Extending Composable Data Services into SmartNICs

Craig Ulmer
Scalable Modeling & Analysis
Sandia National Laboratories
Livermore, CA, USA
cdulmer@sandia.gov

Jianshen Liu
Comp. Science & Engineering
UC Santa Cruz
Santa Cruz, CA, USA
jliu120@ucsc.edu

Carlos Maltzahn
Comp. Science & Engineering
UC Santa Cruz
Santa Cruz, CA, USA
carlosm@ucsc.edu

Matthew L. Curry
Scalable System Software
Sandia National Laboratories
Albuquerque, NM, USA
mlcurry@sandia.gov

***Abstract*—Advanced scientific-computing workflows rely on composable data services to migrate data between simulation and analysis jobs that run in parallel on high-performance computing (HPC) platforms. Unfortunately, these services consume compute-node memory and processing resources that could otherwise be used to complete the workflow's tasks. The emergence of programmable network interface cards, or SmartNICs, presents an opportunity to host data services in an isolated space within a compute node that does not impact host resources. In this paper we explore extending data services into SmartNICs and describe a software stack for services that uses Faodel and Apache Arrow. To illustrate how this stack operates, we present a case study that implements a distributed, particle-sifting service for reorganizing simulation results. Performance experiments from a 100-node cluster equipped with 100Gb/s BlueField-2 SmartNICs indicate that current SmartNICs can perform useful data management tasks, albeit at a lower throughput than hosts.**

***Index Terms*—SmartNICs, HPC data services, BlueField-2**

I. Introduction

Scientific computing users leverage modeling, simulation, and analysis (or ModSim) tools on high-performance computing (HPC) platforms to answer research questions that would otherwise be difficult or impossible to resolve through physical experiments alone. ModSim workflows are typically composed of multiple processing steps that are executed by different tools. For example, a scientific workflow may use mesh generation tools to create input datasets, parallel simulation tools to evaluate different real-world scenarios, visualization tools to analyze the results, and I/O staging tools to aggregate, index, and archive output data. The HPC community has invested significant effort in developing and parallelizing each of these tools to ensure workflows can scale to the tens of thousands of compute nodes found in modern HPC platforms.

Systems researchers have constructed *composable data service* libraries [1] for HPC platforms to make it easier for developers to customize how data flows between different tools in a workflow. These libraries include multiple software components that can be combined to build application-specific services. Components include low-level RDMA software for moving data between compute nodes, key/value data stores for organizing information in memory, and asynchronous compute engines for processing in-transit data.

While composable data service libraries are important in workflows, a criticism of current work is that services run in the system's compute nodes and consume resources that would otherwise be available to ModSim tools. This paper explores a new architecture option: hosting services in programmable network interface cards (or SmartNICs). While slower than servers, SmartNICs offer resource isolation and locality benefits that are attractive for many data services. The immediate research questions are: *How should we construct software to implement services on these devices? Can distributed services perform useful work on SmartNICs?* To answer these questions, we describe current composable data service libraries and define requirements for enabling interoperability with SmartNICs. We discuss a software-stack prototype and evaluate its performance on current hardware. Finally, we present a case study where a distributed, particle-sifting service runs on a 100-node HPC cluster that features BlueField-2 SmartNICs.

II. Composable Data Services for HPC Platforms

A. *Scientific Computing Workflows*

Scientific computing workflows may involve multiple, parallel tools that run on different nodes in an HPC platform at the same time. For example, the workflow depicted in Fig. 1 first uses a low-fidelity simulation to generate coarse-grained results that deep-learning tools can use to make predictions about the simulation's general behavior. These predictions are then used during a high-fidelity simulation to make better decisions about optimizations such as load balancing. Output results from the high-fidelity simulation are then routed through visualization and I/O staging tools to extract insight and reorganize data before it is archived to disk.

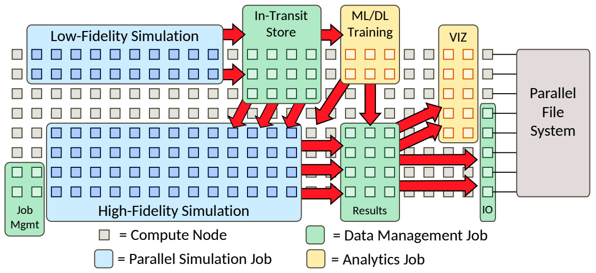

Fig. 1: A workflow is mapped to HPC compute resources

The traditional means of passing data between workflow tasks has been to write intermediate results to disk [2]. While NVMe storage has dramatically improved performance [3], I/O is still a significant impediment in workflows as data must be transformed from an in-application representation to an archival, on-disk format. Additionally, file I/O libraries can be inconvenient for developers as the interfaces are primarily designed to read and write data rather than process it.

B. Composable Data Service Libraries

As a means of improving how data flows between workflow tools, research groups have constructed *composable data service* libraries for HPC platforms, including DataSpaces [4], Mochi [1], and Faodel [5]. These libraries provide flexible communication software that makes it easier to route data from one application's memory space to another's without using the file system. An important aspect of this work is that users are presented with higher-level primitives than are normally found in communication libraries. In addition to low-level RPC and RDMA facilities, composable data service libraries include key/value stores, REST API engines, and I/O drivers for interacting with external data repositories. These features simplify development and enable users to reason about their data at higher levels of abstraction.

Faodel provides an example of a composable data service library that supports multiple HPC platform architectures. Faodel is open-source[1] C++ software that includes drivers for InfiniBand [6], RoCE [7], and Cray Aries [8] network fabrics. Faodel is composed of several components: an RDMA portability library (NNTI) for low-level communication; a state-machine engine (OpBox) for managing asynchronous tasks; a memory-management library (Lunasa) for tracking memory allocations for network-accessible objects; a directory service (DirMan) for maintaining workflow configuration information; a key/blob service (Kelpie) for safely transferring objects between servers; and a lightweight web server (Whookie) to allow users to query a remote service. In prior work we have used Faodel for I/O staging and checkpointing [9], coupling visualization applications to simulation codes, and insulating users from platform-specific storage issues [10].

C. Data Processing Library Extensions

Researchers in the HPC and data science communities have independently constructed advanced, data processing libraries that greatly complement the functionality of composable data service libraries. These libraries define robust data structures for organizing information and are designed to exploit the parallel-processing capabilities of modern CPUs and GPUs. Popular data processing libraries in this space include VTK-m [11], Kokkos [12], and Apache Arrow [13].

Apache Arrow is an open-source[2] project centered around an in-memory format specification and serialization protocol for column-based table data. It is SIMD [14] and vectorization friendly and relocatable, enabling zero-copy access in shared memory. The project includes a number of libraries for efficiently processing this data that are implemented in multiple languages (including C++) running on multiple platforms. In C++ an Arrow table is a two-dimensional data structure with chunked arrays for columns and a schema. Tables can be processed without copying using reference-counted record batches that hold contiguous portions of the data. Record batches enable work to be spread across multiple processors. Moreover, because of the contiguous property within a record batch, data processing can further take advantage of data-level parallelism using SIMD instructions that are generally available on modern x86 and Arm processors. Apache Arrow has been adopted by many research and commercial projects such as Apache Spark [15], Dask [16], and Polars.

D. Service Placement

There are currently three locations in HPC platforms where researchers typically host data management services: in situ, in vitro, and in storage. In-situ approaches embed services inside the individual tools of a workflow. This approach reduces the overhead of interacting with a service, but increases build complexity, sacrifices application resources to the service, and causes faults in either side to affect both. In-vitro approaches host services in external nodes within the platform. This approach provides isolation but adds extra communication overhead and increases the overall node count for a workflow. Finally, in-storage approaches such as Skyhook [17] embed data services within the platform's storage nodes. While storage nodes are an ideal location for these services, system polices may forbid users from executing code in these servers for security and reliability reasons.

III. Smart Network Interface Cards

In recent years multiple hardware vendors have introduced network and storage devices that feature user-programmable CPUs or FPGAs. These resources enable developers to "push down" application-specific functionality to remote hardware to help customize queries and reduce the amount of data returned. Multiple network companies have created powerful SmartNICs that can inspect and process network data as it moves between the host and the network. Current generation SmartNICs feature multiple processor cores, sizable amounts of volatile and nonvolatile memory, and direct access to high-speed communication networks. As such, SmartNICs present a new opportunity for hosting data services in HPC platforms.

A. NVIDIA BlueField-2 SmartNIC

NVIDIA has developed multiple SmartNIC products to serve the security needs of cloud vendors. As illustrated in Fig. 2, NVIDIA's current-generation BlueField-2 SmartNIC supplements a traditional network adapter with eight Arm A72 cores at 2.75GHz, 16GB of DRAM, 60GB of eMMC storage, and two 100Gb/s network ports that can interact with InfiniBand or Ethernet. Special-purpose hardware accelerators are available for offloading encryption, compression, and regular expression operations. At power on the BlueField-2 loads its own OS (e.g., Ubuntu 20.04) from eMMC storage. This OS operates independently from the host and is visible through drivers that provide network and console access to the card.

Multiple researchers have explored leveraging BlueField SmartNICs for different tasks. In iPipe [18] researchers created an actor-based framework to allow computations to

[1]https://github.com/faodel

[2]https://github.com/apache/arrow

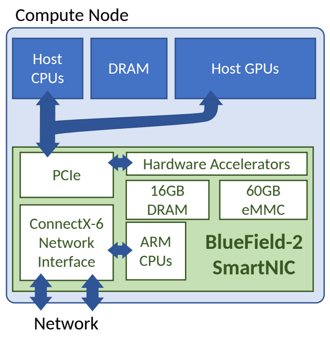

Fig. 2: Compute node with a BlueField-2 SmartNIC

be offloaded to SmartNICs. In prior work we benchmarked the BlueField-2 [19] and demonstrated that its compression hardware can provide significant performance improvements when handling compressed, Apache Arrow data streams [20]. In “Smarter” NICs [21] portions of a molecular dynamics applications were offloaded to SmartNICs to improve overall performance.

B. *Hosting Data Services in SmartNICs*

Following the release of the InfiniBand-based BlueField-2 adapter, multiple institutions have deployed HPC platforms that feature SmartNIC-enabled compute nodes [21]–[24]. These architecture offer an opportunity to migrate data services into SmartNICs. We see multiple advantages in this approach. First, hosting services in SmartNICs enables services to be placed near applications in an isolated space that does not consume host resources. As such, the host can offload low-priority or asynchronous tasks that might otherwise impede applications. Second, SmartNIC-enabled compute nodes add compute and memory resources to the platform without requiring additional network infrastructure. Finally, vendor roadmaps indicate that future generations of SmartNICs will include processor and accelerator enhancements. While current SmartNIC hardware is sufficient for basic data management tasks, upcoming products may take on greater responsibilities in processing data pipelines.

IV. SmartNIC Software Stack for Data Services

To efficiently bridge the gap between resource-rich hosts and resource-constrained SmartNICs running data management services, we need a software stack that can orchestrate these services among hosts and SmartNICs while minimizing impact on applications and maximizing reuse of existing software. This software stack must address communication issues (e.g., How do applications interact with remote SmartNICs over the network? How can SmartNICs work collectively?) as well as computational issues (e.g., How are computations defined and executed by services? How can the system be extended with new operations?). In this section we define our list of requirements for this software stack and discuss how a suitable environment for hosting services in SmartNICs can be constructed through the combination of the Faodel and Apache Arrow libraries.

A. *Service Requirements*

Based on our experiences with workflow environments, we identify five basic requirements we expect from an environment where services execute in embedded devices. (1) Each service endpoint requires a unique identity that other entities in the platform can reference and access via efficient communication mechanisms. (2) Users must be able to control the mapping of services to physical resources at runtime and group several devices together in a way that allows the devices to work together. (3) Users must be able to trigger service computations locally and remotely. (4) The stack should present a flexible data-processing API that is robust and has community acceptance. (5) Data-parallel computations must automatically exploit available CPU resources.

B. *Communication: Faodel*

We selected the Faodel library to serve as a foundation for the communication portion of our software stack prototype because it is open source, written in C++, has support for both x86 and Arm, and includes existing primitives for working with endpoints scattered about a platform. Specific details about how Faodel fulfills our requirements follow.

- **System-wide Accessibility:** Faodel assigns a unique identifier to each endpoint that is used to establish both HTTP and RDMA communication. Faodel’s Kelpie library provides an easy-to-use mechanism for safely transferring key-labeled objects between endpoints using RDMA mechanisms. Users can put, get, list, and delete objects on local or remote endpoints.
- **Resource Pools:** Kelpie uses a simple pool abstraction for grouping multiple endpoints together for related work. A pool contains a list of endpoint members and a distribution policy that maps key labels to pool members. By supplying different pool configurations at start time, users can change the behavior of their data flows.
- **Dispatching Computations:** While Kelpie is agnostic about data formats and computations, it provides two methods for invoking computations at endpoints. First, an endpoint may run its own main loop that periodically inspects state and reacts to changes. Second, users may invoke computations on objects at remote endpoints through user-defined functions.

C. *Computation: Apache Arrow*

Apache Arrow was selected to implement the data computations in this work because it provides a rich set of primitives for storing and querying tabular data, is open-source C++, and is actively developed by a large community. Specific aspects of Arrow that meet our requirements follow.

- **Common Data Representation**: Arrow’s tabular data model is suitable for describing many kinds of scientific datasets and provides a useful standard for data exchange. In addition to efficient, in-memory data structures for storing and processing tabular data, Arrow includes serialization software for converting data to a standard, onwire format. This software simplifies development and improves interoperability with other libraries.

- **Data-Parallel Computations:** One of the benefits of Arrow's robust, tabular data model is that users can specify high-level queries that can be processed efficiently with parallel-processing techniques. Specifically, Arrow includes a streaming data processing engine named Acero [25] that processes complex user queries on tables. Acero extracts a computational graph from a query and then maps the data flow to local processing cores.

D. *Integration Challenges*

We faced two integration concerns while constructing our software stack for data management services. First, small portions of Faodel and Arrow target processor-specific features. While both libraries had previously been ported to x86 and Arm processors, extensive testing was required to ensure data handoffs between the two architectures functioned correctly. The second integration challenge involved finding a means of transporting Arrow data using Faodel's native objects. Our current solution is to use Arrow's IPC serialization mechanisms to embed one or more tables in a Faodel object. A wrapper class was developed to convert between an in-memory Arrow table and the payload section of a Faodel object.

V. BlueField-2 Performance Experiments

We conducted multiple experiments to observe the low-level performance characteristics of the BlueField-2's embedded processors when executing different operations from our software stack. In this section we present measurements for both the Faodel and Arrow portions of the stack.

A. *Bookkeeping Overhead on the SmartNIC*

Faodel provides a stress-test tool for measuring how quickly a system can perform different tasks. Similar to stress-ng [26], performance numbers lack meaning in isolation, but provide a useful way to compare different architectures. Faodel's LocalKV test uses a workload that employs multiple threads to put, get, and delete objects from a local, in-memory, 2D hash map. Key names are intentionally picked to either seek or avoid collisions. This test exercises common data processing tasks, such as hashing, reference counting, lock handling, and managing memory allocations.

We executed the LocalKV test on a diverse set of platforms to observe how the BlueField-2's processors performed compared to other architectures. The processors included: a 32-core AMD EPYC 7543P (Zen3) processor, a 68-core Knights Landing (KNL) processor, and BlueField-1 and BlueField-2 SmartNICs with 16 and 8 Arm cores respectively. As depicted in Fig. 3, aggregate performance (decreases/increases) as thread counts increase in the collision (seeking/avoiding) experiments. Current server processors are roughly four times faster when using the same number of threads, and an order of magnitude faster when using all cores. Interestingly, the BlueField-2 outperforms the data-parallel KNL processors, which were employed in the previous generation of HPC platforms and had known performance limitations [27].

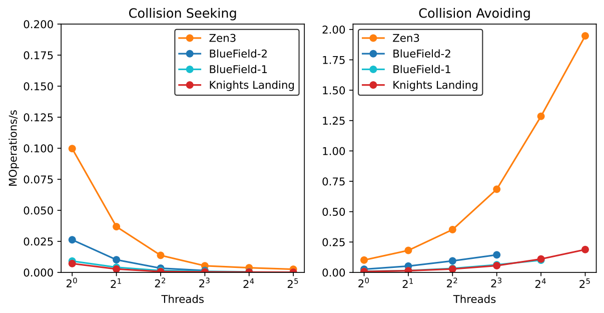

Fig. 3: Performance in Faodel's LocalKV stress test

B. *Processing Arrow Data*

Our Arrow experiments with the BlueField-2 focused on creating queries with inherit parallelism and verifying that execution performance improves as the number of threads increases. For this work we selected two types of queries that operate on three-dimensional particle data. The first query filters an input dataset based on a bounding box that is picked to select 1/8th of the original particles. The second query computes the squared magnitude of the velocity of each particle and returns the minimum and maximum values. We created a particle dataset with 8M records and then measured the amount of time required to complete the queries using a variable number of threads on the BlueField-2 and a host system with a total of 32 Xeon E5-2698 processor cores.

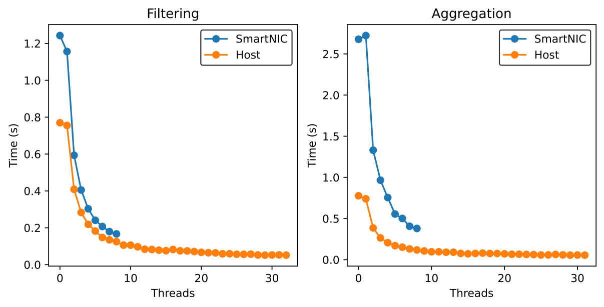

Fig. 4: Apache Arrow threading performance

The performance results presented in Fig. 4 confirm that Apache Arrow can parallelize queries and leverage multiple processor cores to improve performance. Latency drops significantly when moving from 2 to 3 threads for both systems. However, there are only minimal improvements beyond 5 threads. While the host is much more powerful than the embedded processor on the SmartNIC, it is only 37.8% faster than the SmartNIC when using 8 cores for this workload.

VI. Particle-Sifting Example

As a means of demonstrating how SmartNICs can be leveraged in HPC workflows, we constructed a particle-sifting example that uses distributed SmartNICs to organize output data from a simulation into a form that is easier for analysis applications to consume. This example uses the SmartNICs in the simulation's compute nodes to sort, partition, and redistribute the data in one or more passes. Sifting behavior can be controlled by passing in different configuration information at start time. We report performance measurements from a 100-node cluster with 100Gb/s InfiniBand BlueField-2 SmartNICs to explore the tradeoffs for different runtime configurations.

A. *Reorganizing Particle Data*

Scientists employ Particle-in-Cell (PIC) methods [28] in simulation codes to model a wide variety of phenomena [29], [30]. PIC codes track the discrete state of billions of particles as they move about and interact with a model of a physical environment. The sheer size of the particle data precludes users from writing continuous snapshots to disk or storing more than a single time step of data in the simulation's memory. The ability to rapidly sample and export sizable portions of this data would provide an opportunity for users to apply external analytics in a workflow to inspect how the state of individual particles evolves over time. One of the obstacles in exploiting this data, however, is reorganizing it from the simulation's perspective (i.e., *temporal snapshots*, where data is sorted by time step and simulation rank) to a form that analysis applications can leverage (i.e., *particle tracks*, where data is sorted by particle ID and time step). It is therefore useful for a workflow to provide data management services that can transform the data from one representation to another.

We define multiple requirements for building a particle-sifting service. First, the service must be implemented in a distributed manner that spreads the data and work across available resources to ensure efficient execution and memory utilization. Second, processing elements (PEs) must be able to accumulate data and operate asynchronously to allow the system to react to dynamic runtime characteristics. Finally, the service must minimize the amount of time required for a simulation to inject a new wave of data.

B. *Distributed Sifting Implementation*

We constructed software on top of Faodel and Arrow to implement a multistage sifting algorithm that uses a collection of SmartNICs (or Hosts) as PEs in a linear pipeline. As illustrated in Fig. 5, simulation ranks sample particle data for the current time step and inject a copy of it to the PE hosted at the local SmartNIC. Once a user-defined accumulation threshold is crossed, the PE performs a *compaction* operation. During compaction, the PE splits all of its accumulated data into smaller objects based on bits in each record's particle ID field [20], and transmits each output object to its corresponding PE in the next stage of processing. Particles become more sorted as they move through each of the stages.

While PEs can be mapped to any physical SmartNIC or host in the system, it is expected that multiple, neighboring PEs will exist at a single location to reduce communication costs. The actual steering of data between PEs is managed through a combination of a key-labeling scheme and the use of Faodel pools to determine where data is routed. The key-labeling scheme concatenates the next stage's ID and the currently-matched Particle ID bits to pick a unique destination for the data. Additional source info is embedded in a separate portion of the key to avoid collisions with the data from other PEs.

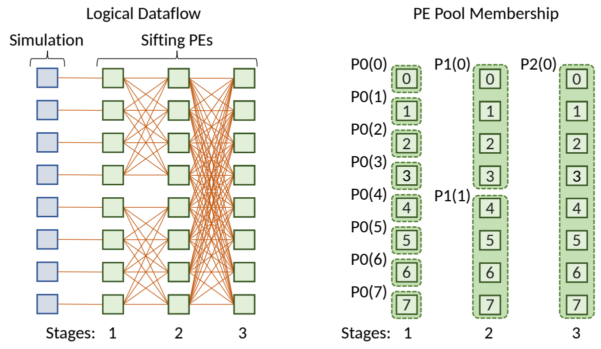

Fig. 5: Dataflow and placement for sifting particle data

Multistage sifting systems with low PE fanout and high numbers of compute nodes can easily result in a few nodes in the system becoming overwhelmed with all the simulation's data. To mitigate this problem, we use Faodel's pool notation to limit the number of nodes to which a PE can distribute data. At start time, software generates a collection of pools in the cluster that correspond to where different PEs reside. For example, the network depicted in Fig. 5 shows three stages and PEs that can split each object into four possible outputs. The 6th PE in stage 1 uses pool "P1(1)" to route to four possible destinations, while the 6th PE in stage 2 uses "P2(0)" to route to eight possible destinations.

C. *Injection Overhead*

The first step in reorganizing the particle data is for each host in the simulation to sample its current data, convert it to serialized Arrow data, and then transfer it to the local SmartNIC. We constructed a benchmark to quantify injection overheads and varied the transfer size from 1M–64M particles (37MB–2.4GB). As presented in Fig. 6, transferring the data to the card through Faodel's primitives consumed 81% of the overall injection time. For 64M particles, we observed an overall transfer rate of 1.32GB/s.

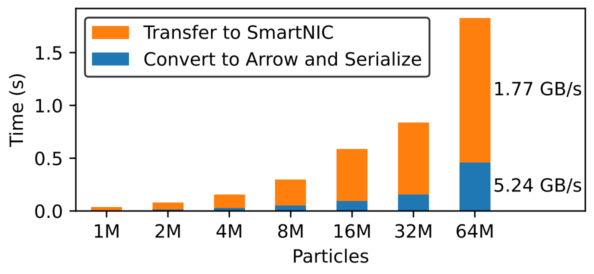

Fig. 6: Data preparation and injection overhead

D. *Impulse Response*

As a means of exploring sifting performance for different configurations, we constructed an impulse response benchmark that injects uniform data to each of stage 1's PEs and then measures the amount of time required for all compaction events to take place in a synchronous manner. We varied the number of splits performed by each PE and selected the minimum number of stages that would be required to fully distribute data across 100 SmartNICs.

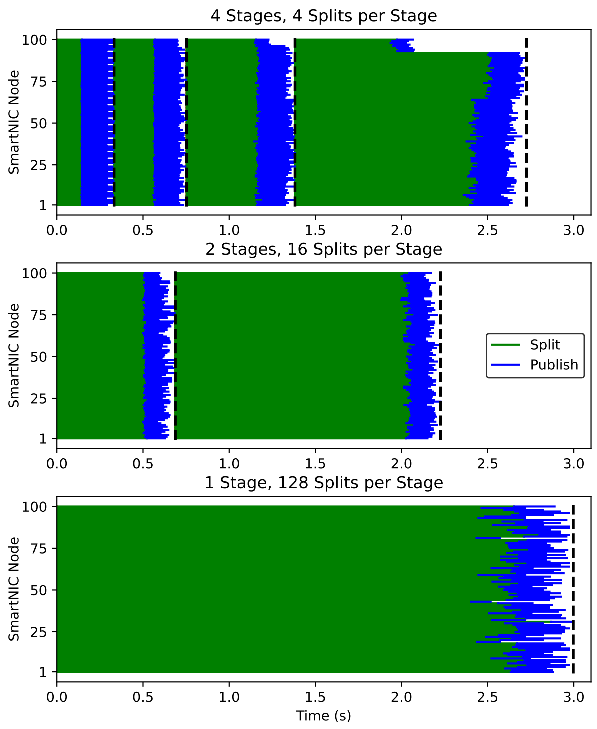

Fig. 7: SmartNIC sifting time for 100M particles

Fig. 7 presents the split and publish timings required to process 100M particles on 100 SmartNICs. While performing 128 splits allows the work to be completed in a single pass, doing so is slightly slower than doing 4-way splits over 4 stages of work. Our experiments indicate that 16 splits per object yielded the best solution for the SmartNICs. In most cases, split time was more expensive than the publish time. Overall, the current implementation provides a relatively uniform distribution of work and data across the nodes.

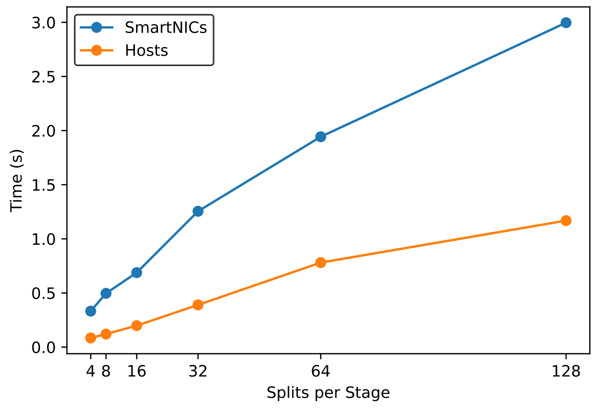

Fig. 8: First stage overhead for 100M particles

Reducing first-stage overhead is important as it makes the sifting network more responsive to injected data. We repeated the previous experiment on 100 EPYC 7543P Zen3 server nodes to measure the first-stage performance for a range of splits. As depicted in Fig. 8, the 32-core host processors were roughly four times faster than the 8-core Arm processors.

In the final set of measurements, we conducted impulse response tests for 10M, 100M, and 1,000M particles. The overall sifting times for 100 SmartNICs and 100 host systems are presented in Fig. 9. Performance scaled linearly in both cases. The host systems were again roughly four times faster than the SmartNICs.

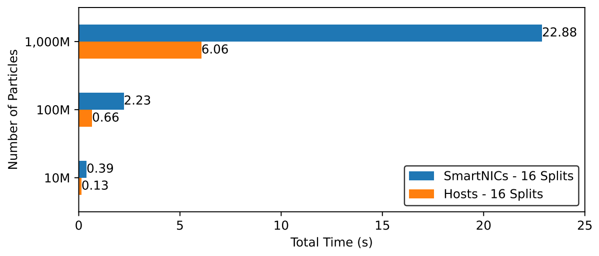

Fig. 9: Total sifting time for different input datasets

E. Discussion

In terms of raw performance, the hosts are noticeably faster than the SmartNICs at sifting the particle dataset in a distributed manner. However, there are multiple scenarios where lower performance is acceptable, such as when time step snapshots take place infrequently or host memory is highly constrained. In these examples it is valuable for the host to be able to rapidly pass data to the SmartNIC, reclaim memory, and return to the simulation.

The overhead of serializing data and injecting data to the SmartNIC was substantially higher than expected and a significant opportunity for improvement. Future work will focus on optimizing the transfer path between the host and its local SmartNIC. NVIDIA's recent DOCA library [31] includes host-to-card DMA transfer software that is expected to remedy this problem. It is also likely that converting, serializing, and injecting data in smaller fragments will help pipeline the process.

This case study demonstrates that Faodel and Arrow can provide a useful environment for hosting data management services on a collection of SmartNICs. The ability to change the behavior of the system by supplying a configuration with different pool definitions enabled us to fine-tune the implementation without rebuilding the software.

VII. Summary

SmartNICs offer a new location in HPC architectures for hosting data management services. Constructing a software stack that can support these services involves developing a communication plane that allows different endpoints in the platform to interact with the SmartNIC, and data processing software that can efficiently dispatch computations on datasets that adhere to a well-defined data model. Future work with SmartNICs must create a stronger coupling between the host and its local SmartNIC, and take advantage of vendor-specific features for accelerating performance.

Acknowledgment

This material is based upon work supported by the U.S. Department of Energy, Office of Science, Office of Advanced Scientific Computing Research under Field Work Proposal Number 20-023266. Sandia National Laboratories is a multimission laboratory managed and operated by National Technology & Engineering Solutions of Sandia, LLC., a wholly owned subsidiary of Honeywell International Inc., for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA0003525. This paper describes objective technical results and analysis. Any subjective views or opinions that might be expressed in the paper do not necessarily represent the views of the U.S. Department of Energy or the United States Government.

References

[1] R. Ross, G. Amvrosiadis, P. Carns, C. Cranor, M. Dorier, K. Harms, G. Ganger, G. Gibson, S. Gutierrez, R. Latham, B. Robey, D. Robinson, B. Settlemyer, G. Shipman, S. Snyder, J. Soumagne, and Q. Zheng, "Mochi: Composing data services for high-performance computing environments," *Journal of Computer Science and Technology*, vol. 35, pp. 121–144, 01 2020.

[2] E. Deelman, T. Peterka, I. Altintas, C. D. Carothers, K. K. van Dam, K. Moreland, M. Parashar, L. Ramakrishnan, M. Taufer, and J. S. Vetter, "The future of scientific workflows," *The International Journal of High Performance Computing Applications*, vol. 32, pp. 159 – 175, 2018.

[3] W. Bhimji, D. Bard, M. Romanus, D. Paul, A. Ovsyannikov, B. Friesen, M. Bryson, J. Correa, G. Lockwood, V. Tsulăia, S. Byna, S. Farrell, D. Gursoy, C. Daley, V. Beckner, B. Van Straalen, D. Trebotich, C. Tull, G. Weber, N. Wright, K. Antypas, and Prabhat, "Accelerating science with the NERSC burst buffer early user program," in *Proceedings of the 2016 Cray User Group Conference*, 2016.

[4] C. Docan, M. Parashar, and S. Klasky, "DataSpaces: An interaction and coordination framework for coupled simulation workflows," in *Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing*, 2010.

[5] C. Ulmer, S. Mukherjee, G. Templet, S. Levy, J. Lofstead, P. Widener, T. Kordenbrock, and M. Lawson, "Faodel: Data management for next-generation application workflows," in *Proceedings of the 9th Workshop on Scientific Cloud Computing*, 2018.

[6] G. Pfister, "An introduction to the InfiniBand architecture," *High Performance Mass Storage and Parallel I/O*, pp. 617–632, 2001.

[7] C. Guo, H. Wu, Z. Deng, G. Soni, J. Ye, J. Padhye, and M. Lipshteyn, "RDMA over commodity Ethernet at scale," in *Proceedings of the 2016 ACM SIGCOMM Conference*, 2016.

[8] B. Alverson, E. Froese, L. Kaplan, and D. Roweth, "Cray XC series network," Cray Inc., Tech. Rep. WP-Aries01-1112, 2012.

[9] M. T. Bettencourt, R. M. J. Kramer, K. Cartwright, E. G. Phillips, C. C. Ober, R. P. Pawlowski, M. S. Swan, I. K. Tezaur, E. T. Phipps, S. Conde, E. C. Cyr, C. D. Ulmer, T. H. Kordenbrock, S. L. N. Levy, G. J. Templet, J. J. Hu, P. Lin, C. A. Glusa, C. Siefert, and M. W. Glass, "ASC ATDM level 2 milestone #6358: Assess status of next generation components and physics models in EMPIRE." Sandia National Laboratories, Tech. Rep. SAND2018-10100, 2018.

[10] G. J. Templet, M. R. Glickman, T. Kordenbrock, S. Levy, G. F. Lofstead, J. Mauldin, T. J. Otahal, C. D. Ulmer, P. M. Widener, and R. A. Oldfield, "Data services for visualization and analysis ASC level II milestone (7186)." Sandia National Laboratories, Tech. Rep. SAND-2020-9451, 2020.

[11] K. Moreland, C. M. Sewell, W. Usher, L.-T. Lo, J. S. Meredith, D. Pugmire, J. Kress, H. A. Schroots, K.-L. Ma, H. Childs, M. Larsen, C.-M. Chen, R. Maynard, and B. Geveci, "VTK-m: Accelerating the visualization toolkit for massively threaded architectures," *IEEE Computer Graphics and Applications*, vol. 36, pp. 48–58, 2016.

[12] H. C. Edwards, C. R. Trott, and D. Sunderland, "Kokkos: Enabling manycore performance portability through polymorphic memory access patterns," *Journal of Parallel and Distributed Computing*, vol. 74, pp. 3202–3216, 2014.

[13] G. Lentner, "Shared memory high throughput computing with Apache Arrow," in *Proceedings of the Practice and Experience in Advanced Research Computing on Rise of the Machines (learning)*, 2019.

[14] T. Willhalm, N. Popovici, Y. Boshmaf, H. Plattner, A. Zeier, and J. Schaffner, "SIMD-Scan: Ultra fast in-memory table scan using on-chip vector processing units," in *Proceedings of VLDB Endowment*, 2009.

[15] S. Salloum, R. Dautov, X. Chen, P. X. Peng, and J. Z. Huang, "Big data analytics on Apache Spark," *International Journal of Data Science and Analytics*, vol. 1, pp. 145–164, 2016.

[16] M. Rocklin, "Dask: Parallel computation with blocked algorithms and task scheduling," in *Proceedings of 14th Python in Science Conference*, 2015.

[17] J. Chakraborty, I. Jimenez, S. A. Rodriguez, A. Uta, J. LeFevre, and C. Maltzahn, "Skyhook: Towards an Arrow-native storage system," in *Proceedings of the 22nd IEEE International Symposium on Cluster, Cloud and Internet Computing*, 2022.

[18] M. Liu, T. Cui, H. N. Schuh, A. Krishnamurthy, S. Peter, and K. Gupta, "Offloading distributed applications onto smartnics using iPipe," *Proceedings of the ACM Special Interest Group on Data Communication*, 2019.

[19] J. Liu, C. Maltzahn, C. Ulmer, and M. L. Curry, "Performance characteristics of the BlueField-2 SmartNIC," *arXiv:2105.06619*, 2021.

[20] J. Liu, C. Maltzahn, M. L. Curry, and C. Ulmer, "Processing particle data flows with SmartNICs," in *Proceedings of the 2022 IEEE High Performance Extreme Computing Conference*, 2022.

[21] S. Karamati, C. Hughes, K. S. Hemmert, R. E. Grant, W. Schonbein, S. Levy, T. M. Conte, J. S. Young, and R. W. Vuduc, ""Smarter" NICs for faster molecular dynamics: a case study," in *Proceedings of the 2022 IEEE International Parallel and Distributed Processing Symposium*, 2022.

[22] N. Diamond, S. Graham, and G. Clark, "Securing InfiniBand networks with the Bluefield-2 data processing unit," in *Proceedings of the International Conference on Cyber Warfare and Security*, 2022.

[23] M. Bayatpour, N. Sarkauskas, H. Subramoni, J. M. Hashmi, and D. K. Panda, "BluesMPI: Efficient MPI non-blocking all-to-all offloading designs on modern BlueField Smart NICs," in *Proceedings of High Performance Computing: 36th International Conference, ISC High Performance 2021*, 2021.

[24] W. Lu, L. E. Peña, P. Shamis, V. Churavy, B. M. Chapman, and S. Poole, "Bring the BitCODE-moving compute and data in distributed heterogeneous systems," in *Proceedings of the 2022 IEEE International Conference on Cluster Computing*, 2022, pp. 12–22.

[25] "Acero: A C++ streaming execution engine," https://arrow.apache.org/docs/cpp/streaming_execution.html, [Online; accessed 2-February-2023].

[26] C. King, "Stress-ng: A tool to load and stress a computer system," http://kernel.ubuntu.com/git/cking/stress-ng.git, [Online; accessed 13-March-2023].

[27] J. Liu, Q. Koziol, H. Tang, F. Tessier, W. Bhimji, B. Cook, B. Austin, S. Byna, B. Thakur, G. Lockwood, J. Deslippe, and Prabhat, "Understanding the I/O performance gap between Cori KNL and Haswell," in *Proceedings of the 2017 Cray User Group Conference*, 2017.

[28] F. H. Harlow, "The particle-in-cell method for numerical solution of problems in fluid dynamics," Los Alamos National Laboratory, Tech. Rep. LADC-5288, 1962.

[29] K. Matyash, R. Schneider, F. Taccogna, A. Hatayama, S. Longo, M. R. Capitelli, D. Tskhakaya, and F. X. Bronold, "Particle in cell simulation of low temperature laboratory plasmas," *Contributions to Plasma Physics*, vol. 47, 2007.

[30] E. Fourkal, B. Shahine, M. Ding, J. S. Li, T. Tajima, and C. M. C. Ma, "Particle in cell simulation of laser-accelerated proton beams for radiation therapy," *Medical physics*, vol. 29 12, pp. 2788–98, 2002.

[31] I. Burstein, "NVIDIA data center processing unit (DPU) architecture," in *Proceedings of the 2021 IEEE Hot Chips 33 Symposium*, 2021.