RaDD Runtimes: Radical and Different Distributed Runtimes with SmartNICs

Preprint · November 2020

3 authors, including:

Ryan E. Grant
Sandia National Laboratories
81 PUBLICATIONS **591** CITATIONS

SEE PROFILE

RaDD Runtimes: Radical and Different Distributed Runtimes with SmartNICs

Ryan E. Grant*†
Sandia National Laboratories
Center for Computational Research
Albuquerque, New Mexico
regrant@sandia.gov

Whit Schonbein
Sandia National Laboratories
Center for Computational Research
Albuquerque, New Mexico
wwschon@sandia.gov

Scott Levy
Sandia National Laboratories
Center for Computational Research
Albuquerque, New Mexico
sllevy@sandia.gov

ABSTRACT

As network speeds increase, the overhead of processing incoming messages is becoming onerous enough that many manufacturers now provide network interface cards (NICs) with offload capabilities to handle these overheads. This increase in NIC capabilities creates an opportunity to enable computation on data in-situ on the NIC. These enhanced NICs can be classified into several different categories of SmartNICs. SmartNICs present an interesting opportunity for future runtime software designs. Designing runtime software to be located in the network as opposed to the host level leads to new radical distributed runtime possibilities that were not practical prior to SmartNICs. In the process of transitioning to a radically different runtime software design for SmartNICs there are intermediary steps of migrating current runtime software to be offloaded onto a SmartNIC that also present interesting possibilities. This paper will describe SmartNIC design and how SmartNICs can be leveraged to offload current generation runtime software and lead to future radically different in-network distributed runtime systems.

CCS CONCEPTS

• **Networks** → **Network architectures**; **In-network processing**; *Network measurement*; • **Software and its engineering** → *Message oriented middleware.*

1 INTRODUCTION

Recently, there has been an explosion in the complexity and number of Network Interface Controller (NIC) designs. There are several physical packaging factors driving the required size of NIC silicon due to high speed signaling that have created 'white' silicon (empty space) that can be used for increased NIC capabilities and even full traditional compute cores. This situation has resulted in the emergence of several SmartNICs on the market, including proposals for radically different new models of computation for in-network compute. It is not clear yet which model will emerge the winner in this space, but regardless of the winning model, there will be opportunities to design and implement new types of runtime software that run in-network.

Given the large space and quick movement in SmartNIC designs, the term 'SmartNIC' applies to several different types of compute-capable NICs. For the purposes of this paper, we primarily discuss SmartNICs designed for HPC/Scientific computing systems. Within this space, more traditional SmartNIC designs offer packet processing engines (PPEs) providing inline compute capabilities that operate on incoming packets in a time window determined by the NIC buffer size and packet arrival rate (e.g., sPIN [32]). Such designs are able to perform several hundred operations on each packet, and only compute on packet arrival. An emerging alternative approach is not limited to compute on packet arrival, and does not have compute time deadlines. This alternative can be implemented out of the packet pipeline – as in Mellanox Bluefield [39] – but in-pipeline solutions have been proposed as well [47]. In this paper, we are particularly interested in the opportunities afforded by non-deadline-based SmartNICs.

An existing runtime software challenge that could be addressed by SmartNICs of this type is resource utilization and sharing. Runtimes that require compute cycles must try to use compute time that does not interfere with host applications. Suspending application execution contexts to perform runtime operations is not ideal as even suspension on a single compute core on a high utilization host can cause system noise that may propagate across execution contexts when synchronization occurs. In other words, if one process is delayed by runtime software execution and reaches a barrier late, this laggard process may delay progress on all application processes. For runtime communication libraries like MPI, moving some of the processing to network hardware makes sense because it offloads the sorts of tasks that may introduce system noise in the form of unexpected delays. In addition, offloading MPI can provide constant message processing progress without having to share compute resources with the host and keep data for the communication library close to where the communication takes place. Resource management is a challenging problem on modern systems, particularly when trying to manage resources beyond a single node in a coordinated manner. SmartNICs provide an interesting opportunity to craft new resource management runtimes that share data using in-network resources on a consistent basis could be used to solve this problem with a intercommunication level and decision granularity that would consume resources at levels that would not be commonly acceptable on a host CPU.

SmartNICs can provide additional resources that support heavier-weight runtime solutions without disturbing the application compute resources. Runtime systems that were impractical prior to SmartNICs due to high resource usage can now be implemented to accelerate application and system performance. In addition, locating these

*Also University of New Mexico Assistant Research Professor, Department of Computer Science, Albuquerque, New Mexico.

†Sandia National Laboratories is a multimission laboratory managed and operated by National Technology and 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.

PREPRINT, Nov 2020, USA
2020. ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

runtime solutions on a SmartNIC would potentially allow for idle network resources to be scavenged, thereby minimizing interference with the host applications. For example, it would be possible for an advanced future runtime to enable streaming of data to and from storage during observed low network-utilization periods. This is easily observable from the SmartNIC itself without requiring inquiries over a system bus like one would have to do with a CPU-side runtime.

SmartNICs provide opportunities beyond simple offloading of software to the NIC. We can envision runtimes that provide system tuning and resource coordination that are independent of hosts entirely. Runtimes on SmartNICs can speculatively copy data and intelligently schedule communication. As the SmartNIC is itself a single shared resource for the node, it is a logical place to marshal node resources and observe overall patterns in multi-application workflows for shared node usage or application-analytic paired workflows.

This paper addresses the new opportunities for radically different runtime deployment and design in the context of SmartNICs. We will detail SmartNIC capabilities and discuss both near-term runtime on SmartNIC opportunities as well as long term radically different distributed runtime designs that could run on SmartNICs in the future.

In this paper, we make the following contributions:

(1) detailed descriptions of different emerging SmartNIC architectures and how they can be used;
(2) demonstrate the opportunities with different SmartNICs approaches for distributed runtimes;
(3) describe new distributed runtime approaches that can be node independent;
(4) enumerate multiple new ideas for distributed runtimes that SmartNICs make possible.

In the remainder of this paper, we survey the relevant background on SmartNICs (Section 2). Then we detail the features of SmartNICs that support offloading of runtime systems and distributed runtime in Section 3. We then propose several novel distributed runtime applications that SmartNICs can make possible, concentrating on resource management, system tuning and I/O use cases in Section 4. Finally, we conclude with a discussion of the approaches and directions for future research trailblazing in this area.

2 BACKGROUND AND RELATED WORK

Network architects first introduced *offload* or *in-network* processing to minimize CPU involvement in packet processing, and to mask packet processing overheads by overlapping them with other CPU computations. Modern network technologies with sophisticated processing engines (PEs) include Mellanox's Bluefield [39] devices, Broadcom's Stingray SmartNICs [8], Netronome's Agilio SmartNICs [41], and other technologies such as CORE-Direct [27], SHARP [26], FlexNIC [35], Azure SmartNICs [21], Mellanox Innova FPGA SmartNICs, sPIN [32] and INCA [47]. These modern SmartNICs succeed a previous generation of such solutions from the early 2000s, such as Myrinet [5], Quadrics [42] and early Portals [3] NICs like Seastar [7].

While SmartNIC technologies vary widely in their underlying hardware – ASICs, network processors, general-purpose CPUs, FPGAs, etc. – they traditionally target low- or mid-level network applications such as packet classification, traffic shaping, support for offloaded multicasting, and similar applications. For example, Ethernet-based packet processing engines (PPEs) with limited processing capabilities have been used to offload processing for lower-level network protocols like TCP and UDP [12–14, 28, 44], researchers have explored the performance impacts of offloading collective operations [10, 30, 53], and recent work explores generalizability [45] and offloading at the NIC [27] and switch [26].

Recent work has also explored message rate efficiency. Message matching (demultiplexing) is important for the majority of large scientific applications, e.g., by virtue of using MPI to manage interprocess communication [4]. However, different host architectures provide greatly different capabilities [2], and host-side message matching may be difficult [46]. Differing host capabilities might be made consistent if handled instead by a SmartNIC, and if matching is not supported natively in SmartNIC hardware [16, 40], approaches developed from a host-side perspective illustrate architectural solutions that may be useful for SmartNICs that also support generalized compute [18, 22, 49]. Finally, matching may be avoided by applications that use one-sided communication semantics like OpenSHMEM [11] or alternative MPI RMA communications with modern MPI libraries that can support efficient multi-threaded solutions [31, 34, 48, 49, 57].

New terminology regarding SmartNICs has tried to differentiate between historical usage of the term 'SmartNIC' and modern SmartNIC designs that we consider in this paper. Several vendors have called the modern SmartNICs we describe in this paper as Data Processing Units (DPUs). For example, an early adopter of the terminology was Fungible's DPU [50]. Runtimes that find success on SmartNICs can easily be supported on most DPU designs, as the term could be applied to many of the most capable SmartNIC designs that we have already listed. In this paper, we will not consider SmartNIC devices that are proprietary like Amazon's Nitro [1], with built in support for particular Cloud computing infrastructures.

Moreover, researchers have proposed using offloaded capabilities for purposes other than originally intended. For example, [47] show how hardware supporting message demultiplexing and collective communications can execute arbitrary kernels to assist host applications, [37] show how MPI message matching hardware can be leveraged to provide a distributed key-value store, [15] show how SmartNIC technology targeting match-action processing (e.g., P4 [6]) can be exploited to offload consensus algorithms, and [25] show how similar hardware can be exploited to perform edge detection for autonomous vehicle guidance. These sorts of generalized compute applications in which NIC hardware is used for compute that was not the intended purpose of the hardware, we believe, herald the future of processing in the network.

2.1 Processing Deadlines

The majority of the works surveyed above share the longstanding strategy of processing data streams as they reside on a NIC. This has two implications. First, when there is no data flowing into a NIC, no work is performed, i.e., no processing occurs when the network is

idle. Second, when data does arrive, computational resources must be available to process that data. Consequently, these approaches work under an implicit *deadline* determined by network speed, buffer availability, number of compute elements, etc. Failing to meet the deadline means the task has failed, and may invalidate the data stream where the failure occurred, resulting in an unrecoverable, undefined data state, retransmisson of data, or even termination of the application. Because of this deadline, the amount of work NIC compute resources can do for any given packet is limited.

Processing deadlines will only become worse as network data speeds increase. For example, recent SmartNICs typically comprise a collection of ARM processors. With many cores, a device may be able process 32 packets in parallel, meaning the deadline is the time to process a packet through the NIC times 32. Assuming 32 2.5 GHz cores with an IPC of 1, 256 byte packets, and a 200Gb/s network, each core can execute at most 819 instructions before it must be available to process the next packet. When network speeds increase to 400Gb/s, all else being equal, the number of available instructions is nearly halved, to approximately 420. Unfortunately, as network speeds increase, we cannot reasonably expect core clock speeds or the number of cores to increase proportionally. Consequently, as network speeds increase and deadlines decrease, packet processing kernels will not necessarily be forward-compatible, making them not suitable candidates for runtime deployment.

There are two strategies for mitigating processing deadlines while retaining the benefits of near-data processing. The first is to move processing *outside* of the packet processing pipeline but nonetheless keep it near the network (i.e., on-NIC). Mellanox Bluefield SmartNICs [39] are a prominent example of this approach. The second is to leverage the capabilities of existing in-pipeline processing elements by redirecting their outputs back into the pipeline so that further processing can be done. INCA [47] and [25] are examples of the latter.

In this work, we utilize the deadline-free capable SmartNIC designs as a baseline to propose radically different system software runtimes operating in the network. Therefore, from this point forward when we refer to SmartNICs we only mean SmartNICs in the deadline-free general compute capable class like Mellanox Bluefield or INCA devices.

2.2 Runtime Systems and Middleware

In this paper we use the term "runtime" to refer to runtime systems and middleware. The runtimes that we are envisioning on SmartNICs are not runtimes in the traditional sense as they are not helping to manage the heap or garbage collection during program runtime, nor are they built-in features to a programming language. The runtime systems and middleware approaches discussed in this paper refer to software executing during the runtime phase of a program execution. The purpose of these runtime systems is primarily to enhance application performance through system tuning, high performance communications, and storage. Alternatively such middleware can also provide features like resilience/error recovery and performance portability. Examples of such runtime systems include MPI libraries, I/O libraries [38] and performance enhancing runtimes, including those using AI/ML techniques [29].

3 DISTRIBUTED RUNTIMES ON SMARTNICS

In order to understand why distributed runtimes on SmartNICs is a useful and viable solution – even if it is a radical departure from traditional runtime/middleware solutions – we provide a deep-dive into non-deadline based SmartNIC compute architectures in this Section. Specifically, we survey features of non-deadline based SmartNICs that could be leveraged for radically new runtime designs.

3.1 SmartNIC Architectural Support

3.1.1 Operating System Support. Non-deadline-based SmartNICs provide the basic programming environment expected for high-performance middleware development, even if they do not provide local operating system support. While some non-deadline SmartNICs, e.g., NVIDIA Mellanox BlueField, support Linux operating system environments, other proposed architectures like INCA execute code natively, without operating system support. SmartNICs that lack local operating system support can access operating system services by shipping system calls to heavyweight OS instances. However, a great deal of high-performance middleware *avoids* system calls, as demonstrated by studies of lightweight operating system kernels [24]. Therefore, because the performance implications of system call shipping are not a major concern, we limit our discussion of the operating system environment on SmartNICs in this paper.

3.1.2 Independent Progress. From the perspective of middleware development and deployment, the most important characteristic of a SmartNIC is that it supports execution progress that is independent of any interaction with the host, other host-side components (e.g., GPUs or FPGA accelerators), or incoming network traffic. In this model, the SmartNIC is essentially a separate, independent host within a node or, at the very least, an independent, accelerator-type compute resource. Support for efficient communication with the host CPU or other components can be provided via the PCIe bus, enabling command exchange between the SmartNIC and these components. In this way, components can signal runtime software on the SmartNIC to perform operations to assist an application, bypassing the host CPU. For example, a SmartNIC running communication software (e.g., MPI) could provide communication support for host-side components that are not efficient at running such software, e.g., GPUs. Likewise, storage middleware can stream data into and out of the node on-demand via signals from the application on the host CPU, offloading such tasks so that the main host resources can be devoted to computation and the production/consumption of that data. I/O middleware like Stitch-IO [38] is a good example of such an application.

3.1.3 New Memory Hierarchy. SmartNICs essentially provide a new level of compute and memory storage in HPC systems. They allow data to be sent and received to local storage on the SmartNIC and can also interact with main host memory and other devices attached to the PCIe bus. Because they are responsible for fast data movement over networks, they can also be leveraged to dynamically steer incoming data to devices, including locally to keep data in the SmartNIC.

Because SmartNICs represent a fundamental change to node architecture, they make memory ownership management more complicated. New architecture solutions for managing memory migration between devices, and shared/private memory ownership on a per-device basis, will help in managing memory on a node. For example,

Compute Express Link (CXL) [54] allows devices in a node to have full private control of memory so that the device can cache data without the overheads of having to manage cache coherency between multiple devices. CXL also allows devices to share memory spaces in a coherent manner with corresponding coherency traffic overheads. In addition, one can switch memory space control, passing it over to other devices and relinquishing a coherent view of a given memory space. For example, a CXL-enabled GPU could have private ownership of a memory region during compute kernel execution, allowing for fast caching and good performance. Then, when the data is ready to be shared, the GPU could enter a shared coherent memory space with the CPU for a portion of the output and pass full ownership control of a different memory region to the SmartNIC.

Approaches like CXL can be used to quickly transfer ownership of memory regions and move chunks of data from one device's memory space to another. This means that the extra architectural level of compute complexity provided by SmartNICs can be leveraged without concerns about SmartNIC compute performance. SmartNICs can compute independently at all times, and if a runtime or user application chooses to migrate compute off of the SmartNIC to a faster compute device in the system, it can leverage memory bus solutions like CXL to quickly move data throughout the node.

3.2 Execution Environment on SmartNICs

SmartNICs are still an emerging technology, and as such do not have common standardized interfaces. While efforts like OpenSNAPI are attempting to provide such standardized interfaces, it will most likely be several years before such interfaces are ready. Therefore, we will detail features of current and next generation SmartNICs that will facilitate runtime development in the near future.

3.2.1 Language and Operating System Support. The majority of existing HPC middleware programs are written in C/C++. Providing support for building and deploying programs written in C/C++ on SmartNICs will enable these existing tools to be rapidly deployed. Similarly, Python has become an increasingly important language for data analysis, machine learning, and artificial intelligence tools. For SmartNICs that support Python language tools, it will be possible to rapidly develop and deploy programs that exploit these techniques. Several earlier-generation and current SmartNICs provide such language development environments [e.g. 32, 39, 47, 55]. Full support for a Linux (or other Unix-type) operating system will also facilitate the rapid deployment of existing source code. However, as noted in Section 3.1.1, support for a complete operating system instance is not an absolute requirement.

3.2.2 Host Communication. Host communication is another key aspect of SmartNIC programming environments that make SmartNIC runtimes possible. There are a variety of mechanisms that can be used to enable SmartNIC-host communication. Typical system bus communication, e.g., PCIe, is the most obvious choice, but more sophisticated memory sharing models like CXL could also be used to created shared memory regions for communication. Traditional interrupts can also be used to communicate with the host CPU. However, due to the performance implications of interrupting the CPU, this is only viable for runtimes that require infrequent communication with the host.

3.2.3 Inter-SmartNIC Communication. A SmartNIC needs to be able to communicate with other SmartNICs, not just their local host CPU. However, the entire purpose of non-smart NICs is to communicate with other NICs. Therefore, this can be accomplished with traditional network communication. The only difference between traditional network communication and inter-SmartNIC communication is that data and control packets can be steered so that they can be processed only by the SmartNIC, if required. This kind of packet filtering/steering is supported in all non-deadline-based SmartNIC designs that the authors are aware of. While some solutions are more difficult to implement than others, the feature is prevalent in the SmartNIC ecosystem.

3.2.4 Host-SmartNIC Coordination. Because the SmartNIC comprises a distinct 'host' (Section 3.1.2), its general-purpose compute capabilities will likely differ from those of the host CPU (e.g., they may be slower). Consequently, when offloading runtimes, situations may arise in which a host must wait for an on-NIC runtime operation to complete before continuing computation. In such situations, a SmartNIC could block host-side compute, and negatively impact performance. For this reason, some form of host-SmartNIC coordination is required. There are two solutions to this problem. The first is to implement the ability to migrate computation off the SmartNIC with low-latency when a faster host-side component becomes available [47]. The second is for runtime developers to only offload code that avoids these situations. Constraining the possibilities for runtime design on SmartNICs is not ideal, but until the former sort of 'fast hand-off' is a universal feature, it is the only solution to creating portable runtimes for SmartNICs.

4 FUTURE DISTRIBUTED RUNTIME DESIGN

With the baseline capabilities of the non-deadline based general compute class of SmartNICs, the possibilities for future entirely in-network runtime software are intriguing.

4.1 Distributed Adaptive Resource Management

Understanding and adapting to resource usage dynamically during job execution is a potentially important future use of SmartNICs. Dynamic resource management is a feature commonly attributed to cloud computing that distinguishes itself from supercomputing. With supercomputing workloads, assessing the resource utilization at every node involved in a job is time consuming and requires resources for coordination that take away compute resources from computationally intensive tasks.

For distributed resource management, developing a adaptive runtime solution that utilizes SmartNICs provides several advantages. First, the work of adapting resource pools and shifting resources, including overall power and thermal budgets, can be done using a component that is not in the compute path. Second, the SmartNIC is closer to the network than other devices in the system, meaning it can isolate the performance impact on other node-level components and even the network itself if appropriate times can be chosen to use network bandwidth to exchange resource utilization data. As network counters are readily available at the SmartNIC, predictive methods could be used to time resource utilization exchange for broader job wide allocation of shared resources, such as job power.

Distributed adaptive resource management could also be helpful for container resource management. Such a runtime could coordinate the shutdown and restart of containers based on resource usage throughout the system. Unlike a traditional cloud computing approach, where such coordination utilizes an adaptive pool of coordination nodes for this purpose, SmartNICs could be allocated to perform this task instead. This would require a larger number of SmartNICs to coordinate tasks. For example, for generalized services one could use SmartNICs as Apache Zookeeper servers to serve clients. This would require more servers than a typical Zookeeper deployment, but one could easily match the number of clients with the number of SmartNICs and still have more compute nodes available for non-coordination roles than would otherwise be possible. This brings new resource management models into HPC systems with lower compute node resource requirements by leveraging SmartNICs. In addition, SmartNICs can enable low overhead use of modern cloud computing paradigms on HPC systems where desired.

4.2 System Tuning

SmartNICs are an ideal solution for offloaded management of system resources and tuning. Just like distributed resource management, offloading the computational requirements from the host CPU removes the cost-benefit tradeoff of attempting online system tuning. Instead of using compute resources that would otherwise be devoted to the compute job to enable tuning, the SmartNIC's limited compute resources can be used.

4.2.1 Network Tuning. For cases of network tuning, SmartNICs are an ideal choice as their compute resources are as close to the network performance counters and tuning mechanisms as possible. As SmartNICs can also observe traffic through the endpoint interface they can also potentially predict and respond to spikes in demand or lack thereof. For example, a SmartNIC may tune the desired header type or encapsulation method to better tune the network to the data payload sizes of the application. Lightweight headers with small maximum transmission unit (MTU) sizes could lead to better network latency while optimizing for large MTU transmissions would be desirable for codes with high bandwidth needs. For some networks that provide different levels of quality of service, these channels can also be optimized. Many QoS solutions also optimize for small versus large messages allowing traffic separation that prevents head of line blocking and provides latency or bandwidth guarantees.

4.2.2 System Performance Tuning. In the case of general system tuning, where the number of used compute cores, threads versus process count and system settings like CPU clock speeds, SmartNICs can be useful. This is a case where unlike the network tuning situation, the location of the SmartNIC itself in a node architecture is a limitation. Direct access to CPU and GPU level settings and performance counters is challenging from a SmartNIC perspective without helper threads on the devices themselves. This will lead to longer latencies in reading counters and similar longer latencies in changing device settings than running entirely natively on the devices within the operating system protection domain.

It is possible to take existing runtime designs that attempt to optimize system performance natively on the device. For example, if we consider Intel's GEOPM project [19] that attempts to optimize CPU performance by adjusting clock frequency and exploiting slack in large HPC jobs we can consider a new type of hybrid runtime software. GEOPM uses MPI to exchange data with a job manager that attempts to equalize performance across many nodes to reduce wait time due to uneven execution time of kernels on multiple nodes. Such network data consolidation and exchange responsibilities could be offloaded onto a SmartNIC, exchanging data with the CPU-side runtime collecting high frequency data. By removing the network communication task from the CPU, the runtime can be freed to perform more intensive operations to optimize local performance. SmartNIC-side runtime components can similarly filter incoming data from the job manager, only updating the CPU-side runtime when new significant job level data changes occur. This frees the CPU-side runtime from having to poll the network for arrival of new data and processing thereof.

4.3 Data Storage and Management

SmartNIC architectures frequently include additional memory resources. For example, the NVIDIA Mellanox BlueField SmartNIC supports up to 16 GB of on-board DDR4 memory [39], and Netronome's Agilio CX line has 2 GB of DDR3 memory [41]. Because these memory resources are in addition to the CPU's memory, they represent storage that can be leveraged in interesting ways. Moreover, because the SmartNIC's computational resources are isolated from the CPU, it can provide intelligent storage services without stealing CPU cycles from or competing for CPU resources with (e.g., introducing cache pollution with library calls) the principal application.

4.3.1 Expanding the Data Storage Hierarchy. The combination of processing and storage that is available on many SmartNICs provides an opportunity to create a fast, global storage resource. Unlike the typical deployment of other distributed storage resources (e.g., SSD, NVMe) the additional computational resources of SmartNICs allows for the potential deployment of more sophisticated storage. For example, we could use SmartNICs to deploy a distributed hash table (DHT), *see e.g.,* [37] (implementing a parallel global logical address space over a DHT implemented using the Portals networking interface), [17, 51, 52] that would provide high-performance, global, runtime storage outside of file system abstractions. By extending its communication across a platform, a DHT would enable high-performance communication and data exchange between cooperating applications, e.g., coupled physics codes, visualization, analysis. Because of the flexibility provided by a SmartNIC's computational resources, it would also be straightforward to introduce persistence semantics into the DHT that would allow for the decision of when and what to write to persistent storage to be configured.

4.3.2 Offloading I/O Management. Scientific simulation remains a critically important class of HPC applications. Many of these simulations continue to be implemented as Bulk-Synchronous Parallel (BSP) applications. However, BSP applications pose significant challenges to HPC I/O systems because their I/O operations are typically very tightly synchronized. BSP execution phases are coordinated across processes and I/O operations are largely confined to the end of an execution phase. As a result, provisioning the capacity of high-performance I/O resources is frequently driven by these periods of high utilization even if the average demand is modest. Because

SmartNICs co-locate additional compute and storage capacity with the networking hardware, they have the potential to be able to store data bound for persistent storage and coordinate data transfer to storage resources with periods of low network activity.

4.3.3 Reducing Data Movement for Visualization. Visualization and analysis are critically important for understanding the output of scientific simulations. However, moving data around HPC systems, e.g., from simulations to visualization and analysis tools, can be extremely time-consuming and energy-intensive [36, 56]. To help reduce the amount of data movement that needs to be transferred for visualization and analysis, we can exploit the computing and storage resources provided by SmartNICs to pre-process data before sending it to these tools without interfering with the ongoing simulation. For example, if the visualization tools require only a subset of the total data set, e.g., data for a single timestep or for a portion of the physical space being simulated, the SmartNIC could ensure that only the relevant subset of the total data is transferred over the network. Similarly, the SmartNIC could be used to analyze data from the simulation and only deliver the results of that analysis instead of the entire data set to a remote analysis tool. By leveraging SmartNICs, we can potentially reduce overall data movement costs while minimizing disruptions of the ongoing simulation.

4.4 Resilience and Recovery

SmartNICs enable new runtime system solutions for supercomputer resilience. As SmartNICs are independent devices in a node, with separate hardware and operating systems, they can continue operation when other components fail. For example, if a CPU experiences a kernel panic software error, the CPU and application will crash in an unrecoverable manner. A SmartNIC will continue operation in this situation and can be used to observe and inform the compute system's RAS solution of the crash. This is a fundamental new feature that a SmartNIC can provide that is useful for error notification and recovery.

4.4.1 Managing Communication State After CPU Failure. In a typical scenario where a compute node's application has crashed due to unrecoverable software or hardware failure, the node becomes unreachable. Therefore, failures present as nodes appearing to go dark and stop responding to remote nodes for long periods of time. Determining that a node has failed and identifying which node has failed typically requires some form of distributed consensus protocol to be implemented for determining which nodes have failed, *see* [9, 15]. Because a SmartNIC represents an entirely independent execution context from the host CPU, it can be leveraged to both: reduce interference from the consensus protocol on the execution of the main computation; and accelerate the identification of failed nodes when the failure is limited to resources associated with the host CPU.

Moreover, even if the surviving nodes determine that a node has failed, it is impossible to know if it will ever return to responding to outside queries at some point in the future. This leads to resilience mechanisms like those that have been implemented for MPI in the past. Such solutions rely on consensus that all nodes are operational and responsive. Failure for a node to be responsive results in that node being culled from the communication group and potentially being replaced by another node. It is very important that the "failed" node be culled from the communication group so that if it resumes operation at some future point, other nodes in the job no longer accept messages from the failed node as it may no longer be in sync with the application's execution. SmartNICs can aid in this process as they could be used to assess a given node's status and if it needs to be culled from a communication group, the SmartNIC can prevent further communication over those channels rather than the communication group having to filter out messages from the culled node.

4.4.2 Recovering Application State After CPU Failure. While mechanisms to recover from a crashed CPU-side operating system hard crash do not yet exist in SmartNICs, such capabilities are possible. For example, SmartNICs could be engineered to signal hard reset of CPUs while maintaining main memory state from prior to the crash. This could enable new recovery mechanisms to rebuild application memory and CPU states. Alternatively, SmartNIC runtime systems could be built to transfer data from the main memory of crashed nodes to replacement nodes for continued operation. Obviously such solutions would only work for failures not caused by main memory errors.

4.4.3 Managing Checkpoints. Currently, the most common approach to fault tolerance is some form of checkpoint/restart, application state is periodically written to persistent storage and when a failure occurs the application can be restarted from its saved state. Because a large part of implementing checkpoint/restart is solving an I/O problem (e.g., moving large blocks of data to storage, managing contention for storage resources), the discussion in Section 4.3 applies with equal force here. Additionally, the SmartNIC could also be used to reduce the volume of checkpoint data that needs to be be written to persistent storage. For example, the SmartNIC could be used to compress the checkpoint [33] while the CPU continues executing the main application or it could be used to save only the portion of the application's state that has changed since the last checkpoint, *cf.* incremental checkpointing [43].

4.5 Failure Prediction

The holy grail of fault tolerance is to be able to predict when and where a failure is going to occur and then migrate the application's resources around the failure. In recent years, significant effort has been devoted to trying to achieve this goal (*cf.* [20, 23]). SmartNICs can potentially be leveraged in the pursuit of this goal by offloading and augmenting node state monitoring and analysis. By removing the failure prediction function from the CPU where it competes with the principal computation for resources, it may be possible to perform more sophisticated analysis. Moreover, because the compute and memory resources on SmartNICs are close to the network, it also opens the possibility for using SmartNICs to share state information across nodes to develop a more complete picture of the overall state of a job's hardware resources.

5 DISCUSSION

In this section we answer several common questions about RADD and how runtimes could be deployed on future generation SmartNICs.

What kind of performance are runtimes expected to have on SmartNICs? Current SmartNIC designs have a wide variety of compute hardware and capabilities. For those designs with traditional

compute cores like Mellanox/NVIDIA Bluefield, they are typically modest multi-core solutions that would exceed the hardware normally allocated to executing runtime software on a compute node. Alternative deadline free designs like INCA are supplemented by accelerators meaning that certain classes of code will run at very high performance, while others will be lower performance than a general compute core. Other solutions are not entirely deadline free but introduce custom flexible hardware like FPGAs. For an FPGA SmartNIC, runtimes could be designed entirely in hardware with performance guarantees.

Won't offloading runtimes interfere with the primary purpose of a SmartNIC, namely, to enable communication? The short answer is, 'no'. There are at least two strategies for avoiding interference. The first is to make the resources supporting runtime offloading more-or-less *independent* from those dedicated to traditional network processing or protocol offloading. Mellanox Bluefield can be viewed from this perspective: the general purpose processor and its associated memory can interact with components dedicated to traditional network processing, but otherwise it functions as an independent, on-NIC host. The second strategy is to intelligently *share* compute resources between runtime offloads and network functionality. For example, a design such as INCA allows offloaded kernels to be *preemptible* in the sense execution can be temporarily suspended, permitting high-priority data transfers to be processed without being slowed down by computation occurring on the NIC. This ability allows for work to be done beyond typical packet-processing deadlines (Section 2.1) as resources are allocated to accommodate runtimes when they are not immediately needed by other incoming packets. Moreover, this strategy turns NIC idle time into a useful compute resource.

Can my host be forced to wait on the distributed runtime on the NIC? If there are dependencies that exist on the host that can result in waiting for runtime software on the SmartNIC to complete, this could become an issue. However, distributed runtimes for system tuning could be designed without these dependencies to avoid such waiting. Alternatively, some non-deadline based designs can be uninterruptible [47] and solutions like Compute Express Link (CXL) can manage shifting ownership of runtime memory from the SmartNIC to other devices on the host. In this way some SmartNIC designs can quickly shift execution to the fastest available compute unit in a node, alleviating this concern.

What does locating runtimes at the network level do for performance? There are two performance advantages to executing runtime software on a SmartNIC. First, it offloads the computation and memory bandwidth requirements to a separate device that is not used for application compute. Second, it makes network performance information easily and quickly accessible. If a runtime is trying to infer network activity or manage network resources, accessing the internal NIC counters is drastically more efficient from the device rather than gathering counter data over a system bus. Such performance advantages can be seen in performance counter data gathering approaches on CPUs where local counter data is a simple register fetch operation.

6 CONCLUSION

In this paper we have examined the features of a class of SmartNICs that enable the design of a new type of distributed runtime located in the network. We described several useful features that SmartNICs need to provide to make runtime development practical and portable. We provided several use cases and designs for future runtime systems to run on SmartNICs, detailing the support that such runtimes need and the benefits they can observe through execution near the network. This paper has provided the foundation for new runtime designs, using emerging technologies that provide a unique opportunity for evolution of runtime systems for supercomputing.

REFERENCES

[1] Amazon. [n. d.]. AWS Nitro System. https://aws.amazon.com/ec2/nitro/. (Accessed on 08/27/2020).

[2] Brian W Barrett, Ron Brightwell, Ryan Grant, Simon D Hammond, and K Scott Hemmert. 2014. An evaluation of MPI message rate on hybrid-core processors. *International Journal of High Performance Computing Applications* 28, 4 (2014), 415–424.

[3] Brian W. Barrett, Ron Brightwell, Ryan E. Grant, Scott Hemmert, Kevin Pedretti, Kyle Wheeler, Keith Underwood, Rolf Riesen, Torsten Hoefler, Arthur B. Maccabe, and Trammell Hudson. 2018. *The Portals 4.2 Network Programming Interface.* Technical Report SAND2018-12790.

[4] David E Bernholdt, Swen Boehm, George Bosilca, Manjunath Gorentla Venkata, Ryan E Grant, Thomas Naughton, Howard P Pritchard, Martin Schulz, and Geoffroy R Vallee. 2020. A survey of MPI usage in the US exascale computing project. *Concurrency and Computation: Practice and Experience* 32, 3 (2020), e4851.

[5] Nanette J Boden, Danny Cohen, Robert E Felderman, Alan E. Kulawik, Charles L Seitz, Jakov N Seizovic, and Wen-King Su. 1995. Myrinet: A gigabit-per-second local area network. *IEEE Micro* 15, 1 (1995), 29–36.

[6] Pat Bosshart, Dan Daly, Glen Gibb, Martin Izzard, Nick McKeown, Jennifer Rexford, Cole Schlesinger, Dan Talayco, Amin Vahdat, George Varghese, and David Walker. 2014. P4: Programming Protocol-independent Packet Processors. *SIGCOMM Comput. Commun. Rev.* 44, 3 (July 2014), 87–95. https://doi.org/10.1145/2656877.2656890

[7] Ron Brightwell, Kevin T Pedretti, Keith D Underwood, and Trammell Hudson. 2006. SeaStar interconnect: Balanced bandwidth for scalable performance. *IEEE Micro* 26, 3 (2006), 41–57.

[8] Broadcom. 2019. Stingray SmartNIC. Retrieved 2019-10-01 from https://www.broadcom.com/products/ethernet-connectivity/smartnic/ps225

[9] Darius Buntinas. 2012. Scalable distributed consensus to support MPI fault tolerance. In *2012 IEEE 26th International Parallel and Distributed Processing Symposium*. IEEE, 1240–1249.

[10] Darius Buntinas, Dhabaleswar K. Panda, and Ponnuswamy Sadayappan. 2001. Fast NIC-based barrier over Myrinet/GM. In *Proceedings 15th International Parallel and Distributed Processing Symposium. IPDPS 2001.* 52–59. https://doi.org/10.1109/IPDPS.2001.924993

[11] Barbara Chapman, Tony Curtis, Swaroop Pophale, Stephen Poole, Jeff Kuehn, Chuck Koelbel, and Lauren Smith. 2010. Introducing OpenSHMEM: SHMEM for the PGAS community. In *Proceedings of the Fourth Conference on Partitioned Global Address Space Programming Model*. 1–3.

[12] Christopher L Chappell and James Mitchell. 2012. Packet processing in switched fabric networks. Patent No. 8285907, Filed December 10th., 2004, Issued October 9th., 2012.

[13] Dennis Dalessandro, Ananth Devulapalli, and Pete Wyckoff. 2005. Design and implementation of the iWARP protocol in software. In *Proceedings of the 17th IASTED International Conference on Parallel and Distributed Computing and Systems*. Phoenix, Arizona, 471–476.

[14] Dennis Dalessandro, Pete Wyckoff, and Gary Montry. 2006. Initial performance evaluation of the neteffect 10 gigabit iwarp adapter. In *2006 IEEE International Conference on Cluster Computing*. IEEE, 1–7.

[15] Huynh Tu Dang, Daniele Sciascia, Marco Canini, Fernando Pedone, and Robert Soulé. 2015. NetPaxos: Consensus at Network Speed. In *Proceedings of the 1st ACM SIGCOMM Symposium on Software Defined Networking Research (SOSR '15)*. ACM, New York, NY, USA, 5:1–5:7. https://doi.org/10.1145/2774993.2774999 event-place: Santa Clara, California.

[16] S. Derradji, T. Palfer-Sollier, J. P. Panziera, A. Poudes, and F. W. Atos. 2015. The BXI Interconnect Architecture. In *2015 IEEE 23rd Annual Symposium on High-Performance Interconnects*. 18–25. https://doi.org/10.1109/HOTI.2015.15

[17] Ciprian Docan, Manish Parashar, and Scott Klasky. 2012. Dataspaces: an interaction and coordination framework for coupled simulation workflows. *Cluster Computing* 15, 2 (2012), 163–181.

[18] M Dosanjh, W Schonbein, RE Grant, PG Bridges, SM Gazimirsaeed, and A Afsahi. 2019. Fuzzy Matching: Hardware Accelerated MPI Communication Middleware. In *2019 19th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID)*. 210–220.

[19] Jonathan Eastep, Steve Sylvester, Christopher Cantalupo, Brad Geltz, Federico Ardanaz, Asma Al-Rawi, Kelly Livingston, Fuat Keceli, Matthias Maiterth, and Siddhartha Jana. 2017. Global extensible open power manager: A vehicle for hpc community collaboration on co-designed energy management solutions. In *International Supercomputing Conference*. Springer, 394–412.

[20] Christian Engelmann, Geoffroy R Vallee, Thomas Naughton, and Stephen L Scott. 2009. Proactive fault tolerance using preemptive migration. In *2009 17th Euromicro International Conference on Parallel, Distributed and Network-based Processing*. IEEE, 252–257.

[21] Daniel Firestone, Andrew Putnam, Sambhrama Mundkur, Derek Chiou, Alireza Dabagh, Mike Andrewartha, Hari Angepat, Vivek Bhanu, Adrian Caulfield, Eric Chung, et al. 2018. Azure accelerated networking: SmartNICs in the public cloud. In *15th USENIX Symposium on Networked Systems Design and Implementation (NSDI 18)*. 51–66.

[22] Mario Flajslik, James Dinan, and Keith D Underwood. 2016. Mitigating MPI message matching misery. In *International conference on high performance computing*. Springer, 281–299.

[23] Ana Gainaru, Franck Cappello, Marc Snir, and William Kramer. 2013. Failure prediction for HPC systems and applications: Current situation and open issues. *The International journal of high performance computing applications* 27, 3 (2013), 273–282.

[24] Balazs Gerofi, Masamichi Takagi, Atsushi Hori, Gou Nakamura, Tomoki Shirasawa, and Yutaka Ishikawa. 2016. On the scalability, performance isolation and device driver transparency of the IHK/McKernel hybrid lightweight kernel. In *2016 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. IEEE, 1041–1050.

[25] René Glebke, Johannes Krude, Ike Kunze, Jan Rüth, Felix Senger, and Klaus Wehrle. 2019. Towards Executing Computer Vision Functionality on Programmable Network Devices. In *Proceedings of the 1st ACM CoNEXT Workshop on Emerging In-Network Computing Paradigms (ENCP '19)*. Association for Computing Machinery, New York, NY, USA, 15–20. https://doi.org/10.1145/3359993.3366646 event-place: Orlando, FL, USA.

[26] Richard L Graham, Devendar Bureddy, Pak Lui, Hal Rosenstock, Gilad Shainer, Gil Bloch, Dror Goldenerg, Mike Dubman, Sasha Kotchubievsky, Vladimir Koushnir, et al. 2016. Scalable hierarchical aggregation protocol (SHArP): a hardware architecture for efficient data reduction. In *Proceedings of the First Workshop on Optimization of Communication in HPC*. IEEE Press, 1–10.

[27] Richard L Graham, Steve Poole, Pavel Shamis, Gil Bloch, Noam Bloch, Hillel Chapman, Michael Kagan, Ariel Shahar, Ishai Rabinovitz, and Gilad Shainer. 2010. Overlapping computation and communication: Barrier algorithms and ConnectX-2 CORE-Direct capabilities. In *Parallel & Distributed Processing, Workshops and Phd Forum (IPDPSW), 2010 IEEE International Symposium on*. IEEE, 1–8.

[28] Ryan E Grant, Mohammad J Rashti, Ahmad Afsahi, and Pavan Balaji. 2011. RDMA capable iWARP over datagrams. In *Parallel & Distributed Processing Symposium (IPDPS), 2011 IEEE International*. IEEE, 628–639.

[29] Taylor Liles Groves, Ryan E Grant, Aaron Gonzales, and Dorian Arnold. 2017. Unraveling Network-Induced Memory Contention: Deeper Insights with Machine Learning. *IEEE Transactions on Parallel and Distributed Systems* 29, 8 (2017), 1907–1922.

[30] K. Scott Hemmert, Brian Barrett, and Keith D. Underwood. 2010. Using Triggered Operations to Offload Collective Communication Operations. In *Recent Advances in the Message Passing Interface (Lecture Notes in Computer Science)*. Springer, Berlin, Heidelberg, 249–256. https://doi.org/10.1007/978-3-642-15646-5_26

[31] Nathan Hjelm, Matthew GF Dosanjh, Ryan E Grant, Taylor Groves, Patrick Bridges, and Dorian Arnold. 2018. Improving MPI multi-threaded RMA communication performance. In *Proceedings of the 47th International Conference on Parallel Processing*. 1–11.

[32] Torsten Hoefler, Salvatore Di Girolamo, Konstantin Taranov, Ryan E. Grant, and Ron Brightwell. 2017. sPIN: High-performance Streaming Processing In the Network. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC '17)*. ACM, New York, NY, USA, 59:1–59:16. https://doi.org/10.1145/3126908.3126970

[33] Dewan Ibtesham, Dorian Arnold, Kurt B Ferreira, and Patrick G Bridges. 2011. On the viability of checkpoint compression for extreme scale fault tolerance. In *European Conference on Parallel Processing*. Springer, 302–311.

[34] Humaira Kamal and Alan Wagner. 2010. FG-MPI: Fine-grain MPI for multicore and clusters. In *2010 IEEE International Symposium on Parallel & Distributed Processing, Workshops and Phd Forum (IPDPSW)*. IEEE, 1–8.

[35] Antoine Kaufmann, SImon Peter, Naveen Kr. Sharma, Thomas Anderson, and Arvind Krishnamurthy. 2016. High Performance Packet Processing with FlexNIC. In *Proceedings of the Twenty-First International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS '16)*. ACM, New York, NY, USA, 67–81. https://doi.org/10.1145/2872362.2872367

[36] Gokcen Kestor, Roberto Gioiosa, Darren J Kerbyson, and Adolfy Hoisie. 2013. Quantifying the energy cost of data movement in scientific applications. In *2013 IEEE international symposium on workload characterization (IISWC)*. IEEE, 56–65.

[37] D. Brian Larkins, John Snyder, and James Dinan. 2018. Efficient Runtime Support for a Partitioned Global Logical Address Space. In *ICPP 2018: 47th International Conference on Parallel Processing*. ACM, Eugune, Oregon.

[38] Jay Lofstead, John Mitchell, and Enze Chen. 2020. Stitch It Up: Using Progressive Data Storage to Scale Science. In *2020 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. IEEE, 52–61.

[39] Mellanox. 2018. Mellanox BlueField SmartNIC. Retrieved 2019-10-01 from https://www.mellanox.com/products/bluefield-overview

[40] Mellanox. 2018. Understanding MPI Tag Matching and Rendezvous Offloads (ConnectX-5). https://community.mellanox.com/s/article/understanding-mpi-tag-matching-and-rendezvous-offloads--connectx-5-x

[41] Netronome. [n. d.]. Agilio CX SmartNICs. https://www.netronome.com/products/agilio-cx/

[42] Fabrizio Petrini, Wu-chun Feng, Adolfy Hoisie, Salvador Coll, and Eitan Frachtenberg. 2002. The Quadrics network: High-performance clustering technology. *IEEE Micro* 22, 1 (2002), 46–57.

[43] James S Plank, Jian Xu, and Robert HB Netzer. 1995. *Compressed differences: An algorithm for fast incremental checkpointing*. Technical Report. Citeseer.

[44] Mohammad J Rashti, Ryan E Grant, Ahmad Afsahi, and Pavan Balaji. 2010. iWARP redefined: Scalable connectionless communication over high-speed Ethernet. In *High Performance Computing (HiPC), 2010 International Conference on*. IEEE, 1–10.

[45] Timo Schneider, Torsten Hoefler, Ryan E Grant, Brian W Barrett, and Ron Brightwell. 2013. Protocols for fully offloaded collective operations on accelerated network adapters. In *Parallel Processing (ICPP), 2013 42nd International Conference on*. IEEE, 593–602.

[46] Whit Schonbein, Matthew GF Dosanjh, Ryan E Grant, and Patrick G Bridges. 2018. Measuring multithreaded message matching misery. In *European Conference on Parallel Processing*. Springer, 480–491.

[47] Whit Schonbein, Ryan E Grant, Matthew GF Dosanjh, and Dorian Arnold. 2019. INCA: in-network compute assistance. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. ACM, 54.

[48] Min Si, Antonio J Peña, Pavan Balaji, Masamichi Takagi, and Yutaka Ishikawa. 2014. MT-MPI: multithreaded MPI for many-core environments. In *Proceedings of the 28th ACM international conference on Supercomputing*. 125–134.

[49] Min Si, Antonio J Peña, Jeff Hammond, Pavan Balaji, Masamichi Takagi, and Yutaka Ishikawa. 2015. Casper: An asynchronous progress model for MPI RMA on many-core architectures. In *2015 IEEE International Parallel and Distributed Processing Symposium*. IEEE, 665–676.

[50] Pradeep Sindhu, Jean-Marc Frailong, Wael Noureddine, Felix A Marti, Deepak Goel, Rajan Goyal, and Bertrand Serlet. 2019. Data processing unit for stream processing. US Patent App. 16/031,945.

[51] Giuseppe Siracusano and Roberto Bifulco. 2017. Is it a SmartNIC or a Key-Value Store? Both! In *Proceedings of the SIGCOMM Posters and Demos*. 138–140.

[52] Craig Ulmer, Shyamali Mukherjee, Gary Templet, Scott Levy, Jay Lofstead, Patrick Widener, Todd Kordenbrock, and Margaret Lawson. 2018. Faodel: Data management for next-generation application workflows. In *Proceedings of the 9th Workshop on Scientific Cloud Computing*. 1–6.

[53] K. D. Underwood, J. Coffman, R. Larsen, K. S. Hemmert, B. W. Barrett, R. Brightwell, and M. Levenhagen. 2011. Enabling Flexible Collective Communication Offload with Triggered Operations. In *2011 IEEE 19th Annual Symposium on High Performance Interconnects*. 35–42. https://doi.org/10.1109/HOTI.2011.15

[54] S Van Doren. 2019. Compute Express Link, Stephen Van Doren (Intel). In *2019 IEEE Symposium on High-Performance Interconnects (HOTI)*. IEEE, 18–18.

[55] A. Wagner, Hyun-Wook Jin, D. K. Panda, and R. Riesen. 2004. NIC-based offload of dynamic user-defined modules for Myrinet clusters. In *2004 IEEE International Conference on Cluster Computing (IEEE Cat. No.04EX935)*. 205–214. https://doi.org/10.1109/CLUSTR.2004.1392618

[56] Felix Zahn and Holger Fröning. 2020. On Network Locality in MPI-Based HPC Applications. In *49th International Conference on Parallel Processing-ICPP*. 1–10.

[57] Judicael A Zounmevo, Xin Zhao, Pavan Balaji, William Gropp, and Ahmad Afsahi. 2014. Nonblocking epochs in MPI one-sided communication. In *SC'14: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE, 475–486.