Exploiting the Performance Benefits of Storage Class Memory for HPC and HPDA Workflows

DOI: 10.14529/jsfi180105

Exploiting the Performance Benefits of Storage Class Memory for HPC and HPDA Workflows

Michèle Weiland[1], *Adrian Jackson*[1], *Nick Johnson*[1], *Mark Parsons*[1]

Byte-addressable storage class memory (SCM) is an upcoming technology that will transform the memory and storage hierarchy of HPC systems by dramatically reducing the latency gap between DRAM and persistent storage. In this paper, we discuss general SCM characteristics, including the different hardware configurations and data access mechanisms SCM is likely to provide. We outline the performance challenges I/O requirements place on traditional scientific workflows and present how data access through SCM can have a beneficial impact on the performance of such workflows, in particular those with large scale data dependencies. We describe the system software components that are required to enabled workflow and data aware resource allocation scheduling in order to optimise both system throughput and time to solution for individual applications; these include a data scheduler and data movers. We also present an illustration of the performance improvement potential of the technology, based on initial workflow performance benchmarks with I/O dependencies.

Keywords: NVRAM, 3D XPoint, SCM, workflows, resource scheduling.

Introduction

Today's supercomputers offer a computing environment that focuses on compute performance first and foremost. The advent of byte-addressable non-volatile memory however means that in the coming years supercomputers will have sufficient memory capacity per compute node to no longer be exclusively used (and useful) for high-performance scientific computation (HPC), but also for high-performance data analytics (HPDA). Rather than simply ingesting data that was generated on a different system, input data for simulations will be prepared directly on the supercomputer where the simulation will be executed, and simulation output will be analysed and post-processed there as well. We predict that the mix of applications running on supercomputers will become broader: in addition to the largely compute intensive HPC applications, there will be memory and I/O intensive HPDA applications as full scientific workflows are enabled on a single system. This break in the status quo motivates the contributions of this paper: a discussion of the performance benefits of non-volatile memory, in particular with a view to optimising the end-to-end performance of workflows with complex compute and data dependencies; and a description of the system software infrastructure that is necessary to support them.

1. Storage Class Memory

Byte-addressable, non-volatile memory (hereafter referred to as Storage Class Memory, or SCM) represents the latest advance in memory technologies. SCM promises to deliver both greater performance and endurance than existing storage technologies, as well as increased density in comparison to DRAM. Compute platforms with SCM will have access to non-volatile memory that is capable of storing several TBs of data per node. This very large memory capacity for servers, and long term high-performance persistent storage within the memory space of the servers, means that new techniques for performing I/O will emerge. SCM enables Direct

[1]EPCC, The University of Edinburgh, Edinburgh, United Kingdom

access (DAX) from applications to individual bytes of data; this is fundamentally different from the block-oriented way I/O is currently implemented [9].

Several manufacturers are working on delivering byte-addressable SCM to the market within the next few years. The development that is furthest advanced to date is the result of a collaboration by Intel and Micron, the 3D XPoint^{TM} NVDIMM [3]. As the name implies, this SCM sits in the DIMM slots next to the CPU and alongside DRAM, with access to the NVDIMM space managed via the processor's memory controller. Unlike DRAM however, data that is stored on the NVDIMM is persistent, which means that it can be used as a (potentially long-term) storage environment as well as memory. 3D XPoint^{TM} NVDIMMs can be used in two different modes [6], which have implications for how applications can exploit these memory spaces. Changing between the two modes requires a system reboot.

1.1. Data Access

SCM has the potential to enable synchronous byte-level I/O, moving away from the asynchronous block-based file I/O applications currently rely on. In current asynchronous I/O, applications pass data to the operating system (O/S) which then uses driver software to issue an I/O command, adding the I/O request into a queue on a hardware controller. The hardware controller will process that command when ready, notifying the O/S that the I/O operation has finished through an interrupt to the device driver.

SCM can be accessed simply by using a load or store instruction, as with any other memory operation an application undertakes. This may require an additional instruction to ensure the data is persistent (fully committed to the non-volatile memory), if persistence is required by an application, or such persistence guarantees may be provided by the hardware through fault tolerant power supplies protecting volatile memory within the system (such as asynchronous DRAM refresh). With SCM providing significantly lower latencies than external storage devices, the traditional I/O block access model, using interrupts, becomes inefficient because of the overhead of context switches between user and kernel mode (which can take thousands of CPU cycles). Furthermore, with SCM it becomes possible to implement remote access to data stored in the memory using RDMA technology over a suitable interconnect. Using high performance networks can enable access to data stored in SCM in remote nodes faster than accessing local high performance SSDs via traditional I/O interfaces and stacks inside a node. Therefore, it is possible to use SCM to greatly improve I/O performance within a server, increase the memory capacity of a server, or provide a remote data store with high performance access for a group of servers to share. Such storage hardware can also be scaled up by adding more SCM memory in a server, or adding more nodes to the remote data store, allowing the I/O performance of a system to scale as required.

However, if SCM is provisioned in the servers in a supercomputer, there must be software support for managing data within the SCM. This includes moving data as required for the jobs running on the system, and providing the functionality to let applications run on any server and still utilise the SCM for fast I/O and storage (i.e. applications should be able to access SCM in remote nodes if the system is configured with SCM only in a subset of all nodes).

As SCM is persistent, it also has the potential to be used to implement techniques for resiliency, providing backup for data from active applications, or providing long term storage for databases or data stores required by a range of applications. With support from the system software, servers could be enabled to handle power loss without experiencing data loss, effi-

ciently and transparently recovering from power failure. Applications could resume from their latest running state and maintaining data, with little performance overhead, especially compared to current techniques of writing data to external storage devices such as high performance filesystems.

1.2. 1-Level Memory

The first of the two modes that SCM can operate in is 1-level memory, or 1LM, which views main memory (DRAM) and NVRAM as two separate memory spaces, both accessible by applications (see Fig. 1). This mode is conceptually similar to the Flat Mode configuration of the high bandwidth, on-package, MCDRAM in current Intel Xeon Phi™ processors (code name Knights Landing or KNL) [10]. DRAM is managed via standard memory APIs, such as *malloc*, and represents the only visible memory space for the operating system. The NVRAM on the other hand is managed by persistent memory and filesystem APIs, such as *pmem i/o* [8] and *mmap*, and presents the non-volatile part of the system memory. Both allow access via direct CPU load and store instructions. In order to take advantage of SCM in 1LM mode, either the system software, or the applications have to be adapted to be able to manually use these two distinct address spaces.

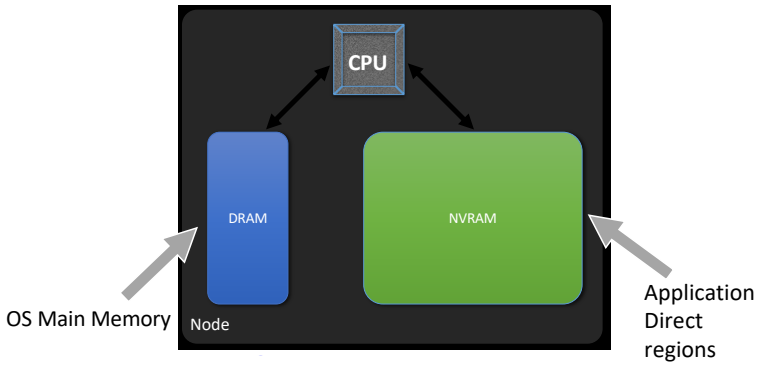

Figure 1. 1LM mode, where DRAM and NVRAM are two separate memory spaces

1.3. 2-Level Memory

2-level memory, or 2LM, configures DRAM as a cache in front of the NVRAM (see Fig. 2). Applications only see the memory space of the SCM; data that is being used is transparently stored in DRAM, and moved to SCM when no longer immediately required by the memory controller (as in standard CPU caches). This is very similar to the Cache Mode configuration of MCDRAM on KNL processors. This mode of operation does not require applications to be altered to exploit the capacity of SCM, and aims to give memory access performance at near to main memory speeds whilst providing the large memory space of SCM. Exactly how well the main memory cache performs depends on the specific memory requirements and access pattern of a given application. Furthermore, in this mode the persistence of the NVRAM contents cannot be guaranteed, due to the volatile nature of the DRAM cache that, at any given time, may hold updated versions of data stored in NVRAM. Therefore, the non-volatile characteristics of SCM are not exploited in this mode of operation. In 2LM mode, it is also possible to divide the SCM space into two partitions: memory (not persistent) and “app direct” (persistent).

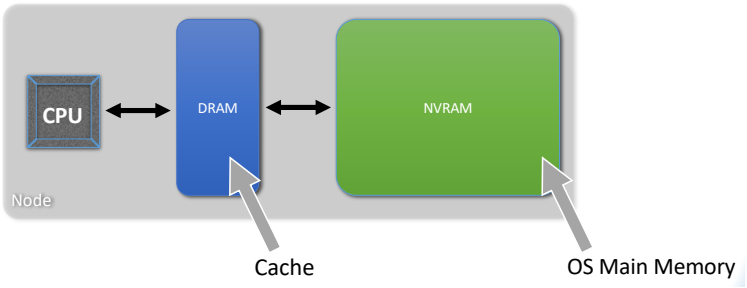

Figure 2. 2LM mode, where DRAM is a cache to the NVRAM memory space

```
#SBATCH --ntasks=64
#SBATCH --time=01:00:00

cd job1
mpirun -n 64 ./job1
cd ../job2
mpirun -n 32 ./job2
cd ../job3
mpirun -n 1 ./job3
```

Figure 3. Single SLURM script submission: there is no queuing time between jobs, but the maximum number of resources is used throughout

2. Workflows

Many scientific simulations are the result of a workflow, i.e. a series of applications that each perform a specific task within the simulation, rather than that of a single application [2]. A workflow may include steps such as data pre-processing and manipulation, computational simulation, output reduction and post-processing, or visualisation. The capacity and performance characteristics of SCM mean that steps of a workflow that would previously have been executed away from the main supercomputer (e.g. on a dedicated high-memory system) are more likely to be performed in situ. Moving the entire workflow onto a single system simplifies the simulation setup and removes the need to make simulation data available across multiple systems. Users currently set up such workflows in three different ways:

1. By putting all the jobs into a single script, requesting the maximum number of resources to be required by the steps of the workflow at any point in time (see Fig. 3).
2. By creating a chain of jobs, each requesting the correct amount of resources, for example by submitting the next job in the workflow through inserting a job submission command at the end of the preceding job (see Fig. 4).
3. By specifying dependency conditions using the job scheduler (see Fig. 5).

Only the final one of these three options implicitly supports the notion that there can be a dependency between jobs, however the dependency in this case is limited to being temporal.

Two different types of workflows can be envisaged: firstly, *monolithic* workflows are comprised of applications that each use the same amount of resources (i.e. the same number of compute nodes); and secondly, *composite* workflows that consist of applications with varying demands on the resources. The example in Fig. 3 is an instance of composite workflow.

```
#SBATCH --ntasks=64
#SBATCH --time=01:00:00

cd job1
mpirun -n 64 ./job1
cd ../job2
sbatch job2.sh
```

Figure 4. Basic job chaining in a SLURM script: the chained job will be put into the queue as if it was submitted manually

```
JOB1_ID=$(sbatch job1.sh)
echo $JOB1_ID
sbatch --dependency=afterok:$JOB1_ID job2.sh
```

Figure 5. Defining a dependency in SLURM: the scheduler uses the job ID to check that the preceding job has completed successfully, and the next job is then submitted into the queue

All of the three workflow setup approaches can have drawbacks: the first approach minimises the end-to-end runtime T_{all}, because there is a single queuing penalty T_{queue} at the start of the job which has to be added on to the time when the job is running T_{run}. However, unless all the workflow steps require the same number of compute nodes, i.e. unless the workflow is monolithic, this approach is wasteful in terms of resources, because the maximum number of compute nodes R_{max} is used for the entire duration of the job:

$$T_{all} = T_{queue} + \sum_{i=1}^{N} T_{(i,run)}, \tag{1}$$

$$R_{all} = R_{max} * T_{all}. \tag{2}$$

If there is a large discrepancy between the resources required by each workflow step (say if one of the steps is serial and another uses hundreds of nodes), not only does this approach quickly become very expensive, it also impacts the utilisation of the system, because although nodes are allocated to a job, they are not active all the time the job is executing but remain remain unavailable for other jobs.

The second and third approaches minimise resource utilisation in the composite workflow case, because for each step N, the correct amount of resources is requested. The time to completion for the workflow however will now have to include additional queuing time T_{queue} on top of compute time T_{run} for each of the N steps of the workflow. On a busy machine where queuing times are long, this approach can have a significant impact on the total time to solution:

$$R_{all} = \sum_{i=1}^{N} R_i * T_i, \tag{3}$$

$$T_{all} = \sum_{i=1}^{N} T_{(i,queue)} + T_{(i,run)}. \tag{4}$$

2.1. Workflows with Data Dependencies

On today's supercomputers, data dependencies with workflows are largely implemented by sharing data through files that are written to, and read from, a shared file system. Therefore, regardless of the approach that is taken for the execution of the workflow, T_{run} includes reading data from the file system at the start of a job, and writing results back at the end, in addition to performing the computation:

$$T_{run} = T_{I/O} + T_{compute}. \tag{5}$$

The aim is to minimise both the resource utilisation R_{all} and the time to solution T_{all} by providing true support for complex workflows within the job scheduler, and by reducing the I/O component $T_{I/O}$ of the runtime as much as possible through allowing workflows to share data without writing to a file system that is external to the compute nodes.

For supercomputers that execute a lot of data-intensive workflows, I/O performance presents a considerable performance bottleneck that the arrival of SCM will help alleviate. On a system with SCM, the data that is produced as part of a workflow can be kept on the compute nodes to be consumed in situ until the workflow concludes. In 1LM mode for instance, this could be achieved by simply using local files. In order to achieve this, the job scheduler and resource manager must understand and support the notions of both workflows and of data locality, so that individual steps of a workflow are placed on compute nodes that (ideally) already have a copy of the data.

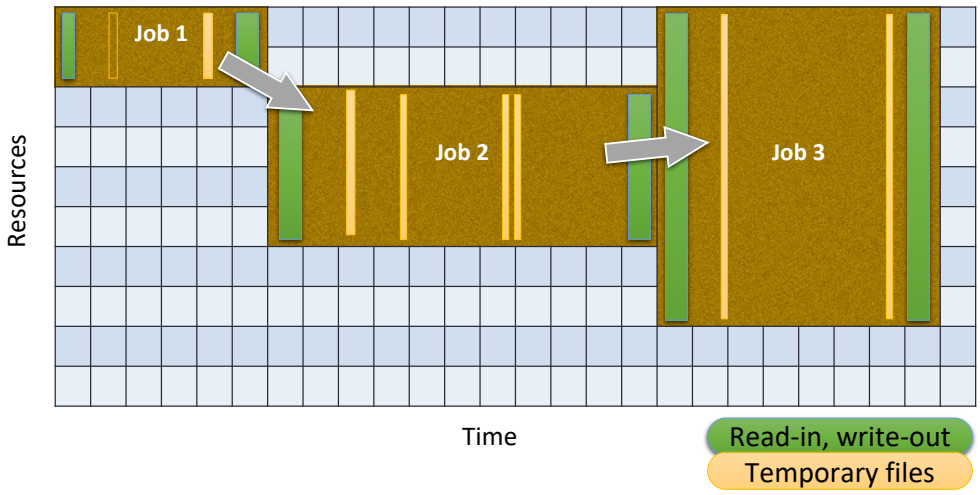

Figure 6. Example of a workflow where information is shared by writing out and reading in data. Data written by Job 1 is ingested by Job 2; and data written by Job 2 is in turn ingested by Job 3

Providing support for workflows without impacting the throughput of standard jobs is potentially complex if no restrictions are applied to the workflows. We therefore limit our support to workflows with the following properties:

1. At the time of submission of the workflow, the full workflow must be known, i.e. no extra steps can be added while the workflow is active.
2. A workflow must have data dependencies.

3. The temporal dependencies of the workflow steps must be fixed, i.e. each job must know which job(s) immediately precede(s) and follow(s) it, and this cannot be changed.
4. As part of the workflow description, the data that is shared between the different jobs is listed explicitly, and only this data forms part of the workflow.

2.2. Using SCM to Optimise Resource Usage and Time to Solution

SCM brings the opportunity for supercomputers to start offering more fundamental support for workflows with data dependencies. SCM can adopt the role of large (persistent) memory or of a fast storage device (or a combination of the two), which means that compute nodes with local SCM will be able to tackle a much wider workload than traditional HPC systems. The key benefit SCM offers is that it will allow applications to ingest and output data (in any form) with minimal involvement from the external file system. Prior to a job running, its associated input data can be pre-loaded onto the SCM of the compute nodes that will be allocated to the job. Similarly, a job can write its final output data locally to SCM; once the job has completed, and assuming the output data does not need to be read by another job, the data can be moved off the compute node and onto external storage. The benefit is that compute resources are used primarily for compute, and not for I/O, and time to solution and system throughput both improve. In theory, once data has been moved from the network attached storage to the compute node, it can remain there and be accessible until it is explicitly removed. In practice, there are a number of questions that arise, such as: who is responsible for moving data to and from a compute node; how long should data be kept on a compute node when it is not being used; what is the impact of moving data in the background on the performance of all jobs; and how does workflow support fit into a charging model for users. In order for workflow support to be transparent to the user (a key requirement for usability), the system software must address these questions.

3. Outline of Required System Software

The system software provides the functionality necessary to fulfil the requirements listed above. From the perspective of providing transparent support for workflows, with not only temporal but also data dependencies, a number of scenarios are supported:

- Applications can request to share data through SCM. This functionality is primarily for sharing data between different components of the same computational workflow, but it could also be used to share a common dataset between a group of users.
- A user can request for data to be loaded into SCM prior to a job starting, or for it to be moved off SCM after a job has completed. This is not dissimilar to current Burst Buffer technology and is not limited to workflows, but it supports the notion that data can be moved in the background, while nodes are performing computations.
- Data access is restricted to the owner of the job or workflow, or to users that are explicitly granted access. Encryption of data is enabled in order to make sure those access restrictions are maintained.
- A user can choose between 1LM and 2LM mode, if they are supported by the SCM hardware. Rebooting a node into a particular mode is achieved through the resource manager, i.e. the user can specify the job environment in the batch submission script.

- The resource manager can allocate nodes based on storage/memory capacity, as well as compute. If data can be temporarily stored on a compute node, the capacity for storage or memory that is available to other jobs will be reduced for the duration.

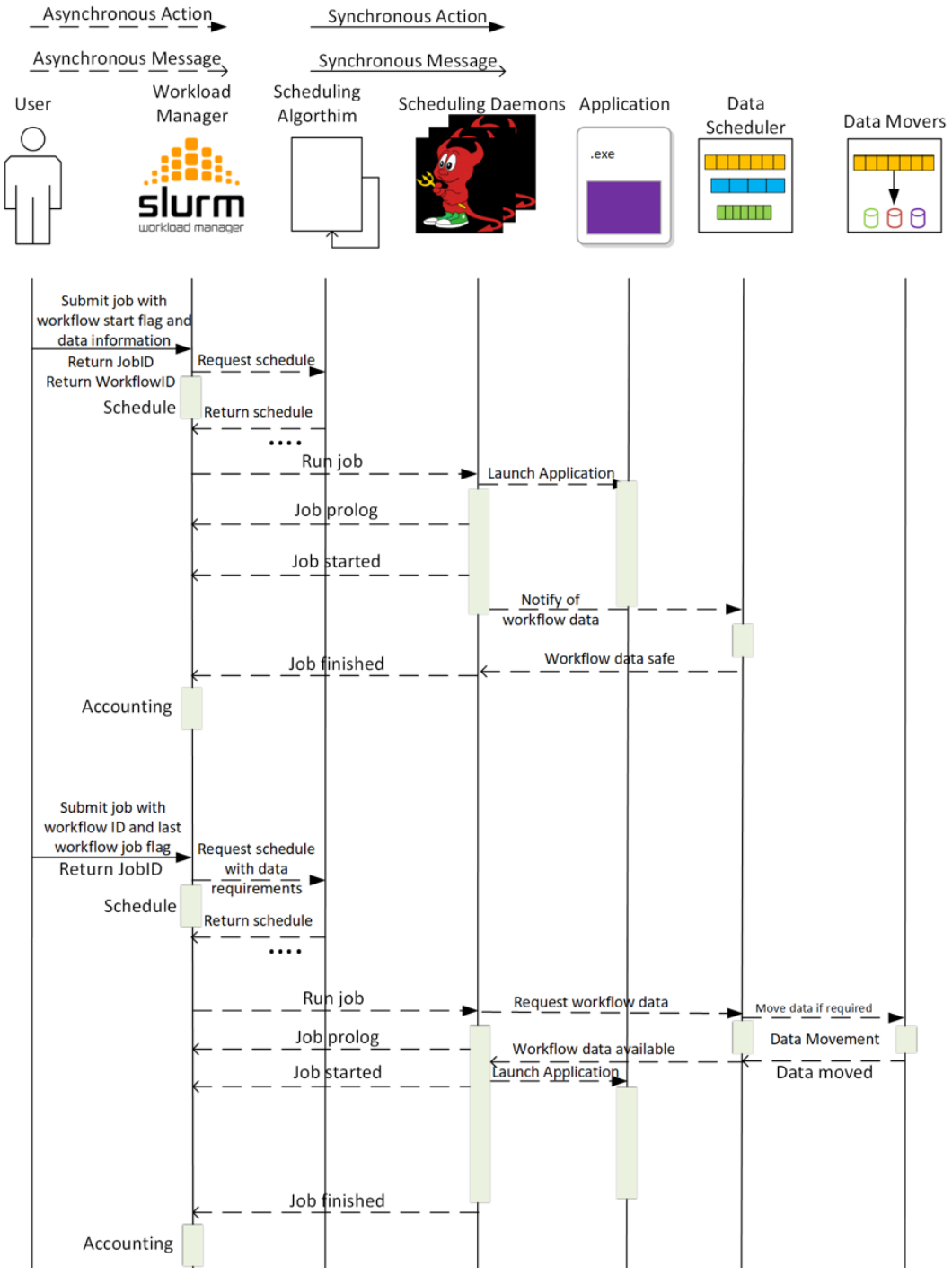

Figure 7. Sequence diagram describing how the workload manager, schedulers and data movers implement a workflow

Figure 7 shows the main components that enable a workflow with data dependencies on a system with SCM: the workload manager, the job and data schedulers, and the data movers.

3.1. Workload Manager and Job Scheduler

As shown earlier, workload managers (the example in Fig. 5 shows SLURM [5]) allow for users to define temporal dependencies between applications. However what is currently not commonly possible on supercomputers is for data dependencies to be defined in the same way. One reason for this is that HPC systems mostly do not have local storage, and it is therefore not possible to keep data close to where the computation takes place: if the data has to be read from or written to an external file system, there are no performance advantages to be gained from understanding the data dependencies.

The workload manager must also be able to query a node's mode of operation, and the amount of free storage (or memory, depending on how the SCM is used) that can be assigned to a job. This latter point is particularly important if data is left on a node for a period of time, e.g. because the next step in a workflow is not ready, and the node's compute capabilities can be used for other work.

The data dependency requirements of a job or workflow are described in the submission script in a similar way in which the compute resource needs would be outlined today. In addition to listing the walltime and number of processes or nodes, the I/O requirements (including the size of the data) are also described. With this additional information, and the workload manager's ability to query the system state, the job scheduler can (if there is sufficient capacity) assign the components of a workflow to be near the data that forms part of this workflow.

The job scheduler is aware of the SCM resources in the system at all times, and it allows users to specify SCM requirements and data movement requirements, to understand the configuration of the compute nodes and to allow users to specify configuration requirements of compute nodes for their jobs. The job scheduler communicates with all job scheduling and data scheduling components of the system software, to enable the system to run in as efficient a manner as possible and ensure user data is secure and safe. The job scheduler consists of multiple scheduling components, including components that can schedule jobs based on data location or energy policies.

3.2. Data Scheduler

The data scheduler operates under the instruction from other software components, be they system software (e.g. the workload manager or job scheduler), or directly through the request of an application. The main responsibility of the data scheduler is to orchestrate the migration of data. This can be between the different hardware levels within a single node, independently from an application, or between different nodes in the system, specifically to and from SCM on other nodes over the high performance interconnect.

The data scheduler also keeps track of data and what hardware that data is located in. On the compute nodes, the data scheduler component provides the local functionality to move data between storage levels (e.g. filesystem, SCM, DRAM) as instructed by the higher level component.

3.3. Data Movers

Data movers are simple software components that are used by the data scheduler to undertake specific data operations, such as copying data between different filesystems, from local to remote compute nodes, and between different data storage targets (for instance between an

object store and a filesystem). Separate data movers implement different operations, allowing targeted optimisations for each operation.

4. Assessing the Performance Optimisation Potential

To evaluate the potential for workflow optimisation that SCM, and an SCM aware job scheduler, can enable we undertook some benchmarking of the I/O costs that a workflow can experience. For these tests we create separate producer and consumer applications, that either generate or read a set of data files. The applications can generate or utilise different numbers or sizes of files to enable a range of different types of workflow interaction to be explored. These applications do not perform any other work, and thus only represent the I/O aspects of the workflow, but they do allow us understand the potential for performance optimisation from SCM functionality over a range of hardware and workflow configurations. We evaluate workflow I/O costs, writing data from the producer and reading data with the consumer, using a single compute node, with three different hardware configurations:

1. External Lustre filesystem;
2. Internal SSD storage device;
3. Memory mapped filesystem.

The external Lustre filesystem is equivalent to many current HPC system configurations. The internal SSD storage device represents compute node local storage, but without the potential read and write performance that true SCM hardware offers. The memory mapped filesystem represents performance that is closer to SCM technology. File-based I/O is generally subject to O/S level caching, keeping recent data in memory for re-use rather than requiring the data to be fetched from disk. Such caching can offer significant performance benefits for recently used file data, but is not representative of workflows where applications could be run on different nodes or at different times, and therefore would not be able to benefit from such I/O caching. Therefore, we ran our single producer-consumer benchmarks both with and without I/O caching to enable the evaluation of the benefit of such functionality and the impact of being unable to utilise it.

The data is collected on a single dual-socket node containing 2 Intel Xeon Platinum 8160F CPUs (24 cores @ 2.10 GHz each) with 192GB of DDR4 memory and a local 800GB Intel SSD DC S3710 Series SSD device. The node is connected to a 750GB Lustre (version 2.9) filesystem. Table 1 presents the average of 5 runs of each benchmark on the different hardware options we have previously outlined without I/O caching, using the following file configurations:

- 10 files, each of 1GB (10 x 1GB);
- 100 files, each of 100MB (100 x 100MB);
- 1,000 files, each of 10MB (1,000 x 10MB).

The results presented are using a single producer and a single consumer application on the node. It is evident from Tab. 1 that significant savings can be made to the workflow overheads associated with transferring data between workflow components. Simple writing to a local storage device (SSD) rather than the external filesystem reduces the I/O cost by up to five times. Moving from writing to a traditional storage device to writing data to memory brings even larger benefits, with the best performance around twelve times faster than writing to the local disk, and around fifty eight times faster than writing to the external filesystem. Whilst SCM is unlikely to achieve performance as good as memory mapped filesystem hosted on DRAM, these

Table 1. Performance of workflow benchmark *without* I/O caching (single producer-consumer), using three different file configurations. The performance is reported as time in seconds. Times in brackets are (write/read) times for the (producer/consumer)

Hardware	10 x 1GB	100 x 100MB	1000 x 10MB
Lustre	291.41 (137.77/153.64)	216.68 (105.34/111.34)	196.80 (102.42/94.38)
SSD	61.26 (34.39/26.87)	54.53 (29.05/25.48)	54.44 (28.09/26.35)
Memory	4.97 (3.02/1.95)	4.47 (2.99/1.48)	4.70 (3.16/1.54)

Table 2. Performance of workflow benchmark *with* I/O caching (single producer-consumer), using three different file configurations. The performance is reported as time in seconds. Times in brackets are the (write/read) times for the (producer/consumer)

Hardware	10 x 1GB	100 x 100MB	1000 x 10MB
Lustre	144.36 (138.94/5.38)	104.26 (102.09/2.18)	103.90 (102.24/1.64)
SSD	36.12 (34.37/2.04)	30.73 (29.11/1.62)	30.08 (28.28/1.80)
Memory	4.62 (3.00/1.62)	4.66 (3.04/1.62)	4.91 (3.25/1.66)

results indicate the performance differences between storage devices, such as fast SSDs, and true memory technologies.

Table 2 presents the average of 5 runs of the same benchmarks, but this time with I/O caching enabled, highlighting the performance optimisation potential enabled by a SCM and data aware job scheduler. It is evident from the results in the table that both the external filesystem and internal storage device benefit significantly from I/O caching enabled at the operating system level. The retention of data within the compute node can improve the performance even with fast I/O devices. We note that I/O caching does not significantly impact the memory mapped filesystem as that approach is already keeping data in memory rather than requiring I/O to access the underlying persistent storage device. As well as evaluating a single producer-consumer combination, we also evaluated the performance impact of workflow optimisation with SCM using multiple producer-consumers on a single node. We benchmarked using 36 producers and 36 consumers, running the same tests as before, albeit with smaller numbers of files to enable the benchmarks to finish in a reasonable time, and the data to be resident in memory.

Table 3 presents the results of the no-caching test, with the main number as the maximum runtime (maximum write time plus maximum read time) for the workflow, and the number in brackets as the minimum runtime (minimum write time plus minimum read time). It is evident from the table that the performance impact of multiple processes undertaking I/O at the same, or similar, times is significantly larger with the external filesystem than that with the local device or writing data to memory.

The performance differential between the Lustre and SSD benchmarks is larger for the multiple process tests ($\approx 5.7 - 6.4\times$) than for the single process tests ($\approx 3.6 - 4.6\times$). The Memory benchmark performance is in fact significantly faster than the single producer-consumer bench-

Table 3. Performance of workflow benchmark without I/O caching (36 producer-consumers). Performance is reported as maximum time in seconds (minimum time in brackets)

Hardware	1 x 1GB	10 x 100MB	100 x 10MB
Lustre	978.25 (790.15)	911.58 (591.07)	906.87 (829.00)
SSD	152.00 (142.99)	155.63 (138.14)	158.12 (150.78)
Memory	0.84 (0.77)	1.06 (0.79)	1.03 (0.66)

Table 4. Characteristics of the jobs used in the illustration of the optimisation potential

Job ID	Number of nodes	$T_{I/O}$ (s)	$T_{compute}$ (s)	T_{run} (s)	Part of Workflow?
1	4	10 + 30	50	90	y
2	1	10 + 0	180	190	n
3	15	10 + 0	30	40	n
4	1	20 + 10	400	430	n
5	7	10 + 10	70	90	n
6	15	10 + 10	20	40	n
7	4	30 + 120	180	330	y
8	8	0 + 60	250	310	n
9	4	10 + 30	10	50	y
10	2	20 + 30	150	200	n

marks, however we believe this is because I/O for the small data sizes used in these benchmarks can exploit cache memory more efficiently. Furthermore, the performance variability of node local I/O (SSD) is also significantly smaller ($\approx 5\% - 12\%$) than for the external filesystem ($\approx 9\% - 54\%$), demonstrating another potential benefit of SCM (reduced performance variability).

4.1. Illustration of Optimisation Potential

In this section, we give a simple example of how data aware workflow scheduling using SCM can improve both the performance of the workflow itself, and improve the resource usage on the system. We assume a system with 20 nodes and schedule 10 very short jobs onto these nodes. The characteristics of the jobs are described in Tab. 4; for illustration purposes, each job is broken down into three distinct phases (input, compute and output), and may or may not be part of a workflow. Jobs that are not part of a workflow are scheduled onto the system by incrementing ID in a round-robin fashion, if there is sufficient space to accommodate them. Jobs that are part of a workflow required the preceding components of the workflow to be completed before they can be executed; in our example, Jobs 1, 7 and 9 form a workflow with data dependencies. It is assumed that no data is stored locally by default.

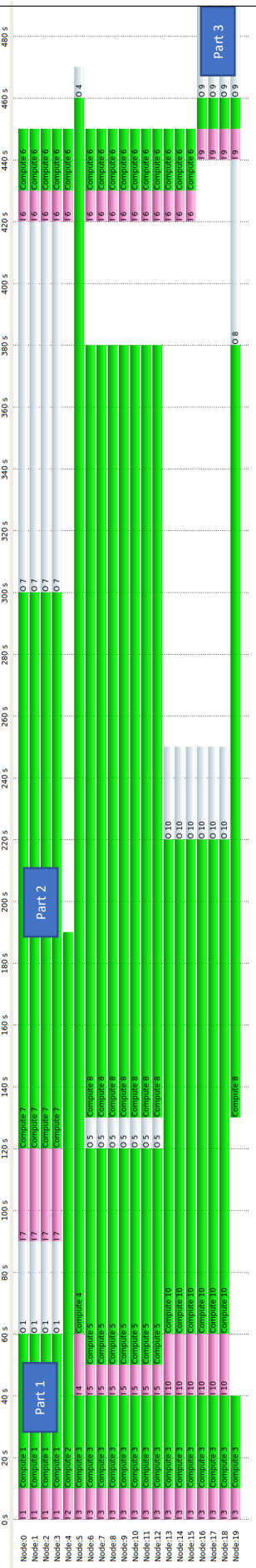

Figure 8. Timeline of example jobs being scheduled, without awareness of data dependencies within a workflow. Time to solution for all three parts (Jobs 1, 7 and 9) of the workflow: 490s

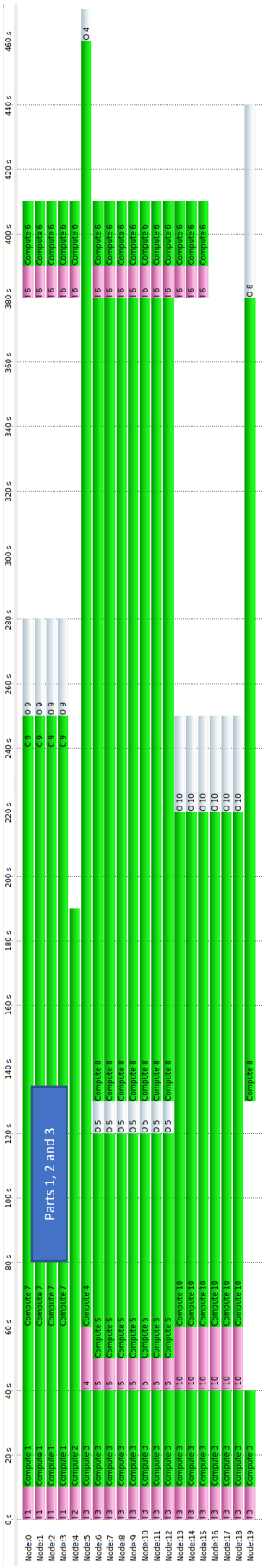

Figure 9. Timeline of example jobs being scheduled, with awareness of data dependencies within a workflow and avoiding writing to/reading from the external file system. Time to solution for all three parts (Jobs 1, 7 and 9) of the workflow: 280s

Figure 8 shows a timeline of the allocation of the example jobs using a standard scheduler that is not aware of data dependencies inside a workflow. As can be seen from the timeline, parts 1 and 2 of the workflow happen to be scheduled onto the same nodes by chance (i.e. node 0–3). However, the 3rd component of the workflow is scheduled on nodes 16–19, because another job has taken over the resources used by the first two parts of the workflow. Despite parts 1 and 2 of the workflow being scheduled on the same resources, Job 1 has to write its output to the external file system. This data represents the input for Job 7, which in turn has to reload the data from the external file system. The total individual runtimes for the workflow components T_{run} are 90s, 330s and 50s, a total of 470s. However, because Job 9 is held in the queue for a short amount of time (in this example 20s), the total time to solution T_{all} increases to 490s.

Figure 9 illustrates how the allocation changes if the scheduler understands data dependencies and the importance of data locality in workflows. Two separate optimisations occur: firstly, the 3 parts of the workflow are now scheduled to use the same nodes 0-3; secondly, and as a direct result of the first step, data can now be kept locally on the compute nodes. There is no need to access the external file system. I/O cost will be close to DRAM speed and thus vastly reduced even when compared to SSD. In our example here, we assume (for simplicity) that I/O cost tends to 0 when using SCM. The first job in the workflow still needs to read its input from external storage, and the final job needs to write back to external storage, but all other I/O can be local. This results in the following runtimes per job:

$$\begin{aligned} T_{run,Job1} &= 10s + 50s = 60s, \\ T_{run,Job7} &= 0s + 180s = 180s, \\ T_{run,Job9} &= 30s + 10s = 40s. \end{aligned} \tag{6}$$

As the jobs are scheduled consecutively onto the same nodes, there is no additional queuing time, and T_{all} is simply the sum of the individual runtimes, i.e. 280s.

As a further optimisation, which we do not consider in this example, it is possible to pre-load and post-move the input/output data of particularly data-intensive jobs in the background, prior to them starting execution, or after they have completed. This increases the resource allocation constraints that the scheduler has to work around, however, the potential gains in time to solution and resource utilisation are significant.

5. Related Work

Recent years have seen a lot of research emerge around the topics of I/O performance (notably with the arrival of new storage technologies), scheduling of large-scale systems and scientific workflows. Daley et al. [1] assess how Burst Buffers can alleviate the I/O bottleneck of some scientific workflows, and acknowledge the associated data management challenges. Also primarily focussed on Burst Buffers, Herbein et al. [4] discuss a technique for making scheduling policies I/O aware, taking into account the different bandwidths of the storage hierarchy in order to avoid I/O contention. Rodrigo et al. [7] address the idea of workflow-aware scheduling, having recognised that workloads on HPC system are a more commonly comprised of an interdependent series of jobs. They propose a workflow-aware extension to the the widely used SLURM scheduler, which goes beyond the simple temporal dependencies between jobs that are supported in most resource managers.

Conclusions

In this paper, we outline the opportunities for performance improvements that byte-addressable storage class memory, such as the upcoming 3D XPoint^{TM} NVDIMMs can bring in particular to data intensive applications. We also present the system software that needs to be put in place in order to support data aware workflow scheduling using persistent memory. Overall, the benefits are clear: if a workflow can be scheduled so that its time to solution is decreased, but without impinging on other jobs, the system resources are freed up sooner and the total workload throughput for the HPC system is improved.

Acknowledgements

The NEXTGenIO project has received funding from the European Union's H2020 Research and Innovation Programme under Grant Agreement no 671951. The project works on addressing the I/O challenge, a key bottleneck as the HPC community is moving towards the Exascale. NEXTGenIO is an internal collaboration between research organisations and industry to develop a prototype hardware platform that uses on-node SCM to bridge the latency gap between memory and storage. The project is also developing the necessary system software to enable the transparent use of SCM, with the aim to improve application performance, as well as system utilisation and workload throughput.

References

1. Daley, C., Ghoshal, D., Lockwood, G., Dosanjh, S., Ramakrishnan, L., Wright, N.: Performance Characterization of Scientific Workflows for the Optimal Use of Burst Buffers. Future Generation Computer Systems (2017), DOI: 10.1016/j.future.2017.12.022

2. Deelman, E., Peterka, T., Altintas, I., Carothers, C.D., van Dam, K.K., Moreland, K., Parashar, M., Ramakrishnan, L., Taufer, M., Vetter, J.: The Future of Scientific Workflows. The International Journal of High Performance Computing Applications 32(1), 159–175 (2018), DOI: 10.1177/1094342017704893

3. Hady, F.T., Foong, A., Veal, B., Williams, D.: Platform Storage Performance with 3D XPoint Technology. Proceedings of the IEEE 105(9), 1822–1833 (Sept 2017), DOI: 10.1109/JPROC.2017.2731776

4. Herbein, S., Ahn, D.H., Lipari, D., Scogland, T.R., Stearman, M., Grondona, M., Garlick, J., Springmeyer, B., Taufer, M.: Scalable I/O-Aware Job Scheduling for Burst Buffer Enabled HPC Clusters. In: Proceedings of the 25th ACM International Symposium on High-Performance Parallel and Distributed Computing. pp. 69–80. HPDC '16, ACM, New York, NY, USA (2016), DOI: 10.1145/2907294.2907316

5. Jette, M.A., Yoo, A.B., Grondona, M.: SLURM: Simple Linux Utility for Resource Management. In: In Lecture Notes in Computer Science: Proceedings of Job Scheduling Strategies for Parallel Processing (JSSPP) 2003. pp. 44–60. Springer-Verlag (2002)

6. Joydeep, R., Varghese, G., Inder M., S., Jeffrey R., W.: Intel Patent on Multi-Level Memory Configuration for Non-Volatile Memory Technology. https://www.google.com/patents/US20150178204 (2013), accessed: 2018-04-11

7. Rodrigo, G.P., Elmroth, E., Östberg, P.O., Ramakrishnan, L.: Enabling Workflow-Aware Scheduling on HPC Systems. In: Proceedings of the 26th International Symposium on High-Performance Parallel and Distributed Computing. pp. 3–14. HPDC '17, ACM, New York, NY, USA (2017), DOI: 10.1145/3078597.3078604

8. Rudoff, A.: Persistent Memory Programming. http://pmem.io (2017), accessed: 2018-04-11

9. Rudoff, A.: Persistent Memory: The Value to HPC and the Challenges. In: Proceedings of the Workshop on Memory Centric Programming for HPC. pp. 7–10. MCHPC'17, ACM, New York, NY, USA (2017), DOI: 10.1145/3145617.3158213

10. Sunny, G.: Getting Ready for Intel© Xeon Phi™ Processor Product Family. https://software.intel.com/en-us/articles/getting-ready-for-KNL (2017), accessed: 2018-04-11