Scheduling HPC Workflows with Intel Optane Persistent Memory

Ranjan Sarpangala Venkatesh
Georgia Institute of Technology
ranjansv@gatech.edu

Tony Mason
University of British Columbia
Georgia Institute of Technology
fsgeek@{gatech.edu,cs.ubc.ca}

Pradeep Fernando
Georgia Institute of Technology
pradeepfn@gatech.edu

Greg Eisenhauer
Georgia Institute of Technology
eisen@cc.gatech.edu

Ada Gavrilovska
Georgia Institute of Technology
ada@cc.gatech.edu

***Abstract*—HPC workloads and their increasing data processing demands have led to using *in situ* execution, which couples simulation and analytics to reduce cross node memory accesses and their negative impact on overall performance. *In situ* executions can benefit from new classes of persistent memory technologies, such as Intel® Optane™ DC Persistent Memory (PMEM), which provide a denser, lower cost, and lower performance memory option for server class machines. However, PMEM creates a new set of trade-offs that must be considered to further improve performance for these HPC workloads and to realize the expected benefits. Prior work has only focused on describing how to tune for a *single* workload component, which may not yield optimal results for the entire workload.**

In this paper, we use a suite of workflows with different characteristics to understand the impact of using PMEM for *in situ* workflow executions with respect to different decisions on how PMEM is shared. Based on our experimental observations, we make recommendations for the considerations that must be incorporated for future workflow schedulers to maximize the benefits of the PMEM resource.

I. Introduction

High Performance Computing (HPC) workloads are increasingly required to process larger data volumes, which lead to excessive data movement costs across memory and storage tiers. One common solution to this is to reduce cross-node traffic by using coupled simulation and analytics applications. These *in situ* executions benefit from using local persistent memory, which provides large memory capacity at access latencies similar do DRAM. This provides memory-speed I/O performance, while leaving the available DRAM capacity to be utilized for core computations [1]. The first commercially available persistent memory, Intel® Optane™ DC Persistent Memory (Intel Optane), has been rapidly adopted by the HPC community. Intel Optane is already being used by several supercomputer facilities including Aurora, Frontera, and Barcelona Supercomputing Center. Intel Optane provides large capacity (up to 6TB per node) and is persistent and byte addressable.

This raises the question, *How to maximize the benefit that in situ workflows can obtain from using PMEM for their data exchanges?* Recent work has developed detailed performance analysis of individual applications on PMEM platforms [2], [3]. These efforts demonstrate that due to certain characteristics of the PMEM technology parameters, such as read-write access and bandwidth asymmetry, the characteristic of the device-internal cache, access stride and width, etc., applications exhibit significant variability of the observed device bandwidth and latency. The extent of this variability is determined by both the configuration of the software stack and by a range of application-level parameters, such as access locality, granularity, stride, presence of concurrent operations, read-write mix, etc., and can range up to 30× [2]. In response, the past work has also developed recommendations regarding the configuration of application parameters and the PMEM software stacks, so as to maximize the potential performance gains.

Unlike a single application, *in situ* workflows iterate over I/O data as it is being streamed across the simulation and analytics components forming the workflow execution pipeline [1], [4]. Optimizing just one of these components has implications on the performance experienced by the other component. Figure 1 illustrates this by showing normalized runtime of two workflows based on miniAMR. The workflows differ with respect to the analytics component – an I/O-intensive read only component vs. a compute-intensive analytics kernel based on matrix multiplication. We run the two workflows in two different configurations, the parameters of which are not germane for the current illustration (but are presented later in section II). Although the simulation app is the same in both the workflows, optimizing purely for this component is insufficient as a change in the analytics kernel can result in a 1.4-1.6× loss in performance, unless some other parameters of how the workflow or its use of the PMEM resources are changed. In fact, as we show in this paper, the effects of a poorly configured use of PMEM for streaming I/O across a set of workflow components, can obviate any benefits expected by use of PMEM. The exact effect of a shared use of PMEM in workflows is application-specific and sensitive to properties such as the composition of the iteration cycle of individual components (i.e., their compute vs. I/O time),

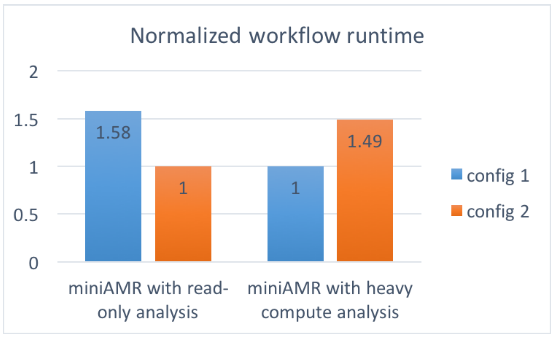


Fig. 1: Performance of miniAMR workflows with different configurations

writer and reader mix and concurrency, the granularity of I/O operations, etc. *Optimizing individual workflow components does not guarantee a best choice in end-to-end workflow performance and there is no single configuration which works for all workflows.* Although prior work has explored the impact of the configuration parameters of the PMEM stack on individual applications, the interplay of these in the context of HPC workflows has not been studied.

In this paper, we explore the effect of the decisions regarding how workflow components are scheduled on a server platform and how they use the available Intel Optane resources for streaming I/O on end-to-end workflow performance. We do this by considering HPC workflows with different characteristics, so as to illustrate the importance of the problem across workloads and to develop recommendation strategies for workflow scheduling and resource management engines. We demonstrated up to 69% performance improvement, measured by end-to-end workflow execution runtime, using synthetic benchmarks and application kernels based on real HPC applications, GTC and miniAMR, on a testbed based on the first-generation Intel Optane memory.

II. Background

We briefly summarize the decision space of *in situ* workflow scheduling, and present the characteristics of PMEM that are most relevant to the question of shared use of its resources for streaming I/O in workflows.

A. *In situ Workflow Deployment Considerations*

In situ workflow scheduling is based on deploying the components of a single workflow to shared server nodes. This allows acceleration of data movements across workflow components, which has been shown possible and important by workflow systems in operation today [5]. *In situ* scheduling presents challenges as well due to the multi-tenancy aspect of the server platform. Simulation and analytics components share server resources – cores, memory hierarchy, I/O devices and their resource allocation and execution must be managed to avoid contention points from dominating the end-to-end execution. Failing to do so eliminates potential benefits from *in situ* execution.

Prior work has explored these challenges in the context of shared use of compute cores and caches, accelerators, and network [6]–[9]. In this work, we focused on challenges related to the shared use of the PMEM resources. For this reason, we focus on workflow deployments which change how the PMEM resource is used and impact the contention points that get exhibited in the path of PMEM access. We assume that PMEM is not used as compute memory during the computational phases of workflow components, since that has been shown to degrade execution time [10], but focus on its use for streaming I/O, as shown important in our prior work [1].

For concreteness, we consider a simplified server platform comprising two compute sockets, each with locally attached DRAM and PMEM resources, as illustrated in Figure 2. We do not focus on deployment scenarios where workflow components share individual cores or even sockets, as in such cases, the dominant factors impacting workflow performance are related to the benefits of shared cache accesses, which may accelerate data movement under certain conditions [11]. Instead we focus on configurations that are directly impacted by the shared use of the PMEM resource and the relative locality of workflow components to the I/O data channels placed in PMEM, or the contention arising from concurrent accesses to the shared PMEM controllers.

We also do not consider scenarios where simulation and analytics can be scheduled in alternation on a single socket. HPC applications are long-running and have large memory footprints, hence analytics needs to be scheduled to run concurrently (or in interleaved manner) with the simulation, so as to operate on its output in an online manner, and needs to be placed on distinct set of resources that do not result in cache and memory pollution.

Locality of PMEM accesses. For the sample platform illustrated in Figure 2, the simulation (writer) and analytics (reader) components of a workflow are placed on two different sockets. The streaming I/O channel is allocated in one of the PMEM modules, either local to the simulation, or local to the analytics. The locality choice impacts the I/O performance that each of the workflow components will experience, as remote PMEM accesses are shown to degrade the effective PMEM latency *and* bandwidth. As such, the question that arises is the **placement** decision that the system scheduler needs to make – which workflow component should be prioritized with respect to its *PMEM locality* so as to maximize the end-to-end performance benefits.

Contention for PMEM resources. *In situ* workflow components can benefit from being executed at the same time, in parallel, in the absence of contention on a single shared resource. This is because contention for shared resources such as accelerators (e.g., GPU [12]) or interconnects ([13]) limit parallelism. The extent to which the shared use of PMEM leads to significant contention, is further depended on how the workflows are **scheduled**: in *parallel*, where simulation and analytics are scheduled to execute during the same time and their I/O operations and access PMEM overlap in time, or *serially*, where the two workflow components are scheduled

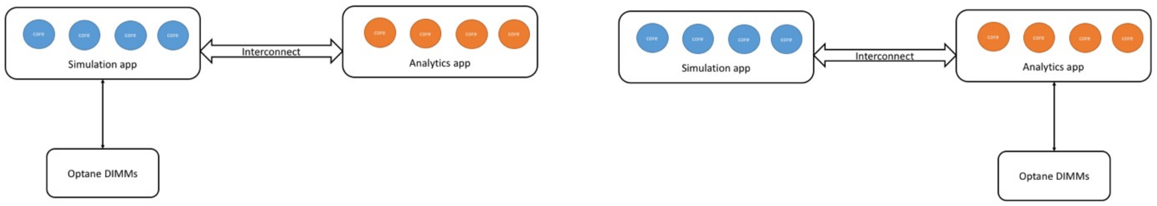


Fig. 2: Considered workflow deployment alternatives. The PMEM component, shown local either to simulation or analytics, is used for streaming I/O data buffers. Additional scheduling decisions concern whether simulation and analytics are permitted to access PMEM in parallel, or their accesses are scheduled to occur in distinct intervals, but scheduling their I/O phases in a serial manner.

TABLE I: Summary of configurations

Config label	Execution Mode	Placement
$S-LocW$	Serial	local-write-remote-read
$S-LocR$	Serial	remote-write-local-read
$P-LocW$	Parallel	local-write-remote-read
$P-LocR$	Parallel	remote-write-local-read

consecutively, and their I/O operations that access PMEM do not overlap in time.

Based on this P arallel vs. S erial scheduling dimension, and whether the placement decisions illustrated in Figure 2 prioritize Locality to the streaming I/O buffers for simulation (W) or analytics (R) , we consider four scheduler configurations ($S-LocW|R$ and $P-LocW|R$) as summarized in Table I.

B. Optane PMEM

Prior work describes the access performance characteristics of Intel Optane PMEM [2], [3], [14]. We summarize their findings here and show how they relate to the choice of workflow scheduling configurations.

Access granularity. Intel Optane PMEM consists of up to 6 persistent memory modules per node. For maximum bandwidth, modules are configured in *interleaved mode*, where data is striped across all modules in the PMEM device. Optane uses 4KB contiguous chunks to form 24KB stripes spread across the 6 interleaved modules, similar to RAID-0. Prior work [2] has shown non-uniform distribution of random 4KB accesses by 6 or more threads can reduce performance. With 4KB accesses, multiple threads eventually end up contending for the same Optane DIMM module. This reduces performance due to its limited bandwidth.

Effective bandwidth. In interleaved mode Optane has a maximum local read bandwidth of 39.4GB/s, and maximum local write bandwidth of 13.9 GB/s. Further, the read bandwidth scales up to 17 concurrent operations, while write bandwidth scaling is limited beyond 4 concurrent operations [14]. Because of these properties, a serially executed workflow with a high degree of concurrent operations experiences bandwidth degradation. For workflows in which the simulation's compute phase is much longer than the I/O phase which issues PMEM access operations, the benefits of parallel computation outweigh the loss of PMEM bandwidth during I/O. For workflows in which the components are I/O intensive, we must carefully control concurrency levels to limit contention in PMEM. By limiting performance degradation while accessing PMEM, we can improve the effective memory bandwidth. And this in turn minimizes the end-to-end workflow runtime.

The effective PMEM bandwidth is also impacted by the locality of the PMEM controller to the issuing CPU. Any memory access to PMEM on a remote NUMA node is much slower with as few as three concurrent memory operations. More than three concurrent memory operations quickly decrease effective bandwidth to under 1GB/s. This is caused by contention for interconnect links [3]. Writes are more adversely affected by remote NUMA memory operations than reads because of their lower baseline performance. Our measurements show a 15X drop in write bandwidth with 24 concurrent write operations compared to a 1.3X slowdown for read operations.

Access latency. Because the PMEM controller includes buffering for cache line writes, writing a small amount of data to idle PMEM has a write latency of 90 ns compared to a read latency of 169 ns. As a result, for workflows that do not throttle memory bandwidth, reducing read latency helps improve overall workflow performance.

III. Goals

The previous section outlined scheduling decisions that expose different PMEM-related performance bottlenecks for co-running workflow components. These are related either to the reduction in the effective PMEM bandwidth available to one or more (including all) workflow components, or to the change in access latencies. Motivated by these observations, the goal of this paper is to explore the following questions:

Q1 How significantly can the workflow performance be affected due to these bottlenecks?

Q2 How are workflow scheduling decisions impacted by the workflow characteristics, concerning the shared PMEM use?

Q3 What are the best practice recommendations that can maximize the benefits that a workflow can experience from using PMEM for its streaming I/O?

In order to answer these questions, in the remainder of the paper we first identify the workflow characteristics which are most related to its performance sensitivity to PMEM's use, and next perform detailed experimental analysis to demonstrate and scope the impact of PMEM and to establish recommendations for scheduling systems.

IV. Representative Workflows

In order to perform the analyses, we first identify the most relevant parameters of the streaming I/O patterns that determine the sensitivity of the workflow performance to the effective performance of PMEM, and then construct a suite of workflows which exhibit different characteristics in the parameter space.

A. Workflow Streaming I/O Parameters

The following parameters of a workflow component determine its sensitivity to changes in effective PMEM bandwidth or access latency:

Concurrency. High concurrency can reduce compute time for certain applications. However, increased I/O concurrency also means increased contention in the Optane internal cache. The thrashing of the internal cache is known to reduce the effective bandwidth of Optane. The concurrency of the workflow components are statically determined via parameters in workflow launch scripts without actually requiring a run.

Object size. The size of the objects streamed across the workflow has implications on the PMEM access granularity and on the overheads incurred in the I/O software stack. Each streaming I/O operation performed over PMEM includes software costs required by the PMEM software stack (e.g., NVStream [1] or NOVAfs [15]) to update I/O metadata in various internal datastructures, as well as costs to perform the actual PMEM read (load) or write (store) operations. When streaming I/O with large object sizes, the software overheads executed by the CPU are drastically reduced, and PMEM bandwidth may be saturated even with small number of threads. When streaming small objects, the aggregate software overheads of each I/O operation can dominate. In particular, in the case of filesystems like NOVAfs, that require a user to kernel space border crossing, the overhead is even more significant. The object size is determined by the workflow, based on the composition of the objects forming the snapshot streamed across the workflow during each iteration, and by the configuration of the I/O stack, determining the granularity at which objects are written/read in and out of PMEM.

Iteration cycle composition. Each iteration cycle of both simulation and analytics components is composed of compute and I/O phases. If I/O time $>>$compute time, there is more overlap among concurrent access operations to PMEM. This causes contention for Optane resources and reduces effective bandwidth and increases access latency. On the other hand, if compute time $>>$I/O time, there is an opportunity to hide the I/O time when executing the workflow in the parallel mode without a significant reduction in the effective bandwidth.

B. Workflow Benchmarks

Based on the above characteristics, we assemble a suite of workflows comprising simulation and analytics components which exhibit different properties. We do this using microbenchmarks and real application kernels.

Microbenchmark. To investigate the impact of object size and thread concurrency on workflow runtime and performance, we use a workflow microbenchmark that allows us to vary different parameters. The simulation and analytics components of the microbenchmark are configured with the same number of threads. Their level of concurrency is set to low, medium or high using 8, 16 or 24 threads respectively. Each iteration stream produces a 1GB snapshot using either small (2 KB) or large (64 MB) objects. Each thread in the microbenchmark performs 10 iterations, hence, a higher number of threads increases the amount of data output and pressure on Optane bandwidth. Both writers and readers perform only I/O and do not have a compute kernel.

Application kernels We also use several workflow benchmarks based on real HPC applications. For the simulation components we use the following applications:

Gyrokinetic Toroidal Code (GTC) is a three-dimensional particle-in-cell application used in micro-turbulence fusion device studies. The checkpoint data constitutes of 2D/3D arrays. We change the original input parameters *npartdom*, *micell* and *mecell* in constant factors to weak scale the workload size of the benchmark. Here, we use GTC to represent a class of applications that whose I/O consists of a few relatively large objects.

miniAMR applies a seven-point stencil calculation on a unit cube computational domain, which is divided into blocks. The blocks all have the same number of cells in each direction and communicate ghost values with neighboring blocks. With adaptive mesh refinement, the blocks can represent different levels of refinement in the larger mesh. In our analysis, miniAMR represents a class of applications that whose I/O consists of a many relatively small objects.

Analytics kernels. The application workflow benchmarks use two different analytics kernels used in our prior work [5].

Read-only kernel. This is a read-only kernel which reads objects from a single writer. This kernel is employed with GTC + Read Only and miniAMR + Read-Only workflows to study the impact of I/O heavy analytics kernel with an insignificant compute phase.

Compute-heavy kernel. This analytics kernel performs matrix multiplication over objects read from individual writer. Our use of this kernel is meant as a compute-intensive stand-in, as opposed to something that might be more appropriate for the science, so that our analysis could focus on the 1-to-1 data flow scenario with appropriate levels of computation that might act in place of heavier analysis. There are different version of this kernel for GTC and miniAMR. When coupled with GTC, it performs 10 million matrix multiplication of large 2D arrays. With miniAMR it performs only 5 matrix multiplications on each object. However, since the miniAMR snapshots are made of 528K small objects the compute phase length is still relatively large.

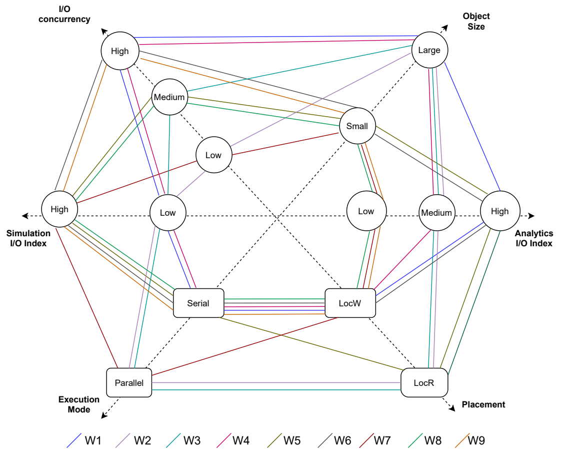


Fig. 3: Workflow parameter space

C. *Workflow Suite*

Using the above microbenchmarks and application kernels, we construct a suite of workflows. Each workflow in the testing suite corresponds to an appropriate combination of a simulation and analytics component, which are then further configured to execute at different levels of concurrency, i.e., with different number of MPI ranks. In the discussion in this paper, we interchangeably use ranks or threads, to refer to these separate execution contexts. The workflows are constructed such that both workflow components are configured with the same number of ranks (i.e., corresponding to a 1:1 I/O exchange), and both components access the complete object (i.e., they operate using the same I/O granularity).

Figure 3 shows the parameter and configuration options exercised by the workflows. The four axes of Simulation I/O Index, Concurrency, Object Size and Analytics I/O index represent workflow parameters. The iteration time of the simulation and analytics kernels is composed of compute and I/O phases. We define I/O index as the ratio of I/O time/Iteration time when the application is executing standalone (i.e., as in *serial*, with no contention) and with node-local access to PMEM. A given workflow would be represented by a sequence of points along each of the axes. For instance, GTC + Read-Only at 24 threads, W1 in figure 3, has low Simulation I/O Index, high concurrency uses large 64MB objects with a high Analytics I/O index. For this workload we determine (shown later in Figure 10a in Section VII) that it should be scheduled in Serial mode with Simulation placed local to the PMEM ($S - locW$).

Since we have 18 total workloads, for legibility, the figure does not represent a complete radar chart, and shows the 9 application kernel workflows only. Instead, the goal of figure is to illustrate the wide spectrum of parameter combinations provided by the workflows used in our experiments. Note that each node on each axis has a fan out of at least 2, illustrating that the suite includes configurations which overlap and differ in other workload parameters.

We also show the scheduling decisions via two additional axes: Execution Mode and Placement. A complete radar chart illustration of all workflows in the suite does confirm that no one single parameter of the workflow configuration space determines the configuration of the scheduling decision.

V. Methodology

Hardware testbed. All experiments are conducted on a testbed with dual-socket Intel Xeon Scalable processors with 28 physical cores per socket. There are two memory controllers per socket each having three channels. Each channel has two DIMM slots. 512GB Optane DIMMs configured in App-Direct Mode. NOVA Linux kernel based on 5.0.

Software stack. From the software stack, PMEM can be accessed via a file system or via an object-based API. Since the choice of the API further impacts the access characteristics of PMEM, we perform the evaluations using a PMEM-aware file system **NOVAfs**, and a PMEM-aware in-memory object store, further specialized for streaming I/O, **NVStream**.

NOVAfs [1] is a log-structured filesystem designed for persistent memory. It maintains a separate log file for each inode to improve concurrency while taking advantage of fast random access provided by persistent memory. The logs are used to provide atomicity of metadata, data, and map atomicity. The data is stored outside of logs to reduce garbage collection costs. NOVAfs supports DAX (direct access) mode to by-pass page cache and creates file mappings that allow load/store access to persistent memory.

NVStream [15] is a data transport designed for HPC workflows over persistent memory. Filesystems, when used as data transport, incur overheads such as POSIX-based system call and journaling/logging costs. NVStream is a log-based versioned object store in userspace that is optimized for streaming workflows. NVStream uses non-temporal stores to maximize persistent memory bandwidth utilization. Furthermore, this avoids cache pollution since simulation applications don't read back snapshot data and they are immutable.

Measurements. The analyses are performed by measuring the execution of the workflows in the suite in each of the four scheduling configurations, $S - LocW|R$ and $P - LocW|R$. In each workflow, the simulation (writer) component instantiates objects in the beginning and periodically produces output in the form of a checkpoint to the PMEM I/O channel. The I/O data is subsequently consumed by the analytics (readers) component. Each checkpoint stores all output objects with a new version number. The writers I/O time is the time to write all objects from DRAM to persistent memory. The readers read individual objects in sequence from the I/O channel. The readers I/O time is the time to read an individual object from persistent memory to local DRAM.

For each workflow, depending on the deployment configuration, writer and reader processes (MPI ranks) are always pinned to cores either local or remote with respect to the persistent memory. The configuration also specifies whether writers and readers are scheduled to execute serially (in $S - LocW$ and $P - LocW$), or parallelly (in $S - LocR$ and $P - LocR$).

We measure the end-to-end execution time for a workflow. In the case of sequentially scheduled workflows, the execution time is shown as a split bar graph with separate writer and reader components. Inspection of split costs gives us insights on the impact of writers' or readers' PMEM locality on write/read performance, and is used to inform placement decisions. For parallel, the execution time is measured as the total time for the readers and writers to complete.

VI. Observations

A. S-LocW: Serial execution in local-write and remote-read

Workflows with I/O intensive simulation and analytics kernels at moderate and high I/O concurrency levels should be executed in $S - LocW$ configuration.

With little opportunity to hide I/O operations with compute phases, the workflow is bandwidth constrained. Thus, remote writes are more adversely affected by PMEM bandwidth contention relative to remote reads. Placing writers local to PMEM maximizes the available bandwidth. Further, serial execution performance is better because co-scheduling analytics increases contention for Optane and reduces the effective bandwidth.

The 64MB workflows shown in figures 4a,4b and 4c exhibit these characteristics. During parallel execution, interconnect contention creates back pressure on the Optane internal cache. The remote reads hold resources that also slow writes. Given the large object size, the software overhead is minimal, therefore minimizing contention for interconnect and Optane bandwidth is critical to achieve good performance for this workflow. Since this workload is composed of large objects, remote-writes experience greater interconnect contention and slowdown than remote-reads. With moderate (16) and large (24) number of threads, serial execution with local writes ($S - LocW$) provides the best runtime across placements, up to 2.5$\times$ better than other scenarios.

Figures 6c and 7c have similar impact on PMEM behavior at high I/O concurrency levels since GTC uses 229MB objects. Thus, with many concurrent MPI ranks the workload is bandwidth constrained because remote-writes begin to dominate the overall runtime, similar to the 64MB object size workflow as shown in figure 4. Hence, $S - LocW$ is the optimal placement and is 6% faster than remote-write/local-read with serial execution, or more for parallel executions.

At high I/O concurrency levels miniAMR + Read-Only and miniAMR + MatrixMult, shown in Figures 8c, 9c, are also able to create a similar contention of PMEM accesses as the GTC workflows. This is because although miniAMR uses 4.5KB objects, it has an I/O heavy simulation kernel. Figure 8c shows that at 24 threads, the cost of remote writes dominates the runtime in remote-write configurations. The workflow begins to saturate the write bandwidth of Optane. Hence, serial execution in $S - LocW$ is 25% faster than in $S - LocR$.

B. S-LocR: Serial execution in remote-write and local-read

Workflows which do not constrain the Optane bandwidth at moderate and high I/O concurrency levels should be executed in $S - LocR$ configuration.

Workflows that either generate I/O in small granularity or have long compute kernels overlapping with large granularity I/O do not throttle the Optane bandwidth. Under such conditions it is important to prioritize reads over writes since reads have a higher access latency when PMEM is not bandwidth constrained. Hence, remote-write/local-read placement is chosen. In this case serial execution typically provides better performance because at moderate to high I/O concurrency levels co-scheduling analytics increases contention for Optane and reduces the effective bandwidth.

The 2K workflow, shown in figure 5c, does not saturate the bandwidth, even with high concurrently levels. Because there are a large number of small objects, this workflow has high software overhead. Thus, the high latency for remote-read degrades workflow performance. In contrast to reads, writes are marked complete once they are stored in the PMEM controller. This differs from reads, which are completed after data is returned from PMEM. Hence, $S-LocR$ is the optimal placement strategy for this workload. Although the bandwidth is not constrained, reduced I/O concurrency due to serial execution helps with contention for Optane internal cache: at I/O concurrency of 24 threads, $S-LocR$ provides 11.5% faster runtime than parallel.

The GTC + Read-Only workflow at medium concurrency level, shown in figure 6b also exhibits similar I/O behavior. Although GTC uses large object sizes, due to its compute intensive simulation kernel the workflow does not saturate the bandwidth at medium I/O concurrency level. Figure 6b shows that at 16 threads $S - LocR$ is 6-7% faster than parallel and provides optimal runtime for GTC + Read-Only workflow.

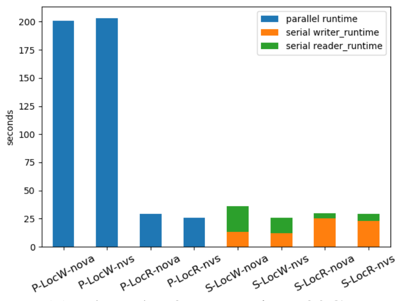
Figure 8b shows miniAMR + Read-Only workflow at medium I/O concurrency level. This workflow is similar to the 2K workflow since miniAMR uses small objects and hence exhibits similar I/O behavior. The figure shows that $S - LocR$ is preferable and is 6% faster than the second best configuration $P - LocR$.

C. P-LocW: Parallel execution in local-write and remote-read

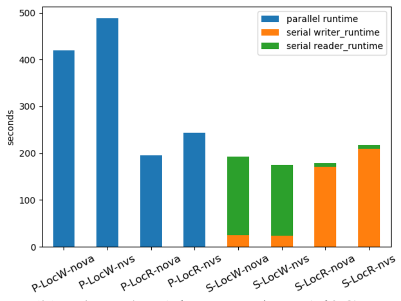
Workflows having I/O intensive simulation with compute intensive analytics at low concurrency levels should be executed in P-LocW configuration.

In this workload, the concurrency levels are low, hence it benefits from parallel execution. The analytics kernels has interleaving compute between PMEM reads reducing contention. As a result, keeping simulation PMEM writes local while having analytics perform PMEM remote reads is the better trade-off.

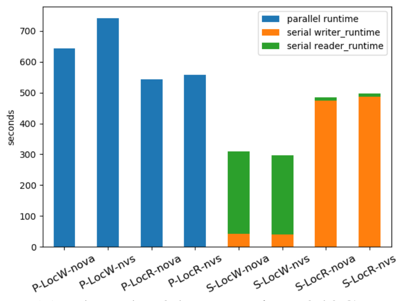
miniAMR + Matrixmult at 8 threads has high simulation I/O Index and low analytics I/O index. As shown in Figure 9a, the $P - LocW$ configuration is 7% better than the next best alternative, $P - LocR$.



(a) Threads: 8, Data size: 80GB

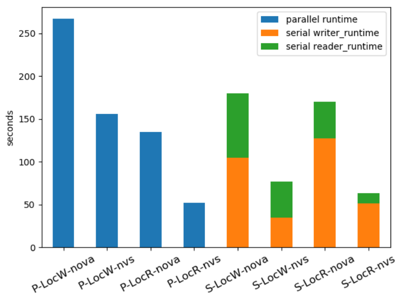


(b) Threads: 16, Data size: 160GB

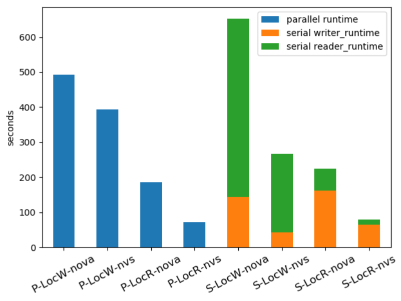


(c) Threads: 24, Data size: 240GB

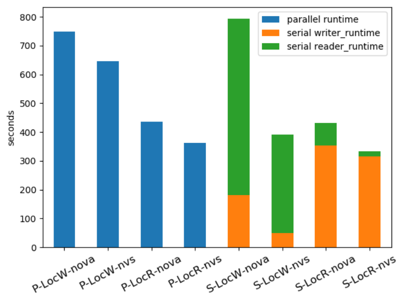
Fig. 4: Benchmark Writer + Reader with 64MB objects: Runtime



(a) Threads: 8, Data size: 80GB

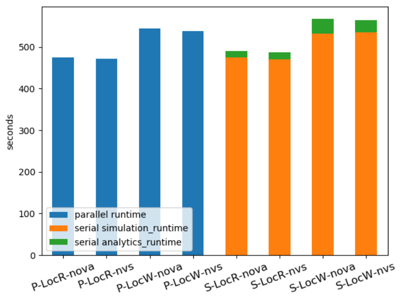


(b) Threads: 16, Data size: 160GB

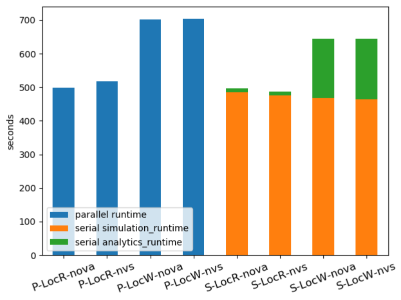


(c) Threads: 24, Data size: 240GB

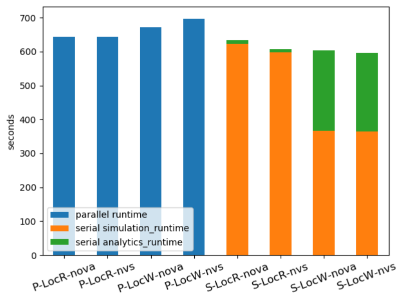
Fig. 5: Benchmark Writer + Reader with 2K objects: Runtime



(a) Threads: 8



(b) Threads: 16



(c) Threads: 24

Fig. 6: GTC + Read only: Runtime

D. P-LocR: Parallel execution in remote-write and local-read

Workflows at low and moderate levels of I/O concurrency which do not constrain the I/O bandwidth should be executed in $P - LocR$ configuration.

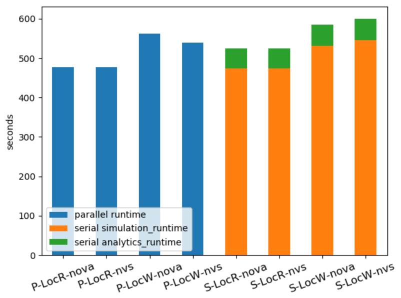
For this workload configuration optimizing the read latency is important because the workflows are not bandwidth constrained. This is similar to the workflows with $S - LocR$ optimal configuration, though the reduced I/O concurrency levels mean there is some performance benefit in using parallel execution.

Figures 5a and 5b show 2K workflows at low and medium levels of I/O concurrency. Since these 2K workflows use a large number of small objects the bandwidth is not saturated and due to their low and medium I/O concurrency there isn't high contention for Optane internal cache. This allows leveraging parallel execution at low and moderate concurrency levels. $P - LocR$ provides optimal runtime for 8 and 16 threads as shown in Figures 5a and 5b, and is 10 to 14% faster than $S - LocR$. miniAMR + Read-Only shown in figure 8a at low concurrency level exhibits similar I/O behavior.

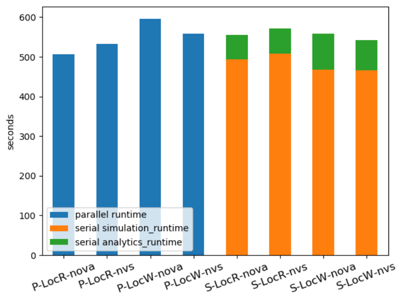
Figures 6a and 7a show that at low thread concurrency levels, both GTC+ Read only and GTC + matrixmult have the optimal runtime using $P - LocR$. Despite GTC using large objects, the performance is dominated by the long compute phase, and thus this workflow configuration is not bandwidth saturated. In addition, the I/O concurrency is low enough that parallel execution yields better runtime performance, in spite of the remote-write data placement. Parallel execution overlaps simulation and analytics kernels and is 3-9% faster than serial execution. Even with 16 threads, as shown in figure 7b, despite the resulting increase in PMEM bandwidth pressure, the overlap opportunities allow for parallel execution to provide optimal performance.

VII. Performance Implications

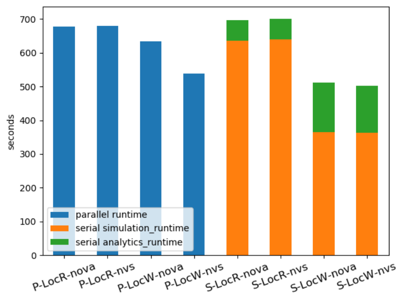
No single optimal configuration Figure 10 shows workflow runtimes normalized to the runtime of the best configuration. The graphs demonstrate that there is no single configuration optimal across workflows. Further, always prioritizing simulation or analytics may not be optimal for the overall workflow's runtime. For instance, when comparing the results in 10a and 10b, we observe that changing the analytics kernel used with



(a) Threads: 8

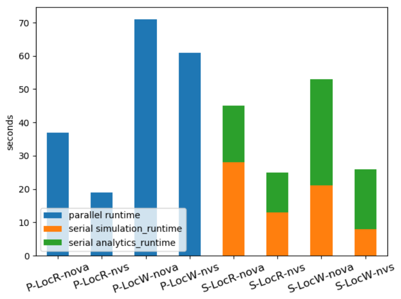


(b) Threads: 16

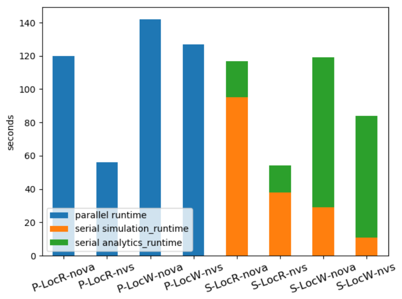


(c) Threads: 24

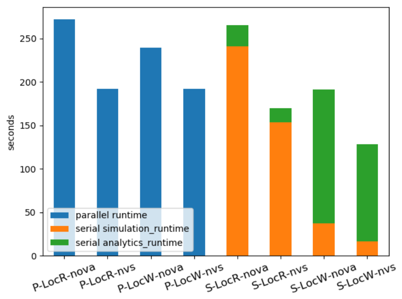
Fig. 7: GTC + matrixmult: Runtime



(a) Threads: 8

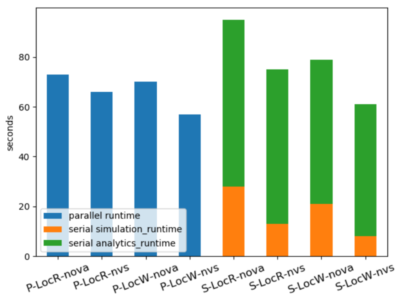


(b) Threads: 16

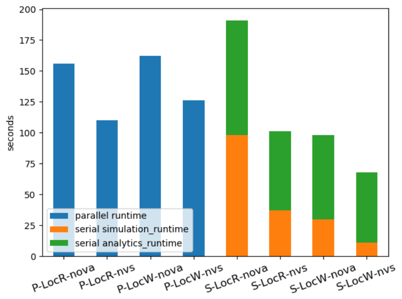


(c) Threads: 24

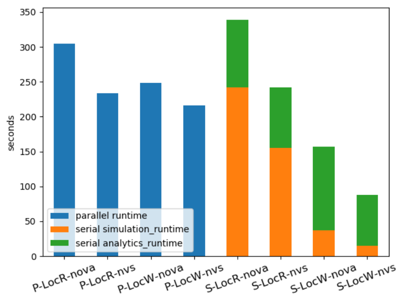
Fig. 8: miniAMR + Read only: Runtime



(a) Threads: 8



(b) Threads: 16



(c) Threads: 24

Fig. 9: miniAMR + matrixmult: Runtime

GTC will lead to 24% loss of performance, unless we make corresponding adjustment in the scheduling and placement decision (e.g., comparing $S-LocR$ and $P-LocW$ for the 16 thread workloads). A similar comparison for miniAMR in Figures 10c and 10d shows that failure to properly configure the workload placement and scheduling based on its use of PMEM can lead to a slowdown of up to 70%.

Relevant for workflows with different parameters. Our benchmarks and HPC workflows cover a wide spectrum of workflow parameters in terms of I/O index, object size, and I/O concurrency levels as shown in figure 3. Workflows with the same I/O indexes can create different I/O bottlenecks when varying the I/O concurrency levels. We show that choosing the right configuration is very relevant across workflows and workflow parameters.

Observations not tied to a particular storage mechanism. We use two mechanisms, NOVA and NVStream, to show that the problem of choosing optimal configuration is relevant across storage mechanisms. In addition, we also show that the choice of a configuration ($S-LocW-P-LocR$) is not optimal entirely due to any particular idiosyncrasy of the storage mechanism. We actually see similar trends with both NOVA and NVStream for large objects, especially with GTC. However, NVStream reduces the software I/O costs compared to NOVAfs, and the change in software overheads has an impact on the observations made for workflows which perform I/O using many small objects.

VIII. Summary and Recommendations

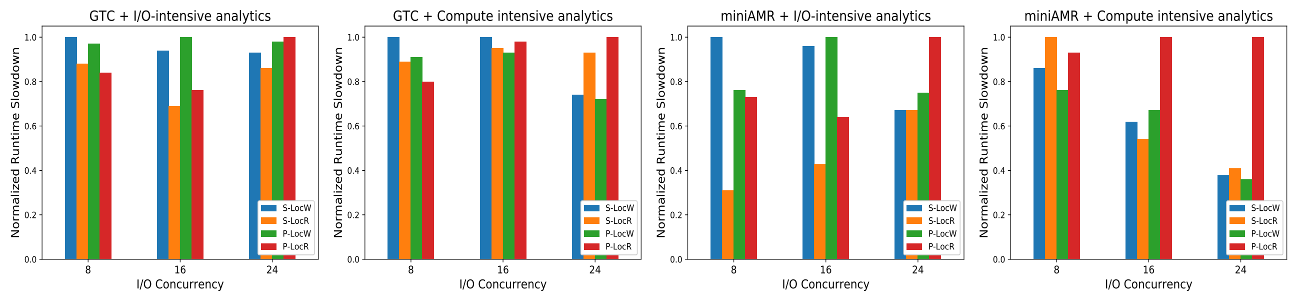
The observations made in Section VI open up new opportunities for smart workflow scheduling for platforms with shared PMEM resources. In order to aid with such scheduler designs, we summarize the observations as follows:

Maximize effective bandwidth by limiting concurrent device accesses. Workflows which have *high levels of concurrency in simulation and analytics kernels, should be executed in serial modes* either using $S-LocW$ or $S-LocR$ configurations. High levels of concurrency, i.e. using all cores in the CPU, cause contention for Optane resources such as the internal cache. Serial execution mode limits contention by having analytics begin execution after simulation has completed.

Placement choices at high concurrency levels are made based on bandwidth utilization. *Workflows which constrain*

TABLE II: Configuration recommendations for Workflows

#	Sim Compute	Sim Write	Analytics Compute	Analytics Read	Object Size	Concurrency	Config	Illustrative Workflows
1	Nil	high	Nil	high	large	low, medium or high	$S - LocW$	64MB workflows: Fig 4a,4b,4c
2	high	low	low to high	medium, high	large	high	$S - LocW$	GTC + Read-Only: Fig 6c GTC+MatrixMult: Fig 7c
3	low	high	low	high	small	high	$S - LocW$	miniAMR + Read-Only Fig 8c
4	low	high	high	low	small	medium, high	$S - LocW$	miniAMR + Matrixmult Fig 9b,9c
5	low	high	Nil	high	small	high	$S - LocR$	2K workflows: Fig 5c
6	high	low	low	high	large	medium	$S - LocR$	GTC + Read-Only: Fig 6b
7	low	high	low	high	small	medium	$S - LocR$	miniAMR + Read-Only Fig 8b
8	low	high	high	low	small	low	$P - LocW$	miniAMR + Matrixmult Fig 9a
9	Nil, low	high	Nil	medium, high	small	low, medium	$P - LocR$	2K workflows: Fig 5a, 5b miniAMR+Read-Only: Fig 8a
10	high	low	low to high	high	large	low, medium	$P - LocR$	GTC + Read-Only: Fig 6a GTC+MatrixMult: Fig 7a,7b



(a) (b) (c) (d)

Fig. 10: Workflow runtime normalized to the fastest configuration

the bandwidth should prioritize writes over reads using local-write/remote read placement. Remote writes are more adversely affected than remote reads during bandwidth contention. However, *when the bandwidth is not the bottleneck, reads should be prioritized*. This is because writes are marked complete and return as soon as they are placed in the iMC queue. However, remote reads will have to wait until the data is fetched from Optane media.

Workflows with simulation and analytics kernels at *low levels of concurrency should be executed in parallel modes* either using $P-LocW$ or $P-LocR$. At such low concurrency levels for most workflows, i.e., using 8 CPU cores or less for each application, the slowdown caused due to contention is minimal. Hence, taking advantage of parallel execution is the better trade-off.

The number of MPI ranks of simulation and analytics kernel is a workflow parameter which determines the level of concurrent access to PMEM. However, the actual level of concurrency experienced by PMEM is a complex function of number of MPI ranks, software overhead to perform writes/reads which is based on object size and interleaving compute in simulation and analytics application.

High software stack I/O overheads lower PMEM contention and allow for concurrent executions. The 2K workflow at 16 MPI ranks has large number (528K) of small objects in a snapshot resulting in high software overhead. This reduces the effective level of concurrent access to PMEM. Hence, it is executed in parallel mode in $P - LocR$ configuration. However, GTC + Read-Only workflow also at 16 MPI ranks is executed serially in $S - LocR$ configuration. This is because GTC uses large objects which require minimal software overhead. And hence PMEM experiences a higher level of concurrent access compared to the 2K workflow. Another case in point is the 64MB workflow at 8 MPI ranks. Although the concurrency level is low, there is very minimal software overhead and also there are no compute phases. Hence, it is executed in the $S - LocW$ configuration.

Interleaved compute hides effects of access contention and high remote latency. Interleaving compute with I/O during each interval, reduces the effective device access contention, and allows for parallel execution. In addition, the compute phase allows for hiding of remote access latencies. For instance, GTC with low number of threads can be scheduled with $P - LocR$ which prioritizes analytics and allows for parallel execution, although its I/O phase is similar to the purely I/O-intensive microbenchmark. A compute-intensive analytics also allows for the PMEM placement to prioritize simulation. For instance using MatrixMult with miniARM prefers $P - LocW$ (see Figure 9a), whereas using Read-Only analytics prefers $P - LocR$ (see Figure 8a).

IX. Related Work

The commercial release of Optane was followed by several works which investigated the device performance characteristics. Yang *et al.* [2] measured basic microarchitectural parameters of latency and bandwidth. Their work listed the key insights and best practices to take advantage of Optane performance characteristics. Peng *et al.* [3] studied the impact of Optane on in-memory graph processing in memory mode. They devised fine-grained NUMA-aware allocation and write isolation policies to distribute traffic between Optane and DRAM to provide higher bandwidth and capacity. Further, they present a roofline model for energy and power efficiency across different distributions of read-only workload. In the context of HPC, Wu *et al.* [16] analyzed the performance of I/O-intensive HPC applications over Optane as a block device. They compared Optane and HDD for MPI I/O and checkpoint workload, and investigated the impact of Optane on PVFS2, a parallel file system. None of the prior work has explicitly focused on the end-to-end effects of using Optane for HPC workflow I/O, which is the focus of our investigation.

X. Conclusions

Persistent memories will be integrated into future HPC systems to address data demand, as they provide large capacity at latencies comparable to DRAM. After years of speculation about the technical parameters of these components, they have finally become commercially available, through the release of the Intel Optane DC Persistent Memory. This paper takes an experimental approach in evaluating the impact of Intel Optane PM on the performance of HPC workflows. We demonstrated that achieved performance can vary up to 70% depending on how workflow components are configured to use the shared PMEM resource. Further, we showed that the preferred configuration decision depends on a number of workflow parameters. Using experimental analysis based on a suite of workflow benchmarks, we made a set of recommendations that have to be considered by future workflow schedulers for PMEM-based HPC systems. Our future work is to explore how these recommendations can be practically incorporated in scheduling systems.

Acknowledgements

We thank the anonymous reviewers for their valuable feedback and comments. This work is partially supported by NSF awards SPX-1822972 and CNS-2016701, and via access to an Optane PMEM hardware testbed provided by Intel.

References

[1] P. Fernando, A. Gavrilovska, S. Kannan, and G. Eisenhauer, "Nvstream: Accelerating hpc workflows with nvram-based transport for streaming objects," in *Proceedings of the 27th International Symposium on High-Performance Parallel and Distributed Computing*, 2018.

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

[3] I. B. Peng, M. B. Gokhale, and E. W. Green, "System evaluation of the intel optane byte-addressable nvm," in *Proceedings of the International Symposium on Memory Systems*, 2019.

[4] S. Klasky, H. Abbasi, J. Logan, M. Parashar, K. Schwan, A. Shoshani, M. Wolf, S. Ahern, I. Altintas, W. Bethel *et al.*, "In situ data processing for extreme-scale computing," *Scientific Discovery through Advanced Computing Program (SciDAC11)*, 2011.

[5] A. Champsaur, J. Lofstead, J. Dayal, M. Wolf, G. Eisenhauer, P. Widener, and A. Gavrilovska, "Smartblock: An approach to standardizing in situ workflow components," in *2017 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*. IEEE, 2017.

[6] T. D. Doudali and A. Gavrilovska, "Comerge: Toward efficient data placement in shared heterogeneous memory systems," in *Proceedings of the International Symposium on Memory Systems*, 2017.

[7] T. D. Doudali, S. Blagodurov, A. Vishnu, S. Gurumurthi, and A. Gavrilovska, "Kleio: A hybrid memory page scheduler with machine intelligence," in *Proceedings of the 28th International Symposium on High-Performance Parallel and Distributed Computing*, 2019.

[8] T. D. Doudali and A. Gavrilovska, "Mnemo: Boosting memory cost efficiency in hybrid memory systems," in *2019 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*. IEEE, 2019.

[9] S. Kannan, N. Bhat, A. Gavrilovska, A. Arpaci-Dusseau, and R. Arpaci-Dusseau, "Redesigning lsms for nonvolatile memory with novelsm," in *2018 {USENIX} Annual Technical Conference ({USENIX}{ATC} 18)*, 2018.

[10] J. Kim, K. Sajjapongse, S. Lee, and J. S. Vetter, "Design and implementation of papyrus: Parallel aggregate persistent storage," in *2017 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. IEEE, 2017.

[11] K. Chandrasekar, B. Seshasayee, A. Gavrilovska, and K. Schwan, "Improving data reuse in co-located applications with progress-driven scheduling," in *RESPA Workshop at SC*, 2015.

[12] A. Goswami, Y. Tian, K. Schwan, F. Zheng, J. Young, M. Wolf, G. Eisenhauer, and S. Klasky, "Landrush: Rethinking In-Situ Analysis for GPGPU Workflows," in *2016 16th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid)*, 2016.

[13] A. Ranadive, A. Gavrilovska, and K. Schwan, "FaReS: Fair Resource Scheduling for VMM-Bypass InfiniBand Devices," in *2010 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing*, 2010.

[14] J. Izraelevitz, J. Yang, L. Zhang, J. Kim, X. Liu, A. Memaripour, Y. J. Soh, Z. Wang, Y. Xu, S. R. Dulloor *et al.*, "Basic performance measurements of the intel optane dc persistent memory module," *arXiv preprint arXiv:1903.05714*, 2019.

[15] J. Xu and S. Swanson, "{NOVA}: A log-structured file system for hybrid volatile/non-volatile main memories," in *14th {USENIX} Conference on File and Storage Technologies ({FAST} 16)*, 2016.

[16] K. Wu, F. Ober, S. Hamlin, and D. Li, "Early evaluation of intel optane non-volatile memory with hpc i/o workloads," *arXiv preprint arXiv:1708.02199*, 2017.