the 7th ACM Symposium on Cloud Computing (SoCC'16), Santa Clara, California, USA*, pages 182–195. ACM, October 2016.

[14] F. Pianese, P. Bosch, A. Duminuco, N. Janssens, T. Stathopoulos, and M. Steiner. Toward a cloud operating system. In *Proc. of the 2010 IEEE/IFIP Network Operations and Management Symposium Workshops (NOMS'10), Osaka, Japan*, pages 335–342. IEEE, April 2010.

[15] D. M. Ritchie and K. Thompson. The unix time-sharing system. *Bell System Technical Journal*, 57(6):1905–1929, 1978.

[16] Y. Shan, Y. Huang, Y. Chen, and Y. Zhang. Legoos: A disseminated, distributed OS for hardware resource disaggregation. In *Proc. of the 13th USENIX Symposium on Operating Systems Design and Implementation (OSDI'18), Carlsbad, California, USA*, pages 69–87. USENIX Association, October 2018.

[17] J. Smets-Solanes, C. Cérin, and R. Courteaud. Slapos: A multi-purpose distributed cloud operating system based on an erp billing model. In *Proc. of the 2011 IEEE International Conference on Services Computing (SCC'11), Washington, DC, USA*, pages 765–766. IEEE, July 2011.

[18] M. M. Swift. Towards O(1) Memory. In *Proc. of the 16th Workshop on the Hot Topics in Operating Systems (HotOS'17), Whistler, British Columbia, Canada*, pages 7–11. ACM, May 2017.

[19] N. Vasilakis, B. Karel, and J. M. Smith. From Lone-Dwarfs to Giant Superclusters: Rethinking Operating System Abstractions for the Cloud. In *Proc. of the 15th Workshop on the Hot Topics in Operating Systems (HotOS'15), Kartause Ittingen, Switzerland*, pages 15–15. ACM, May 2015.

[20] D. Wentzlaff, C. Gruenwald, III, N. Beckmann, K. Modzelewski, A. Belay, L. Youseff, J. Miller, and A. Agarwal. An operating system for multicore and clouds: Mechanisms and implementation. In *Proc. of the 1st ACM Symposium on Cloud Computing (SoCC'10), Indianapolis, Indiana, USA*, pages 3–14. ACM, June 2010.

Author Biography

Heather Craddock is a Master's student at Delaware State University. She acts as both an undergraduate teaching assistant and as a research assistant in the Cybersecurity research lab, with a main research focus on insider threat detection. She has been published at the International Conference on Cloud Computing. Prior to attending DSU, she received her B.S. in Computer Science and B.S. in Mathematics from Mississippi Valley State University.

Lakshmi Prasanna Konudula is a second-year master's student specializing in Computer Science track at Delaware State University. She is currently a Graduate Research assistant at Delaware State University. Her research broadly covers detection of cybersecurity threats and database systems. She received her bachelor's degree in Information Technology from Jawaharlal Nehru Technological University, Kakinada, India in 2015. She was a Data Analyst at Dell International Services for 2 years. Her major goal is to develop techniques using AI and machine learning to detect and respond to cybersecurity threats efficiently. Her interests include Information Security, Data Analytics, large-scale analyses of software vulnerabilities and their fixes, and the development of systems to assist with remediation. Her recent publication has been accepted by CLOUD 2019 (Research Track of SCF 2019).

Gökhan Kul is an assistant professor at Delaware State University. His research broadly covers database systems and cloud systems security. He has published at reputable journals and conferences such as IEEE TKDE and IEEE TrustCom focusing on data leakage, concept drift, and threat detection. He contributes to research reproducibility efforts at VLDB and SIGMOD reproducibility committees. He also leads the efforts for acquiring the NSA/DHS National Center of Academic Excellence in Cyber Defense designation for Delaware State University. Prior to joining DSU, he received his Ph.D. in August 2018 at the University at Buffalo, where he taught graduate level courses on Database Systems. He received his M.S. degree at METU in Turkey while working as a software engineer at METU Computer Center.