FIRST LINE OF THE TITLE
SECOND LINE OF THE TITLE

by

Author
A Thesis
Submitted to the
Graduate Faculty
of
George Mason University
In Partial fulfillment of
The Requirements for the Degree
of
Master of Science
Discipline

Committee:

_____ Dr. First Last, Thesis Director

_____ Dr. First Last, Committee Member

_____ Dr. First Last, Committee Member

_____ Dr. First Last, Department Head

_____ Dr. First Last, Dean

Date: _____ X Semester Year
George Mason University
Fairfax, VA

The Complete Title is to be Repeated Here without any Line Breaks for the Second Page and for the Abstract Page

A thesis submitted in partial fulfillment of the requirements for the degree of Master of Science at George Mason University

By

Author
Bachelor of Science
My Other Former School, Year of first degree

Director: Dr. First Last, Professor
Department of Name of Department

X Semester Year
George Mason University
Fairfax, VA

Dedication

I dedicate this dissertation to ...

Acknowledgments

I would like to thank the following people who made this possible ...

Table of Contents

List of Tables

List of Figures

Abstract

THE COMPLETE TITLE IS TO BE REPEATED HERE WITHOUT ANY LINE BREAKS FOR THE SECOND PAGE AND FOR THE ABSTRACT PAGE

Author, MS

George Mason University, Year

Thesis Director: Dr. First Last

Enter abstract text.

Chapter 1: Introduction

1.1 Motivation

Serverless computing is an increasingly popular cloud execution model that liberates application developers from the burden of traditional infrastructure management. With serverless platforms (e.g., AWS Lambda, Google Cloud Functions, Azure Functions), developers solely focus on writing their code as event-driven functions that will execute on-demand in response to events or triggers. Cloud providers are responsible for dynamically allocating and scaling resources to meet demands as the event triggers occur. With a pay-as-you-go pricing model, users only pay for the resource consumed during their function invocations, making serverless computing a cost-effective solution.

Cloud providers designed serverless functions to be stateless, meaning that they do not retain state between function invocations. This intentional statelessness is a fundamental aspect for achieving high elasticity. By eliminating the need to store state within the function invocation, serverless platforms promote scalability and ease of deployment. Cloud providers can execute functions in parallel, allowing for efficient resource utilization. Any data needed between function invocations must be stored in remote storage.

Although the stateless nature of serverless computing is key to achieve high elasticity, it limits the type of applications that can run efficiently on serverless platforms. Previous studies [1] have found that data-intensive applications running in serverless platforms (i.e., data analytics, ML workflows, databases) are limited by the capacity and performance gaps that exist among the existing storage services. Object storage services, such as AWS S3, provide cheap long-term storage, but exhibits high access latencies. On the other hand, in-memory clusters, such as AWS ElastiCache, exhibit low access latencies and high throughput, but they are expensive and are not transparently provisioned. In between,

key-value databases, such as AWS DynamoDB, provide high throughput, but are expensive and can take a long time to scale.

Given the limitations of existing storage solutions, previous works motivate the development of a serverless storage service capable of handling the wide variety of workloads running on serverless platforms. These studies mention three requirements that such service must meet. First, it should provide low latency and high throughput for a wide range of object size and data access patterns. Second, it should be transparently provisioned and should scale to meet workload demands. Third, it must ensure isolation and predictable performance across applications and tenants.

To meet the first requirement, cloud providers must first close the capacity and performance gap between main memory and persistent storage media. As mentioned above, existing storage service have fixed tradeoffs that reflect the traditional memory hierarchy built from RAM, flash memory, and magnetic disk drives. Leveraging Non-volatile memory is a promising approach to bridge the gap between the memory and storage tiers. Non-volatile memory combines the persistence and capacity of traditional storage with the low latency and byte addressability of main memory. This technology experienced a breakthrough with the release of Intel Optane DC Persistent Memory.

Non-volatile memory technology experienced a breakthrough with the release of Intel Optane DC Persistent Memory Module (PMM). Optane PMM is an emerging technology where non-volatile media is placed in a Dual In-Line Memory Module (DIMM) and installed on the memory bus, alongside traditional DRAM (Dynamic Random Access Memory) []. Similar to DRAM, this technology presents a byte-addressable interface and achieves speeds comparable to DRAM (2x-3x lower). The main difference between the two is that Optane PMM has higher capacities and can retain data when the system is shutdown or loses power. This allows Optane PMM to be used as a form of persistent storage with memory-like speeds.

The unique combination of persistence and low access latency makes Optane PMM an ideal candidate to speed up data-intensive workloads running in serverless platforms. Thus,

thesis presents an analysis on how to make efficient use of Optane PMM to build a serverless storage service.

1.2 Research Questions

With the release of Intel Optane DIMM, researchers have started to understand its characteristics, capabilities, and limitations. The initial expectation was that Intel Optane DC PMM would behave similar to DRAM, but with a lower performance (higher latency and lower bandwidth). However, recent studies suggest that it should not be treated as a "slower, persistent DRAM". Compared to DRAM, Optane DC PMM exhibits complicated behaviors and its performance changes based on multiple factors, such as the access size, access type (read vs. write), and degree of concurrency.

Intel Optane DC PMM differs from DRAM in two ways. First, there is a mismatch between the CPU cacheline access granularity (64-byte) and the 3D-XPoint media access granularity (256-byte) in Intel Optane DC PMM. This difference can lead to write or read amplification if the data access is smaller than 256 bytes. Second, to balance the gap in access granularity, the Intel Optane DC PMM implements a small (16KB) write-combining buffer to merge small writes and reduce write amplification. However, the buffer's limited capacity (16 KB) can cause contention within the device, limiting its ability to handle access from multiple threads simultaneously.

The complex behavior of Intel Optane DC PMM introduces interesting challenges for building a serverless storage service using this technology. Previous works have found that serverless functions vary considerable in multiple ways, including the way they access and process data, and their quality-of-service (QoS) demands. Furthermore, these workloads can spike by orders of magnitude and change dramatically over time. Knowing how these large-scale variations affect the system's performance and QoS for applications can assist in building an efficient serverless storage service.

Consequently, this thesis addresses the following research questions:

- How does Optane PMM affect the system's performance when used as persistent storage for serverless functions?
- How does Optane PMM performance under serverless workloads affect the (QoS) for applications?
- How can we overcome the limitations of Optane PMM to make efficient use of the device in a serverless scenario?
- How do we keep the system optimized and compliant with QoS requirements over time as workload shifts occur?

1.3 Contributions

The experiments described in Section 3 provide various helpful insights on the Optane PMM behavior when used as persistent storage for serverless workloads. First, we discover that sharing the Optane PMM among hundreds of serverless functions lead to performance loss (higher latency and lower bandwidth) in the device. This fact was expected given the contention issues experienced by Optane PMM with higher thread counts. Second, we discover that, depending on the workloads, the performance degradation in Optane PMM affects one performance metric more than the other (latency vs. bandwidth). This suggests that QoS of some applications might be affected more than others. Therefore, we conclude the success of Optane PMM should be measured by its capability of meeting the QoS requirements of the current workload.

To help alleviate the limitations of Intel Optane PMM, we introduce a control layer that runs on top of Optane and guides the efficient use of the device under dynamic workloads. Our control layer, called NVM Middleware, is designed to limit the access to persistent memory to reduce its contention. While doing so, the NVM Middleware keeps track of the type of applications running in the system and applies different optimization policies for each one to ensure that their QoS requirements are met. Using machine learning, the

NVM Middleware learns how to scale resources to meet the current demand and dynamically adapts them to changing workloads. We propose using online reinforcement learning algorithms, given that data access patterns in serverless workloads can change over time.

- We present an experimental study that describes the capabilities and limitations of Intel Optane PMM when used as persistent storage for serverless workloads. To our knowledge, Optane PMM has not been tested yet in this scenario.
- We present the NVM Middleware, a control layer promotes the efficient use of Optane PMM, while ensuring that QoS requirements for different type of applications are met.
- We propose a Reinforcement Learning model and framework that allows the NVM Middleware to learn from historical data and adapt resources to changing workloads.
- Finally, we present empirical results that demonstrate the benefits of our solution.

Chapter 2: Background and Methodology

2.1 Intel Optane DC Persistent Memory

Persistent memory, also known as Non-volatile Memory (NVM), is a new addition to the memory/storage hierarchy shown in Figure 2 that fills the performance/capacity gap between DRAM and storage by combining traits of both worlds. Like DRAM, persistent memory comes in the form of Dual In-line Memory Modules (DIMMs) that reside on the memory bus. Therefore, applications can access persistent memory like they do with traditional DRAM, eliminating the need to page blocks of data back and forth between memory and storage. However, unlike DRAM DIMMs, persistent memory DIMMs offer greater capacity and can retain data when the system is shutdown or loses power. Thus, persistent memory can dramatically increase system performance and enable a fundamental change in computing architecture.

Intel Optane DC Persistent Memory Module (PMM) is the first commercially available persistent memory technology. This technology comes in DIMM form factor and embeds capacities up to 512GiB. Intel Cascade Lake processors are the first CPUs to support Intel Optane PMM. Like traditional DRAM, the Optane DIMM sits on the memory bus and connects to the processor's integrated memory controller (iMC). Figure 1 shows a typical system configuration of a hybrid node with DRAM and PMM. A user can have up to one Intel Optane DIMM per channel and up to six on a single socket providing capacities up to 3TiB per socket. Thus, an 8-socket system could access up to 24TB of persistent memory.

To ensure persistence, Intel Optane PMM sits within Intel's asynchronous DRAM refresh (ADR) domain. Intel's ADR domain ensures that CPU stores that reach the ADR domain will survive a power failure. The iMC maintains read and write pending queues (RPQs and WPQs) for each Optane DIMM and the ADR domain includes WPQs. Once the

data reaches the WPQs, the ADR domain ensures that the iMC will flush the updates to persistent memory media on power failure.

The iMC communicates with the Optane DIMM using the DDR-T protocol in cache line access granularity (64B) (Figure 2). The memory access to NVDIMM arrive first at an Apache Pass Controller which coordinates access to the Optane Media. Similar to SSDs, the Optane DIMM perforsms address translation for wear-leveling and bad block management. Thus, it keeps an address indirection table (AIT) for this translation.

The actual access to storage media occurs after address translation. Intel Optane DIMM physical media access granularity is 256 bytes. Thus, the Controller translates smaller requests into largest 256-byte accesses, causing write amplification as small stores become read-modify-write operations. The controller has a small write-combining buffer to merge adjacent writes.

Intel Optane PMem can operate in two modes: memory and App Direct. Memory mode uses Optane PMem as a large capacity main memory without persistence. DRAM is not visible to the users, and instead it serves as a cache for Optane PMem that is transparently managed by the operating system. In App Direct mode, Optane PMem DIMMs appear as independent, non-volatile storage devices. This allows Optane PMem to be used as a byte-addressable persistent memory that is mapped into the system physical address space and directly accessible by applications [].

2.2 Serverless Computing

Serverless computing is a cloud computing execution mode that enables developers to deploy their code without provisioning or managing server infrastructure. The term "serverless" is misleading, as servers are still being used by cloud providers to run the code for developers. However, instead of requesting and managing resources, developers simply provide their code, and the cloud providers handle the servers on behalf of their customers. Cloud providers are responsible for provisioning resources, scaling, fault tolerance, monitoring,

security patches, and so on. Finally, developers simply pay by the execution time and resources used on their code invocations.

Function-as-a-service (FaaS) is the core compute engine for serverless computing. It was first introduced on 2015 by AWS Lambda, and since then, other commercial and open-source offerings have appeared, i.e., Google Cloud Functions, Azure Functions, Apache OpenWhisk, and others. With FaaS, a developer implements the application logic as stateless functions in a high-level language, such as Java, Python, C, C++, and so on. The code is then packaged together with its dependencies and submitted to the serverless platform. Finally, the developer associates an event to each function, i.e., HTTP requests, file uploads, and more. Once a trigger is fired, the cloud provider executes the code associated with that trigger.

2.3 Reinforcement Learning

Reinforcement Learning considers a problem of a learning agent that actively learns from its own experience. Such agent interacts with its environment and periodically receives a reward signal. The agent's goal is to maximize the rewards in the long run. However, the agent is not told which actions to take. Instead, it must discover the actions that yield the highest rewards through trial-and-error.

Figure 1 illustrates a typical reinforcement learning scenario. The agent interacts with the environment in discrete time steps. At each time step t, the agent senses the environment's current state st E S, where S represent the full set of environment states. It then chooses an action at E A(st), where A(st) represents the set of all actions available in the current state. The environment moves to a new state st+1, and the agent receives a reward rt associated with the transition (st,at,st+1).

At any given time, the agent's behavior is defined by a policy. Roughly speaking, a policy is a mapping from perceived states of the environment to actions to be taken when in those states. The agent's purpose is to learn the optimal, or near-optimal, policy that maximizes total reward it receives in the long run.

Exploration vs Exploitation tradeoff One of the challenges that arises in Reinforcement Learning is the choice between exploiting a familiar action known for a reward and exploring unfamiliar actions for unknown rewards, known as the exploration-exploitation tradeoff. The dilemma is that the agent cannot pursue exploration nor exploitation exclusively without failing the task. Instead, it must find a balance between exploration and exploitation. The agent must try a variety of actions to gather enough information and progressively favor for those that appear to be the best.

QLearning Q-Learning is a model-free reinforcement learning algorithm, where the agent learns which is the best action to take given the current state. The agent assess the quality of an action by means of a quality-function (Q-function) Q(s,a), denoting the expected total discounted reward if the agent takes action a on state s and acts optimally thereafter. Given the Q-function, the agent's optimal policy is to choose the action that yields the highest reward.

Figure 4 illustrates the Q-Learning algorithm. The Q-function can be implemented using a simple lookup table. At each step, the agent selects an action a, and observes the reward r and the new state st+1. Then, the agent applies one-step Q-learning, given by: Qlearning formula Where 0¡alpha¡1 is the learning rate and determines to what extent new information overrides the old one. The learned Q-function directly approximates the optimal Q-function, independent of the policy being followed.

Chapter 3: A shim layer for persistent memory

As we have discussed, the release of Intel Optane PMM opens a major opportunity for serverless storage services. This memory technology provides a unique combination of affordable larger capacity, high-performance, and support for data persistence [intel article]. When configured in App-Direct mode, the Optane DIMM and DRAM DIMMs act as independent memory resources under direct load/store control of the applications. This allows the Optane PMM capacity to be used as byte-addressable persistent memory that is mapped into the system application space and directly accessible by applications. Together, these advantages enable Optane PMM to be used as persistent storage with memory-like speeds.

Unfortunately, the resource contention observed within Optane PMM can impose serious performance and contractual implications for a multi-tenant serverless storage service. Given the hallmark autoscaling features of serverless computing, the memory's limited ability to handle accesses from multiple threads can degrade the overall system's performance when workload spikes occur. Furthermore, these storage systems make efficient use of their infrastructure by allowing multiple users, or tenants, to share the physical resources. The performance degradation caused by Optane PMM can lead tenants to experience significant performance variations. The latter inhibits service providers from offering certain service level agreements.

To reduce the contention effect, previous studies recommend limiting the number of threads that access Optane PMM simultaneously. In [fast], Yang et. al they improve the performance of an NVM-aware file system by limiting the number of writer threads that access each Optane DIMM. Similarly, Ribbon [] implements a concurrency control mechanism to reduce the overhead introduced by Cache Line Flushing. While this approach provides a viable solution, it introduces management problems for a system administrator of a multi-tenant serverless storage.

Given the wide heterogeneity of applications running in serverless platforms, implementing efficient concurrency control mechanisms for optimizing an Optane-based serverless storage service is a challenging task. These challenges are discussed in section 3.1, but in short, service providers have three crucial tasks when implementing these control mechanisms. First, they must provide predictable performance, ensuring that all the SLAs from different workloads are met. Second, they must scale resources transparently to meet the current workload demand. Finally, they must come up with policies that allow their system to adapt quickly to sudden workload shifts. To this end, we propose a solution that takes on these responsibilities from the service providers.

In this work, we present a shim layer that addresses the shortcomings of Intel Optane PMM highlighted above, while meeting the different service level agreements from multiple tenants under shifting workloads. Our shim layer, called NVM Middleware, distinguishes between latency-critical and throughput-oriented workloads and applies different concurrency control mechanisms for each one. This enables the system to reduce the contention on the memory device, as well as the interference among workloads with different service level agreements. In addition, we propose the development of a reinforcement learning agent to adapt the NVM Middleware quickly to changing workloads. The agent takes into account the characteristics and service level agreements and learns from past experiences to scale resources accordingly.

3.1 Motivation

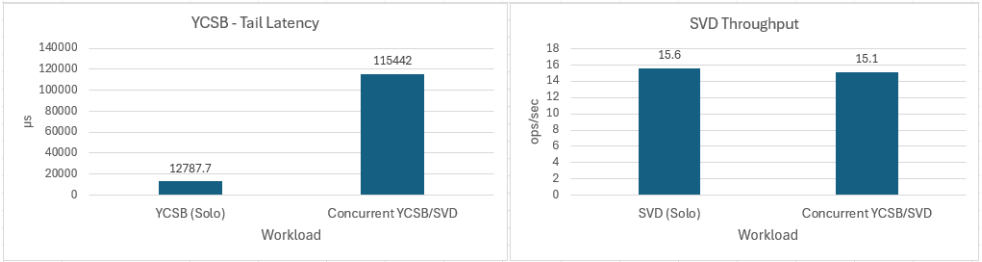

Figure 3.1: Performance Study

3.2 Architecture

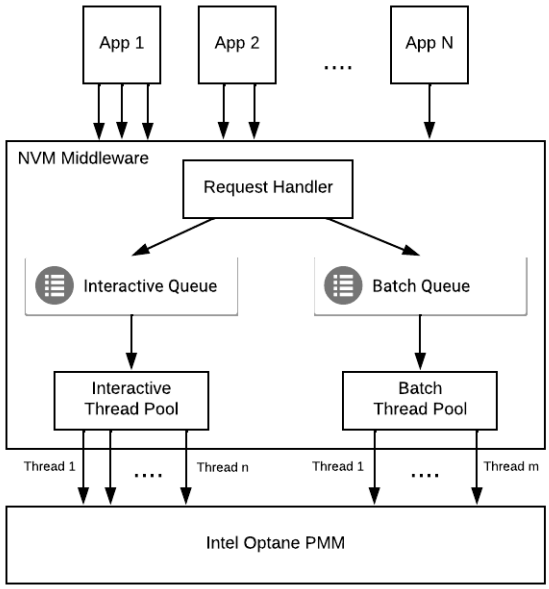

Figure 3.2: NVM Middleware Architecture

3.3 Programming Interface

3.4 Dynamic concurrency control with Reinforcement Learning

Start section here

Table 3.1: Programming Interface

API Name	**Functionality**
start(db, interactiveThreads, batchThreads)	Creates PMEMKV database. Start control threads.
close()	Close PMEMKV database. Stop all threads.
get(key, mode)	Retrieves key from persistent memory.
put(key, value, mode)	Writes key to persistent memory.

Table 3.2: The State Representation

Name	Description
interactiveThreads	Number of interactive threads.
batchThreads	Number of batch threads.
interactiveQueueSize	Interactive Queue Size.
batchQueueSize	Batch queue size.
interactiveBlockSize	Average block size of interactive workload.
batchaBlockSize	Average bloc size of batch workload.
interactiveRWRatio	Read/Write ratio of interactive workload.
batchRWRatio	Read/Write ratio of batch workload.

Table 3.3: The set of possible actions

Action	$interactiveThreads_{t+1}$	$batchThreads_{t+1}$
0	$interactiveThreads_{t+1} = interactiveThreads_t$	$batchThreads_{t+1} = batchThreads_t$
1	$interactiveThreads_{t+1} = interactiveThreads_t + 1$	$batchThreads_{t+1} = batchThreads_t$
2	$interactiveThreads_{t+1} = interactiveThreads_t - 1$	$batchThreads_{t+1} = batchThreads_t$
3	$interactiveThreads_{t+1} = interactiveThreads_t$	$batchThreads_{t+1} = batchThreads_t + 1$
4	$interactiveThreads_{t+1} = interactiveThreads_t$	$batchThreads_{t+1} = batchThreads_t - 1$
5	$interactiveThreads_{t+1} = interactiveThreads_t + 1$	$batchThreads_{t+1} = batchThreads_t + 1$
6	$interactiveThreads_{t+1} = interactiveThreads_t + 1$	$batchThreads_{t+1} = batchThreads_t - 1$
7	$interactiveThreads_{t+1} = interactiveThreads_t - 1$	$batchThreads_{t+1} = batchThreads_t + 1$
8	$interactiveThreads_{t+1} = interactiveThreads_t - 1$	$batchThreads_{t+1} = batchThreads_t - 1$

Bibliography

[1] Eric Jonas, Johann Schleier-Smith, Vikram Sreekanti, Chia-Che Tsai, Anurag Khandelwal, Qifan Pu, Vaishaal Shankar, Joao Carreira, Karl Krauth, Neeraja Yadwadkar, et al. Cloud programming simplified: A berkeley view on serverless computing. *arXiv preprint arXiv:1902.03383*, 2019.

Biography

Include your *biography* here detailing your background, education, and professional experience.