A NUMA-aware NVM File System Design for Manycore Server Applications

June-Hyung Kim, Youngjae Kim, Safdar Jamil, and Sungyong Park
Department of Computer Science and Engineering
Sogang University, Seoul, South Korea
{junehyung, youkim, safdar, parksy}@sogang.ac.kr

***Abstract*—NOVA, a state-of-the-art NVM-based file system, is known to have scalability bottlenecks when multiple I/O threads read/write data simultaneously. Recent studies have identified the cause as the coarse-grained lock adopted by NOVA to provide consistency, and proposed fine-grained range-based locks to improve the scalability of NOVA. However, these variants of NOVA only scale on Uniform Memory Access (UMA) architecture and do not scale on Non-Uniform Memory Access (NUMA) architecture. This is because NOVA has no NUMA-aware memory allocation policy and still uses non-scalable file data structures. In this paper, we propose a NUMA-aware NOVA file system which virtualizes the NVM devices located across NUMA nodes so that they can be used as a single address space. The proposed file system adopts a local-first placement policy where file data and metadata are placed preferentially on the local NVM device to reduce the remote access problem. In addition, the lock-free per-core data structures proposed in this file system allow data to be updated concurrently while mitigating the remote memory access. Extensive evaluations show that our NUMA-aware NOVA for parallel writing is scalable with respect to the increased core count and outperforms vanilla NOVA by 2.56-19.18 times.**

I. Introduction

In recent few years, several file systems have been proposed for Non-Volatile Memory (NVM) devices, such as 3D-Xpoint [1]–[4]. Among them, NOVA [2], a state-of-the-art NVM file system, ensures higher throughput and lower read-write latency than block-based file systems. NOVA also ensures consistency of file data and metadata through its log-structured design. NOVA adopts per-inode logging which logs metadata for every write operation. However, NOVA does not provide any degree of scalability in terms of I/O throughput when concurrent shared file I/Os are performed [5], [6]. This is mainly due to the coarse-grained locks on inodes to guarantee consistency of its per-inode logs, which negates the benefits of concurrent nature of NOVA and high-performance NVM devices.

To solve the scalability issue due to low concurrency caused by coarse-grained inode lock in NOVA, it has been suggested to use a fine-grained range based Readers-Writer (RW) lock, referred as *range lock* instead of the coarse-grained mutex locks for inodes [5], [7], [8]. The range lock-based NOVA succeeds in scaling performance for shared file I/Os. However, this fine-grained lock solution applies only to the Uniform Memory Access (UMA) architecture and does not scale on Non-Uniform Memory Access (NUMA) architecture. This is

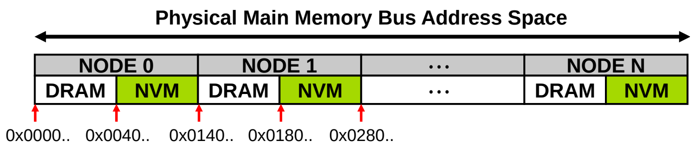


Fig. 1: Physical address space configuration method of a NUMA server where NVM is installed on each node.

because NOVA is not designed for NUMA environment. For example, NOVA places all file data and metadata on a single NUMA node [9]. Therefore, threads running on other nodes must access the file via remote memory access, which leads to huge performance loss.

In this paper, we propose a NUMA-aware NOVA to show the scalability on the NUMA-based manycore servers. Our work has the following contributions:

- **Virtualizing NVM Devices:** In order to store files across multiple NUMA nodes, the non-contiguous physical address space of NVMs located at multiple nodes is virtualized into one logical address space as shown in Figure 1. We also propose a *local first* write policy to place file data and metadata preferentially on the NVM device allocated to the CPU where the thread is executing.
- **Lock-Free Per-Core Data Structures:** We suggest lock-free per-core data structures such as per-inode log using *Global Log* and *Local Log* to ensure the scalability by allowing multiple threads to perform write operations concurrently.
- **Evaluation of Intel Optane DC Manycore Servers:** We implemented our proposed ideas in Linux environment. Our extensive evaluations with an Intel Optane DC manycore server show that our proposed NUMA-aware NOVA for parallel writing is scalable with respect to the increased number of cores and outperforms vanilla NOVA by 2.56-19.18 times. In terms of parallel read, the NUMA-aware NOVA is scalable up to 56 cores and outperforms vanilla NOVA by 2.54 times.

II. Background and Motivation

A. NOVA File System

NOVA is a state-of-the-art NVM file system. It adopts log-structure design to guarantee consistency and records the

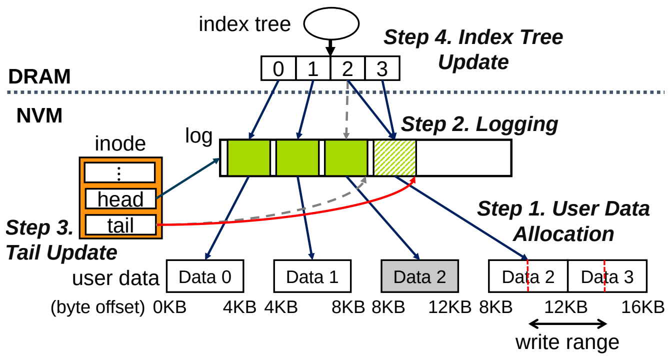


Fig. 2: Write I/O flow of NOVA file system.

modifications in the per-inode log at NVM in a Copy-On-Write (COW) manner. The core data structures used in NOVA are shown in Figure 2. The index tree in DRAM is maintained for each file and utilized to retrieve data pages corresponding to the file offset of the read/write operation. The Inode in NOVA is of 128 B stored in NVM and includes head and tail pointers for the start and end of the log entries stored in the inode log. The inode log is a series of 4 KB log pages. Each log entry is 64 B while the user data is stored in 4 KB data pages and is allocated in NVM by per-CPU memory page allocator.

The overall write I/O flow of NOVA is shown in Figure 2 where a user requests to write data in the range of 10 KB-14 KB of a file. At step 1, NOVA will allocates two new data pages as the unit of 4 KB write operation and the content of previous data page (Data 2) is copied to the newly allocated pages, and user data is overwritten in the range to be updated. At step 2, the corresponding log entry is appended to the inode log using the tail pointer from inode as it points to the last committed log entry. Thus, the new log entry is written right after the tail pointer. If the tail pointer is at the last log entry, a new log page is allocated and the log entry is written to the new log page. At step 3, the tail pointer is updated to reflect the position of the new log entry. Finally at step 4, the index tree in DRAM is updated such that the new index node points to the new log entry and the NOVA's atomic write operation is finished.

B. Scalability Limitation with Range-based RW Lock

NOVA uses per-inode locks to avoid inconsistent cases between multiple threads when they write data on the same file at different offsets or they update the log at the same time. For instance, suppose that thread $T1$ and $T2$ write on the same file at the same time and there is no per-inode lock. If $T1$ is writing a log entry (step 2 of Figure 2) but $T2$ appends a log entry before $T1$ updates the tail pointer (step 3 of Figure 2), the log entry written by $T1$ can be overwritten by $T2$'s log entry. As a result, this will lead to the lose of $T1$'s data. In our recent work [5], we identified that the NOVA's per-inode lock becomes the bottleneck for parallel I/O as it serializes threads that try to read or write the same file. Then we proposed a fine-grained range-based RW lock. The proposed range lock first

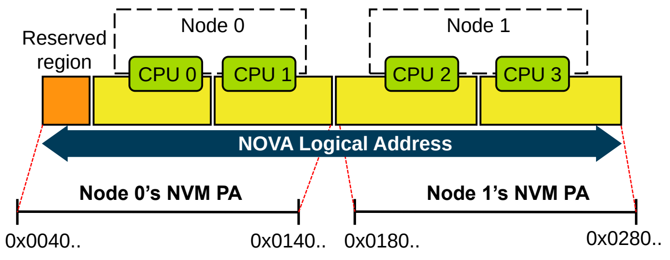


Fig. 3: An example of the NOVA with NUMA-aware memory management.

checks the range of the write request; if write ranges do not overlap, the lock can be acquired and the user data (step 1 of Figure 2) can be written concurrently. While the fine-grained range-based RW lock ensures file system scalability in the UMA architecture, it does not help in scaling performance in the NUMA architecture.

III. NUMA-aware per-CPU Log Structure Design

A. NOVA file system on Manycore servers

Currently, manycore servers are mostly based on NUMA architecture, and the system applications being designed for manycore machines are required to be aware of their internal architectures to fully exploit their performance. However, NOVA is an NVM based file system designed to reside on a single NVM space. Figure 1 illustrates the non-contiguous address space configuration of multiple NVM devices installed on a NUMA-based manycore machine. Since only NVM space on a single NUMA node can be used to place data and log pages in current NOVA file system, threads executing on other NUMA nodes will cause remote memory access. This becomes the scalability bottleneck. In what follows, we explain how we mitigate these challenges and how we design the NUMA-aware file system.

B. A Unification of NVM Devices

The NUMA-aware file system can reduce remote memory accesses by distributing files over multiple NUMA nodes. In order to design the NUMA-aware NOVA, we have virtualized the non-contiguous NVM address space from multiple NUMA nodes to a single contiguous virtual address space. For this, we import the NVM devices' information from all NUMA nodes to the superblock when mounting the file system. As a result, the physical addresses of NVM devices are linearly mapped, which allows NUMA-aware NOVA to easily place user data and metadata in multiple NVM devices.

To mitigate the remote memory accesses, a memory allocation policy is required. In our proposed NUMA-aware NOVA, we introduced a *local first write* policy, where threads give preference to write files to a local NVM device. Figure 3 shows the proposed NUMA-aware memory allocation policy with two NUMA nodes. In this case, threads running on CPU-0 and CPU-1 prefer to write files in physical address space ranging from *0x0040* to *0x0140*, while CPU-2 and CPU-3 write over the range from *0x0180* to *0x0280*. Also, NUMA-aware NOVA

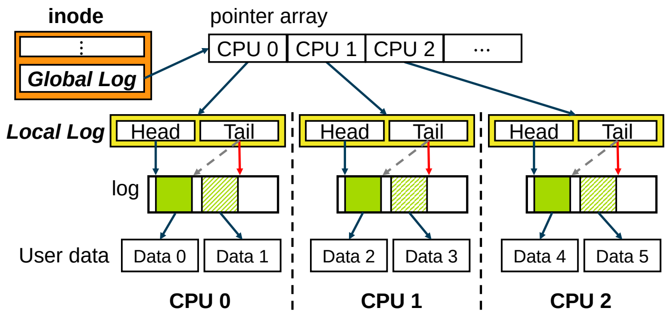


Fig. 4: NOVA with per-CPU log structure.

divides the entire logical address space based on the size of NVM devices and creates per NUMA node partitions. Further, the partitions are divided into the number of cores in each node and are managed by the per-CPU memory allocator in NOVA. Data and log pages for each thread are allocated from the region allocated to the core on which the thread is running. This allows each I/O thread to allocate data or log pages from the local NVM space first.

C. Lock-Free Per-CPU log structure

File data structures shared among various threads become a major cause of scalability bottleneck for the file system in a NUMA-based system. If files are shared among multiple threads running on different NUMA nodes, it leads to huge performance loss due to frequent remote memory access. Furthermore, since the shared resources are protected by locks to ensure consistency, concurrent update operations are serialized. Therefore, we propose per-CPU data structures for scalable NOVA and extend the NOVA's per-inode log structure to be a lock-free per-CPU log.

Figure 4 shows the proposed lock-free per-CPU log structure for NOVA, where we introduce two new persistent data structures: *Global Log* and *Local Log*. The *Global Log* is a pointer array for each inode that indexes each CPU and corresponding *Local Log* pointer as shown in Figure 4. The *Local Log* is a per-CPU private data structure with head and tail pointers for per-CPU logs. These two data structures allow multiple threads from different NUMA nodes to perform file operations concurrently and improve the scalability.

The new write flow in the NUMA-aware NOVA can be as follows. Suppose that three threads want to write data to a shared file on CPU cores 0, 1, and 2. Then, each thread allocates a data page or a log page using its core's memory allocator based on the *local first* allocation policy from the local NVM device. As a result, threads can update their data and log pages concurrently through local memory access. Additionally, they can append a log entry to the log page after each thread updates the tail pointer of *Local Log* with reference to *Global Log*. Finally, the index tree of DRAM is updated to point to the new log entries similar to the existing NOVA, and the parallel write operation of NUMA-aware NOVA completes.

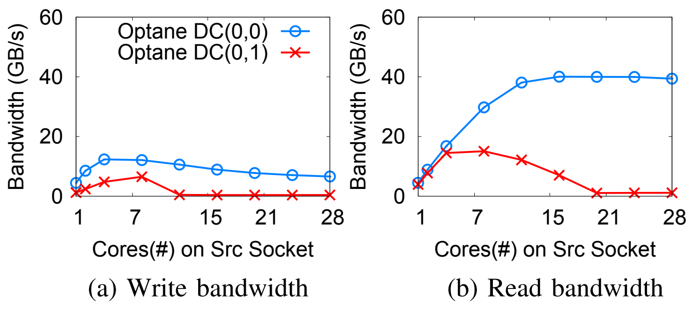


Fig. 5: The result of bandwidth measurement as the number of cores increases in Testbed. Threads are bound to CPU, having one-to-one mapping between them.

IV. Experimental Results

A. Intel Optane DC PM Server

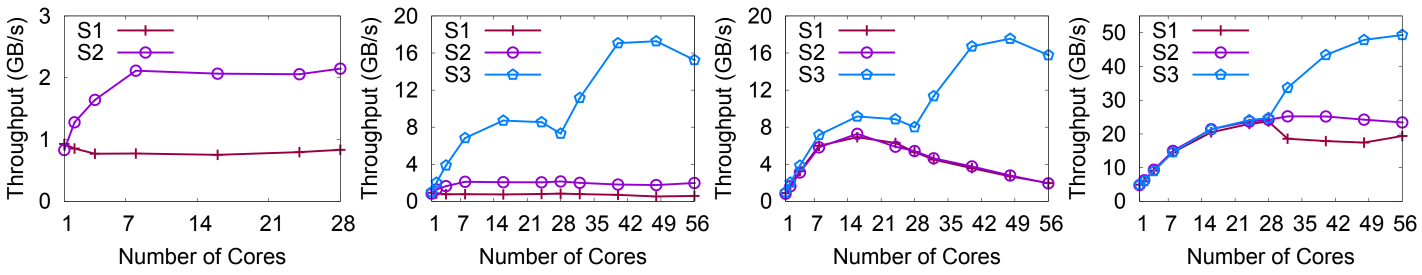
To evaluate the scalability of NOVA, we conducted experiments on a Manycore server equipped with Intel Optane DC Persistent Memory (PM) modules. The detailed specifications of the Testbed are shown in Table I. Each socket is equipped with 6 Intel Optane DC PM modules. The hardware setup used in the experiments is as follows. The memory space of the PM modules within a single socket is provided by the operating system as a contiguous physical address space. In this setup, the access to the PM is interleaved across 6 PM modules in a single socket. However, the server does not allow to aggregate the PM modules across sockets.

To understand the performance of Intel Optane DC PM modules, we measured the bandwidth of each memory module of our Testbed by increasing the number of cores with read-only and write-only workloads. For the experiments, we used the Memory Latency Checker (MLC) tool by Intel [10]. The MLC tool measures the peak bandwidth for workloads, in which multiple threads sequentially read or write data from or to memory devices. In particular, for accurate memory latency measurements, the MLC tool disables the hardware pre-fetcher.

Figure 5(a) shows the measured write bandwidth of PM. In the figure, Optane DC(src, dst) indicates the cores in the socket number src performing I/O operations when they access the memory buffer allocated to Optane DC PM in socket number dst. Optane DC(0,0) reaches the peak bandwidth of 12.3 GB/s on 4 cores but it gradually degrades with the increasing number of cores. On the other hand, the maximum bandwidth of Optane DC(0,1) reaches at the peak of 6.5 GB/s on 8 cores, but it also continues to decrease with the increasing

TABLE I: Intel Optane DC PM Server.

CPU	Intel(R) Xeon(R) Platinum 8280M v2 2.70GHz CPU Nodes (#): 2, Cores per Node (#): 28
Memory	DRAMs per Node (#): 6, DDR4, 64 GB * 12 (=768GB)
PM	Intel Optane DC Persistent Memory PMs per Node (#): 6, 128 GB * 12 (=1.5TB)
OS	Linux kernel 4.13.0



(a) N-to-1 Write in UMA (DWOM) (b) N-to-1 Write (DWOM) (c) N-to-N Write (DWOL) (d) N-to-N Read (DRBL)

Fig. 6: Evaluation of the scalability of the NOVA file system according to the parallel I/O pattern on the Intel Optane DC server. Threads are bound to CPU, having one-to-one mapping between them.

number of cores. It is assumed that the write bandwidth is reduced by the hardware limitation of the Optane DC PM server as described in [11]. Figure 5(b) shows the measured read bandwidth of PM. Optane DC(0,0) bandwidth increases linearly as the number of cores increases, resulting in a maximum of 40.04 GB/s on 28 cores. The bandwidth of Optane DC(0,1) increases as the number of cores increases, resulting in a maximum of 15 GB/s on 8 cores. However, the bandwidth drops after that due to the same hardware limitations.

In short, we have the following observations from Figure 5. First, the read bandwidth of PM is higher than the write bandwidth. The read bandwidths of Optane DC(0,0) and Optane DC(0,1) are 3.24x and 2.31x higher than those of the corresponding write bandwidths, respectively. Second, the degree of the NUMA effect in PM is high. In terms of write, Optane DC(0,0) shows 1.86x higher write bandwidth on 8 cores than Optane DC(0,1), and 16.5x higher write bandwidth on 28 cores. In terms of reading, the read bandwidth of Optane DC(0,0) on 8 core is 1.98x higher than that of Optane DC(0,1) and 35.7x higher on the 28 core.

B. *Evaluation of NUMA-aware NOVA*

For the evaluation, we implemented three versions of NOVA:

- *NOVA(V) (S1)*: Vanilla NOVA.
- *NOVA(RL) (S2)*: NOVA using a range lock.
- *NOVA(RL+NUMA) (S3)*: NOVA using a range lock with NUMA-aware design.

The FxMark benchmark [6] that can generate various workload patterns is used for the evaluations. Specifically, we used three parallel I/O workloads, DWOM, DWOL, and DRBL. The DWOM workload performs multi-threaded shared file writes where multiple threads write to a shared file (N-to-1 write). The DWOL workload performs multi-threaded private file writes where multiple threads write to their private files (N-to-N write). The DRBL workload performs multi-threaded private file reads where multiple threads read the files after writing the files (N-to-N read).

Figure 6(a) verifies the scalability of NOVA(RL) using the DWOM workload in the UMA architecture, while NOVA(V) does not provide any scalability due to the per-inode lock. NOVA(V) shows the peak bandwidth of 0.8 GB/s on 1 core, and it decreases slightly as the number of cores increases. On the other hand, NOVA(RL) shows its maximum throughput of slightly higher than 2 GB/s on 8 cores and maintains the throughput up to 28 cores. This is because the coarse-grained lock is replaced by the fine-grained range lock and thus shared files can be updated in parallel. However, the throughput does not scale after 8 cores for NOVA(RL) as the file log cannot be updated concurrently.

Figure 6(b) shows the results using the DWOM workload in the NUMA architecture. NOVA(V) and NOVA(RL) show the same results as those in the UMA environment. Since these are not NUMA-aware, there is no performance improvement in the NUMA-based Manycore servers. On the other hand, NOVA(RL+NUMA) shows the scalable performance up to 48 cores due to the per-CPU log and *local first write* policy. The throughput of NOVA(RL+NUMA) increases up to 17.2 GB/s on 48 cores, which exceeds the Testbed's NUMA boundary of 28 cores. In addition, NOVA(RL+NUMA) shows high parallelism than NOVA(RL) despite NUMA-effect due to the lock-free data structure. For instance, the throughput of NOVA(RL+NUMA) on 16 cores is 31% higher than that of NOVA(RL).

Figure 6(c) presents the results using the DWOL workload in the NUMA architecture. As a range lock is basically designed to improve the DWOM performance, the performance of NOVA(RL) using the DWOL workload is not much different from that of NOVA(V). It scales up to 16 cores, but after that, it slightly decreases as the number of cores increases. We suspect that the hardware bottleneck of the Optane DC PM server shown in Figure 5 causes this problem. Besides, the performance after 28 cores continues to decrease due to the file arrangement without NUMA-awareness. However, NOVA(RL+NUMA) scales up to 48 cores with the maximum throughput of 17.69 GB/s similar to the experiments using the DWOM workload.

Figure 6(d) compares the results using the DRBL workload in the NUMA architecture. In terms of parallel read, all versions of the NOVA scale up to 28 cores. However, NOVA(V) and NOVA(RL) do not scale after 28 cores. In the case of NOVA(V), the per-inode lock degrades the performance due to the single reader counter problem [12]. On the other hand, NOVA(RL) does not perform well due to the NUMA-effect because all files are placed only at node 0. Finally, NOVA(RL+NUMA) places each thread's file in the local NVM with *local write first* policy. This allows threads to read files with local memory access. As a result, NOVA(RL+NUMA) scales up to 56 cores and shows the peak bandwidth of 49.32 GB/s on 56 cores.

V. Conclusion

In this paper, we developed a NUMA-aware NOVA file system to support file system scalability on Manycore machines. Specifically, we first virtualized the NVM modules of several NUMA nodes and reduced the number of remote accesses that can occur when performing parallel I/O through the local write first policy. Second, we extended the NOVA's per-inode log to the lock-free per-CPU log. Extensive evaluations have shown that NUMA-aware NOVA is scalable for parallel writing as the number of cores increases on Intel Optane DC Manycore machines, and shows 19.18 times higher write throughput than vanilla NOVA.

Acknowledgments

This work was supported by Institute of Information & Communications Technology Planning Evaluation (IITP) grant funded by the Korea government(MSIT) (No. 2014-0-00035, Research on High Performance and Scalable Manycore Operating System). Sungyong Park is the corresponding author.

References

[1] Intel, "Revolutionizing Memory and Storage," https://www.intel.com/content/www/us/en/architecture-and-technology/intel-optane-technology.html, 2017.

[2] J. Xu and S. Swanson, "NOVA: A Log-structured File System for Hybrid Volatile/Non-volatile Main Memories," in *Proceedings of the 14th USENIX Conference on File and Storage Technologies (FAST 16)*, 2016, pp. 323–338.

[3] H. Volos, S. Nalli, S. Panneerselvam, V. Varadarajan, P. Saxena, and M. M. Swift, "Aerie: Flexible File-system Interfaces to Storage-class Memory," in *Proceedings of the 9th European Conference on Computer Systems (EuroSys 14)*, 2014, pp. 14:1–14:14.

[4] R. Kadekodi, S. K. Lee, S. Kashyap, T. Kim, A. Kolli, and V. Chidambaram, "Splitfs: reducing software overhead in file systems for persistent memory," in *Proceedings of the 27th ACM Symposium on Operating Systems Principles (SOSP)*, 2019, pp. 494–508.

[5] J.-H. Kim, J. Kim, H. Kang, C.-G. Lee, S. Park, and Y. Kim, "pNOVA: Optimizing Shared File I/O Operations of NVM File System on Manycore Servers," in *Proceedings of the 10th ACM SIGOPS Asia-Pacific Workshop on Systems (APSYS 19)*, 2019, pp. 1–7.

[6] C. Min, S. Kashyap, S. Maass, and T. Kim, "Understanding Manycore Scalability of File Systems," in *Proceedings of the 2016 USENIX Conference on Usenix Annual Technical Conference (ATC 16)*, 2016, pp. 71–85.

[7] A. Kogan, D. Dice, and S. Issa, "Scalable range locks for scalable address spaces and beyond," in *Proceedings of the Fifteenth European Conference on Computer Systems (EuroSys 20)*, 2020, pp. 1–15.

[8] J. Corbet. (2017, June) Range reader/writer locks for the kernel. [Online]. Available: https://lwn.net/Articles/724502/

[9] J. Xu, J. Kim, A. Memaripour, and S. Swanson, "Finding and Fixing Performance Pathologies in Persistent Memory Software Stacks," in *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS 19)*, 2019, pp. 427–439.

[10] Intel, "Intel Memory Latency Checker v3.8," https://software.intel.com/en-us/articles/intelr-memory-latency-checker.

[11] J. Yang, J. Kim, M. Hoseinzadeh, J. Izraelevitz, and S. Swanson, "An empirical guide to the behavior and use of scalable persistent memory," in *Proceedings of the 18th USENIX Conference on File and Storage Technologies (FAST 20)*, 2020, pp. 169–182.

[12] I. Calciu, D. Dice, Y. Lev, V. Luchangco, V. J. Marathe, and N. Shavit, "NUMA-aware Reader-Writer Locks," in *Proceedings of the 18th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming (PPoPP 13)*, 2013, pp. 157–166.