UniHeap: Managing Persistent Objects Across Managed Runtimes for Non-Volatile Memory*

Daixuan Li, Benjamin Reidys, Jinghan Sun, Thomas Shull, Josep Torrellas, Jian Huang
University of Illinois Urbana-Champaign

Abstract

Byte-addressable, non-volatile memory (NVM) is emerging as a promising technology. To facilitate its wide adoption, employing NVM in managed runtimes like JVM has proven to be an effective approach (i.e., managed NVM). However, such an approach is runtime specific, which lacks a generic abstraction across different managed languages. Similar to the well-known filesystem primitives that allow diverse programs to access same files via the block I/O interface, managed NVM deserves the same system-wide property for persistent objects across managed runtimes with low overhead.

In this paper, we present UniHeap, a new NVM framework for managing persistent objects. It proposes a unified persistent object model that supports various managed languages, and manages NVM within a shared heap that enables cross-language persistent object sharing. UniHeap reduces the object persistence overhead by managing the shared heap in a log-structured manner and coalescing object updates during the garbage collection. We implement UniHeap as a generic framework and extend it to different managed runtimes that include HotSpot JVM, cPython, and JavaScript engine SpiderMonkey. We evaluate UniHeap with a variety of applications, such as key-value store and transactional database. Our evaluation shows that UniHeap significantly outperforms state-of-the-art object sharing approaches, while introducing negligible overhead to the managed runtimes.

1 Background and Motivation

Non-volatile memory (NVM), such as phase-change memory (PCM), resistive RAM (ReRAM), NVDIMM, and Intel DC persistent memory, has become a promising technology that offers near-DRAM speed, scalable storage capacity, and data durability. To facilitate its wide adoption in practice, its management and use in software systems have attracted much attention recently.

Specifically, many NVM frameworks and libraries have been developed, such as Mnemosyne, NVHeaps, and Intel PMDK. However, most of them require developers to explicitly specify the persistent data structures in their programs, which significantly increases the development burden. To address this issue, recent researches proposed to integrate NVM into managed runtimes like JVM, in which they leverage the runtime system to transparently manage objects in NVM. As the managed languages, such as Java, Python, and JavaScript, have become the most popular programming languages, such an approach is becoming pervasive. We define this approach as *managed NVM* in this paper.

Although utilizing managed runtimes to use NVM has proven to be an effective approach to simplify the NVM programming, state-of-the-art approaches are runtime specific, and lacking an important system-wide property – *persistent object management across managed runtimes.* It is not easy for a Python program to directly access an object persisted by a Java program, as their runtime-specific object format and layout are different.

As system-wide shared resource, managed NVM deserves data sharing, and it provides non-volatility as shared persistent storage does. Similar to the file systems developed for persistent storage, which manages data in the format of files, and allow different programs to access shared files with block I/O interface, it is highly desirable to enable diverse managed languages to access shared persistent objects efficiently, which could pave the way for developing managed NVM into a generic approach.

As more applications are developed based on managed runtimes today, they usually use different runtime instance for each individual component. And platform operators also prefer to deploy multiple runtime instances on the same machine to best utilize the compute and memory resources. Take the web service for example, its frontend uses JavaSript runtime while its backend adopts JVM.

To achieve persistent object sharing, a straightforward approach is to leverage the persistence layer available in managed runtimes to persist objects to file systems or database. Typical examples include Java Persistence API (JPA) and Java Data Objects (JDO). However, they cause significant performance overhead. Wegiel et al. proposed to exploit the shared memory to enable the cross-language cross-runtime communication, unfortunately, it does not support NVM. The industry has developed Thrift and Protocol Buffers to facilitate the interoperation across multiple languages, however, they suffer from significant marshalling and unmarshalling overheads.

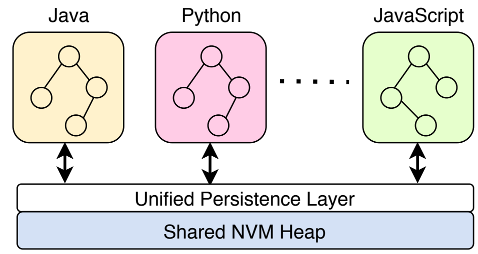

Figure 1: Overview of UniHeap

Table 1: The mapping of language types in UniHeap.

Java	boolean, byte	char	int	long	float	double	reference, array
Python	-	-	int	long	float	-	list, dict, tuple
JavaScript	boolean	-	num	num	num	num	array
UniHeap	char	short	int	long	float	double	reference

2 Design and Implementation

In this paper, we develop a lightweight NVM framework, named UniHeap, for persistent object management across a diversity of managed runtimes(see Figure 1). UniHeap has a unified persistence layer located between the upper-level runtimes and the underlying managed NVM heap.

Unified Persistence Layer. We design a unified persistence layer (UPL) in UniHeap for two purposes. First, UPL has a language-neutral object model, such that it can be extended to support new managed languages. Second, UPL should facilitate object persistence for managed runtimes to achieve low persistency overhead. Unlike recently proposed PCJ for NVM, UniHeap does not introduce new type system. UniHeap provides two built-in types: *numeral* type and *reference* type. It does not provide container types, such as *list, dict,* and *tuple* in Python, as developer can implement their own container type based on these built-in types. As shown in Table 1, the numeral type includes *char, short, int, long, float,* and *double.* For the reference type, the object field stores the pointer to other persistent objects. It is worth noting that *array* is also treated as an object in UniHeap. Thus, its object field can store a pointer to an array. As we provide a transparent type system for managed

*This work has been published at SYSTOR'21 [1].

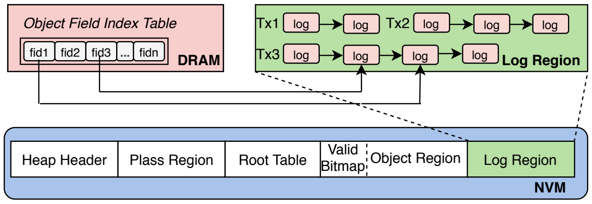

Figure 2: The shared NVM heap structure in UniHeap.

runtimes, different managed language needs to map their type into the type system of UniHeap. We show the mapping of the popular managed languages that include Java, Python, and JavaScript in Table 1. As for Object persistence, we manage to stay compatible with the data persistence approaches in the existing NVM frameworks.

Shared NVM Heaps. UniHeap stores persistent objects in shared NVM heaps. It organizes the shared NVM heap into five regions as shown in Figure 2: heap header, plass region, root table region, object region, and log region. The heap header stores the metadata for the corresponding heap, including the heap name (32 bytes) and heap size (8 bytes). The plass region stores all the class descriptors for UniHeap objects. The root table stores all the durable root objects. Each root is a key-value pair with the format of $<root\_name, root\_addr>$. The object region contains an object valid bitmap, which is used by the GC for reclaiming persistent objects in UniHeap. Its remaining part is organized into numerous fixed-size (16 bytes) chunks, each of which stores an object header. The log region stores the object updates in a log-structured manner.

To facilitate out-of-place update, we decouple the object header from its data, based on the insight that object header is not frequently updated in a transaction. We store the object header in the object region, and object data in the log region. Once a managed runtime allocates an object from the shared NVM heap, UniHeap will bump the pointer in the object region to allocate an object header.

The out-of-place update approach can reduce write traffic to NVM, however, it requires address remapping to retrieve the latest data for read operation. To address this challenge, UniHeap employs a *per-object* mapping table for address translation. Since each object field has a fixed index value during the object lifetime, UniHeap use the field index to store the address of the latest updates in the log region. The address translation procedure is efficient ($O(1)$). In addition, UniHeap caches the object field index table in the fast DRAM for further performance improvement (see Figure 2).

Persistent Object Sharing. UniHeap provides a set of interfaces for managed runtimes to access the shared NVM heap, we classify them into two categories: language-neutral and language-related. The language-neutral interfaces do not need support from managed runtimes. For example, UniHeap implements the *alloc_obj* interface with its own object allocation policy, which is independent from the managed runtime. For those language-related interfaces that include only *init_plass* and *exists_plass*, they require runtimes to have their own implementations. For instance, when UniHeap initializes a class with *init_plass*, UniHeap requires runtime support to fill up the plass structure, since the class metadata is stored in the language source file.

Coordinated GC for Persistent Objects. UniHeap uses the mark-and-compact GC to reclaim persistent objects. As managed runtimes use UniHeap to store their persistent objects, UniHeap will track their references and reachability from runtimes. UniHeap is able to track each runtime's own reference to a persistent object efficiently, can coordinate with multiple runtimes to reach a system-wide safety point, and then stop the world to perform the GC. The GC of UniHeap is crash safe by ensuring each GC phase is idempotent. Similar to previous work, the GC of UniHeap consists of four phases: (1) marking phase, (2) relocation phase, (3) compaction phase, and (4) clean-up phase. The marking phase is naturally idempotent, since this phase does not change the heap states. As for the compaction phase, we maintain the old object region until the clean-up phase. Therefore, upon a crash or failure, UniHeap can redo the GC during the system recovery.

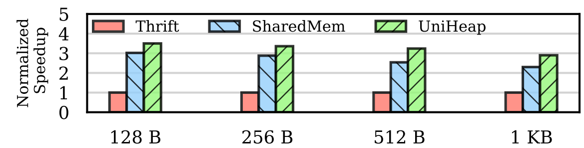

Figure 3: Performance comparison of different persistent object sharing approaches.

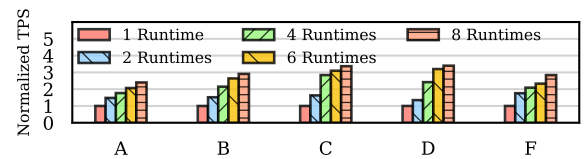

Figure 4: Scalability of UniHeap.

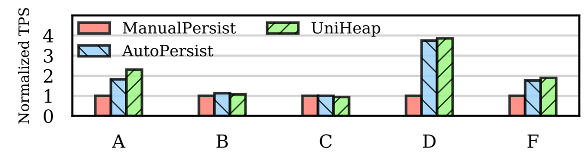

Figure 5: Performance of running HotSpot JVM with UniHeap, when using QuickCached+YCSB.

Overall, we make the following contributions in this paper.

- We propose a generic NVM framework that provides a unified object model to enable efficient persistent object sharing cross diverse managed runtimes within NVM.
- We present a shared NVM heap for managing persistent objects. It manages objects in a log-structured manner and supports both in-place and out-of-place updates to reduce data persistence overhead, while ensuring the crash-safety.
- We develop an efficient GC scheme by decoupling the metadata and data of persistent objects in the NVM heap, and coordinate GC operations with managed runtimes to ensure the correctness of object cleanups.
- We enable UniHeap to support three popular managed runtimes, including HotSpot JVM, cPython, and JavaScript.

3 Evaluation

We implement UniHeap system prototype as shared library with 9,163 lines of C programming code. As for the shared NVM heap, UniHeap uses the memory-mapped interface with Direct Access (DAX) enabled for fast access to the NVM device. To evaluate the efficiency of UniHeap, we run a variety of data-intensive applications and typical benchmarks for different managed runtimes, including Yahoo Cloud Service Benchmarks (YCSB) for Java, Python Performance Benchmark Suite for Python, and JetStream2 for JavaScript. Our evaluation(see Figure 3,Figure 4,and Figure 5) shows that (1) UniHeap performs better than state-of-art approaches of persistent object sharing; (2) It can scale the persistent object sharing as we increase the number of managed runtimes; (3) UniHeap enables persistent object sharing across different runtimes without introducing much performance overhead. We presented the detailed evaluation in [1].

References

[1] Daixuan Li, Benjamin Reidys, Jinghan Sun, Thomas Shull, Josep Torrellas, and Jian Huang. Uniheap: Managing persistent objects across managed runtimes for non-volatile memory. In *Proceedings of the 14th ACM International Conference on Systems and Storage*, SYSTOR '21, New York, NY, USA, 2021. Association for Computing Machinery.