Enhancing InfiniBand with OpenFlow-Style SDN Capability

Jason Lee, Zhou Tong, Karthik Achalkar, Xin Yuan
Department of Computer Science
Florida State University
Tallahassee, Florida, USA
{jalee,tong,achalkar,xyuan}@cs.fsu.edu

Michael Lang
Computer, Computational, and Stat. Sci. Div.
Los Alamos National Laboratory
Los Alamos, NM, USA
mlang@lanl.gov

***Abstract*—InfiniBand is the *de facto* networking technology for commodity HPC clusters and has been widely deployed. However, most production large-scale InfiniBand clusters use simple routing schemes such as the destination-mod-k routing to route traffic, which may result in degraded communication performance. In this work, we investigate using the OpenFlow-style Software-Defined Networking (SDN) technology to overcome the routing deficiency in InfiniBand. We design an enhanced InfiniBand with OpenFlow-style SDN capability and demonstrate a use case that illustrates how the SDN capability can be exploited in HPC clusters to improve the system and application performance. Finally, we quantify the potential benefits of InfiniBand with OpenFlow-style SDN capability in balancing the network load by simulating job traces from production HPC clusters. The results indicate that InfiniBand with SDN capability can achieve much better network load balancing than traditional InfiniBand for HPC clusters.**

***Keywords*-Software Defined Networking, High Performance Computing, Fat Tree, Simulation, InfiniBand, OpenFlow**

I. Introduction

InfiniBand is an established interconnection networking technology that has been widely deployed in High Performance Computing (HPC) systems. In the November 2015 top 500 Supercomputer list, 235 out of the 500 fastest supercomputers in the world use InfiniBand as their interconnects [1]. As HPC systems continue to increase in size towards exascale, addressing issues with performance, scalability, cost, and resilience poses a major challenge for InfiniBand and other HPC interconnect technologies. A fundamental trade-off among these competing factors must be achieved in order to develop efficient future extreme-scale HPC systems.

The emerging Software-Defined Networking (SDN) technology represents a vantage point in the design space of HPC interconnects that has not been fully explored. SDN has several features that make it attractive for exascale HPC interconnects as it potentially provides the ideal combination of performance, cost, scalability, and resilience for the future extreme-scale HPC systems:

- Performance: SDN allows for dynamic reconfiguration of the network to provide per-flow resource management and routing, which is significantly more flexible than the deterministic routing scheme that is widely employed in the current InfiniBand-based HPC clusters. The ability to manage traffic at the flow level potentially enables network resources to be utilized much more effectively.
- Cost: SDN is designed for Internet and data center applications with large numbers of installments. The economics of scale dictates that SDN technology will be more cost-effective as the technology matures.
- Scalability: The network operations in an SDN are simpler than those in networks with advanced adaptive routing schemes such as the global adaptive routing in the Cray Cascade system [2]. Hence, SDN is more scalable than interconnects with advanced adaptive routing schemes and may strike the ideal balance between the network complexity and capability for future exascale HPC systems.
- Resilience: The flexible system reconfiguration in a SDN facilitates resilience management at the network level, which has become increasingly important as the system size increases.

HPC systems and applications can take advantage of SDN features to maximize their effectiveness. HPC systems commonly observe repeating data workflows such as those for analytics, visualization, and I/O data flows to storage systems or gateway nodes with the data flows sharing many well-utilized traffic patterns. SDN functionality provides ample opportunities both at the application level and the system level to optimize for such workloads by providing custom adaption for the traffic patterns.

Current SDN development is mainly based on OpenFlow [3], which is standardized for Ethernet based networking infrastructure. However, such infrastructure, does not support the low latency communication that many traditional HPC applications require. On the other hand, InfiniBand offers low latency and high bandwidth communication as well as many other features that are attractive to HPC applications. However, InfiniBand does not support the per-flow resource management and routing in an OpenFlow-style SDN; almost all production large-scale InfiniBand clusters use a simple deterministic single-path routing scheme such as the destination-mod-k routing [4], which can result in degraded communication performance. We note that adaptive routing

SC16; Salt Lake City, Utah, USA; November 2016
978-1-4673-8815-3/16/$31.00

has been proposed for InfiniBand [5]. However, we are unaware of any production InfiniBand cluster that utilizes adaptive routing.

In this work, we investigate using the OpenFlow-style SDN technology with per-flow resource management and routing to overcome the routing deficiencies of InfiniBand. We design an enhanced InfiniBand with the per-flow resource management and routing capability, which will be called **SDN-enhanced InfiniBand**. A use case for SDN-enhanced InfiniBand is described where the job scheduler interacts with the SDN controller during job allocation, which allows custom routes to be used for each application. The use case illustrates how the SDN capability can be exploited in HPC clusters to improve system and application performance. We further quantify the potential benefits of SDN-enhanced InfiniBand in improving network load balancing by simulating job traces from production HPC clusters. The results indicate that the proposed SDN-enhanced InfiniBand can achieve much better network load balancing in comparison to conventional InfiniBand.

The rest of the paper is structured as follows: the background is discussed in Section II; Section III introduces the design of SDN-enhanced InfiniBand; Section IV describes a use case and reports the results of our study on the potential benefits of SDN-enhanced InfiniBand. Section V presents related work; and Section VI concludes the paper.

II. Background

A. SDN and OpenFlow

The key idea of SDN is to separate the network control plane from the network data plane. This decoupling allows for the network control (e.g. routing) to be performed by third-party software independent of data-forwarding equipment vendors. SDN supports layers of abstractions for the network control, and promises various degrees of network flexibility and scalability including:

- High-level virtual representation of networks
- Scalable architecture that provides flexible routing at the flow level
- The ability to add new network features via open, industry-standard interfaces

The structure of SDNs [6] is depicted in Figure 1. There are three layers in an SDN: the infrastructure layer, the control layer, and the application layer. The infrastructure layer consists of network elements that perform the simple data plane function of packet forwarding. At the control layer, the SDN controller controls and interacts with network elements through the *SDN southbound interface*. The *SDN northbound interface* above the SDN controller interacts with the SDN applications that determine the behavior of the SDN.

OpenFlow [3] is the enabling technology for SDN. It is a realization of the SDN southbound interface for Ethernet

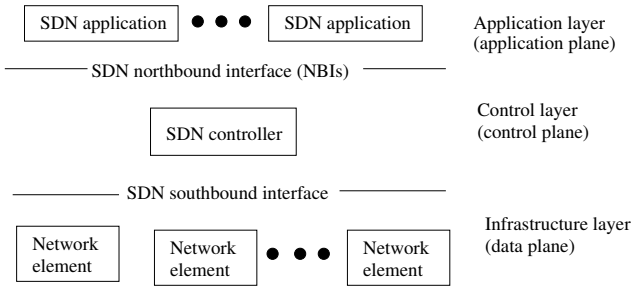

Figure 1. SDN abstraction

and TCP/IP-based networking infrastructure that defines the interface between network elements and the SDN controller. OpenFlow specifies the protocols and packet formats for the SDN controller to control the network elements and for network elements to report their status to the SDN controller. This allows the SDN controller to obtain the global view of the network, therefore providing per-flow resource management and routing for individual flows. In addition, OpenFlow also specifies the necessary functionality in the network elements to support the OpenFlow standard.

The most remarkable new function in an OpenFlow-style SDN is the dynamic per-flow resource management and routing using the global view of the network. An OpenFlow switch maintains a flow table that can be dynamically updated to change the packet forwarding behavior on-the-fly. This, coupled with the global network view, presents significant opportunities for optimizing network resource utilization and efficiency.

To support diverse Internet applications, the "flow" concept in OpenFlow is very generic. A flow can be defined by any bit pattern in headers of typical Internet packets including the source and destination addresses (IP and Ethernet), source port, destination port, protocol and some other Internet packet header fields. Communication in HPC applications is likely to have fewer varieties than that in Internet applications. Therefore, the flow concept for SDN-enhanced HPC interconnects should be much simpler than those defined in OpenFlow.

B. InfiniBand and SDN

InfiniBand is an open standard interconnect specification developed by the InfiniBand Trade Association [7] and is currently the dominating networking technology for high-end commodity HPC clusters. InfiniBand was designed with many features that make it very attractive for HPC interconnects. InfiniBand provides high bandwidth (up to 300Gb/s with 12xEDR) while having low latency (700 nanoseconds end-to-end latency [8]) which is desired by many HPC applications. InfiniBand supports functionality that closely matches the requirements of HPC applications, including remote direct memory access (RDMA), multicasting, and operating system kernel bypassing. Additionally, InfiniBand

was designed to scale to thousands of nodes, making both small networks that desire high speed connections as well as large networks, such as those within supercomputers, possible with the same technology.

The current InfiniBand standard already supports some SDN functionality [9]. In particular, the InfiniBand standard [7] requires each InfiniBand subnet to have a centralized controller called the Subnet Manager (SM), which is responsible for the overall operation of the subnet. The SM's tasks include collecting the network topology information, computing routes, and setting up forwarding tables in the network elements. Each network element in an InfiniBand subnet, such as a switch, is required to have a control plane agent called *Subnet Management Agent* (SMA). The SMA allows a network element such as a switch to report its status to the SM and to perform the required actions from the subnet manager, such as adding an entry in its forwarding table. SM and SMA communicate with the Subnet Management Protocol using Subnet Management Packets (SMPs), which is a special type of InfiniBand Management Datagrams. Hence, the SM in an InfiniBand network performs the functionality of the network controller and the network application in an SDN: the SM has the global view of the network and oversees the operation in the network. The InfiniBand protocol governing the interaction between SM and SMAs is similar to OpenFlow in an SDN that implements the southbound interface.

With these mechanisms, the network abstraction in SDN can be supported in the current InfiniBand [7]. Figure 2 shows how the SDN network abstraction in Figure 1 can be mapped to InfiniBand: InfiniBand switches correspond to network elements in the infrastructure layer in an SDN. The InfiniBand SM corresponds to the combined SDN controller and SDN application. The InfiniBand subnet management protocol corresponds to the SDN southbound interface. Note that InfiniBand also has more sophisticated controllers such as the InfiniBand virtual network controller. Such entities can be considered as an SDN application that is built over the basic network abstraction provided by the subnet manager.

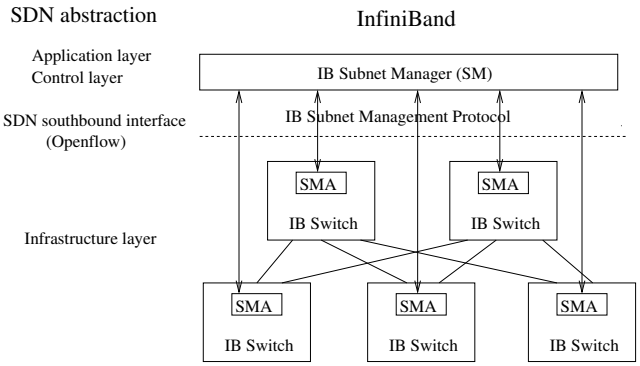

Figure 2. SDN abstraction mapping to InfiniBand (IB)

C. Routing in SDN and InfiniBand

While InfiniBand provides some SDN functionality as discussed, it does not support the per-flow resource management in an OpenFlow-style SDN. As mentioned earlier, production InfiniBand clusters mostly use a simple deterministic single path routing scheme such as the destination-mod-k (DmodK) routing. Such a routing scheme cannot balance the network traffic and often result in degraded communication performance for some common HPC traffic patterns [10], [11]. SDN on the other hand is able to provide globally optimal routes through routing with a global view of the interconnect. By directing traffic based on the global network view, routes that achieve better overall load balancing can be computed in an SDN. This paper focuses on the potential benefits of SDN routing over the deterministic single path routing in InfiniBand.

Adaptive routing has also been proposed for InfiniBand [5]. To be scalable, such an scheme adapts traffic based on local traffic conditions, which results in local optimizations. Moreover, deploying adaptive routing in InfiniBand requires the software communication stack to be updated. Although adaptive routing in InfiniBand has been proposed for many years, due to the difficulties, we are not aware of any production InfiniBand cluster that utilizes adaptive routing. We do not consider InfiniBand adaptive routing in this paper.

D. Related InfiniBand Concepts

The design of SDN-enhanced InfiniBand minimizes the modifications to the current InfiniBand standard. SDN-enhanced InfiniBand utilizes some header fields of the data packets and the Management Datagrams. Next, we will briefly introduce the related InfiniBand concepts. Readers should refer to the InfiniBand standard [7] for more details.

Figure 3 depicts the InfiniBand data packet format. An InfiniBand data packet contains an 8-byte Local Routing Header (LRH), an optional 40-byte Global Routing Header (GRH), a 12-byte Base Transport Headers (BTH), one or more optional variable sized extended Transport Headers, an optional 4-byte immediate data or R_Key field, the message payload, and a 4-byte invariant CRC and 4-byte variant CRC. Our SDN-enhanced InfiniBand will perform operations based on select fields in the Local Routing Header (LRH) and Base Transport Header (BTH), which are both present in all InfiniBand packets. Among the fields in LRH: our design uses the `Service Level` (SL) field that is used to determine the Quality-of-Service level of the packet, `Destination Local ID` (DLID), and `Source Local ID` (SLID). The related fields in BTH used in this work include `OpCode` that determines the packet type, the `Destination QP` (DestQP) that can be used to identify a flow, and `Packet Sequence Number` (PSN).

SDN-enhanced InfiniBand introduces a new class of InfiniBand Management Datagrams (MADs) that is used for management tasks and realizing management protocols

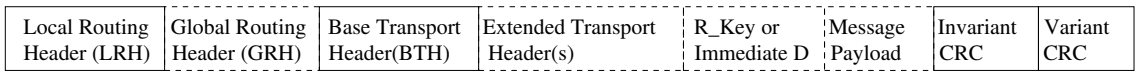

Figure 3. InfiniBand data packet format

such as the subnet management protocol in InfiniBand [7]. The format of the InfiniBand MAD is shown in Figure 4. The subnet management reporting and controlling data are carried in the data field in the MAD. The SM and SMA are required to support a number of MAD classes as defined in the InfiniBand specification [7].

III. The Design of SDN-Enhanced InfiniBand

SDN-enhanced InfiniBand incorporates the dynamic per-flow resource management and routing capability into the existing InfiniBand. To support the dynamic per-flow control capability in InfiniBand, related components in the InfiniBand control plane must be modified and/or enhanced. The components include: InfiniBand switches, InfiniBand subnet management entities (subnet manager and subnet management agents) and protocols, and InfiniBand data packet processing logic. Our main design objective is to support dynamic per-flow routing while minimizing the modifications to the current InfiniBand standard. Next, we will discuss important design choices and describe the modifications in InfiniBand components in order to support dynamic per-flow resource management.

A. Flow Table and Pre-establishment of Flow Table Entries

In an OpenFlow network, a packet will be matched against the flow entries in the flow tables in a switch. If a match is found, the actions specified in the flow table will be performed to process the packet. If a match is not found, the packet is sent to the controller, which will decide how to handle the packet. In the current InfiniBand network, packets are routed by a static forwarding table that is indexed by the DLID field in the LRH header to decide an output port. To support the dynamic per-flow resource management capability, a flow table with similar functionality as that of an OpenFlow switch must be added to each InfiniBand switch in our SDN-enhanced InfiniBand so that flow specific actions can be applied to packets belonging to different flows.

byte	**bits 31-24**	**bits 23-16**	**bits 15-8**	**bits 7-0**	
0	BaseVersion (1)	MgmtClass (0x09)	ClassVersion (1)	R (0/1)	Method (0)
4	Status (0)		ClassSpecific (0)		
8	TransactionID (#)				
12					
16	AttributeID (0)		Reserved (0)		
20	AttributeModifier (0)				
24	Data (OpenFlow control packet)				
...					
252					

Figure 4. InfiniBand Management Datagram format and the values for the new OpenFlow class in SDN-enhanced InfiniBand (values in parentheses)

One important design choice is the timing when the flow table entries should be populated. In an OpenFlow network, a flow table entry may be reactively installed by the controller after a flow table miss. For a typical HPC application, this would be very undesirable since a flow-table miss will result in a packet latency that is orders of magnitude larger than without a flow-table miss. Moreover, such penalty will be propagated through all processes due to the relatively frequent synchronization operations in the HPC application. To alleviate this problem, the proposed SDN-enhanced InfiniBand maintains the forwarding table in the current InfiniBand and uses it to realize default routes for packets with flow-table misses. In SDN-enabled InfiniBand, flow table entries for a flow are pre-established. Due to the capability to provide default routes through the forwarding table, our SDN-enabled InfiniBand removes the necessity to reactively setup the flow table entry when a flow table miss occurs.

The packet processing operations are as follows: a packet will be matched against flow entries in the flow table in the switch. If a match is found, the actions specified in the flow table will be performed to process the packet. If a match is not found, the default route determined by the forwarding table will be used. This avoids the potential significant performance penalty for HPC applications when flow table misses occur. HPC systems/applications may utilize SDN-enhanced InfiniBand as the traditional InfiniBand by not using flow tables. To utilize the per-flow resource management functionality, HPC systems and applications can pre-establish the flow-table entries at the job allocation time or during the execution of the application before the communication happens.

In summary, each switch in an SDN-enhanced InfiniBand network will be equipped with a flow table that is similar to the flow table in an OpenFlow switch in addition to the forwarding table in the traditional InfiniBand switch. The forwarding table will support default routes for all packets while more optimized per-flow routes will be realized by the flow table, whose entries are pre-established (e.g. at job allocation time) before the packets that utilize the flow table entries are communicated. Note that many HPC applications have phased behavior with communication and computation alternating during program execution. The flow table can also be set-up before each communication phase. Note also that the forwarding tables will also be required for the InfiniBand initialization and for routing management packets.

Although requiring flow table entries to be pre-established

reduces the per-flow resource management flexibility in comparison to the OpenFlow-based SDN, it exposes the per-flow resource management capability to each parallel application and/or the system, allowing for customized routing mechanisms to be used for each application or even each communication phase within an application. Studies have shown that using application specific routing scheme can significantly improve the performance of the communication infrastructure for HPC applications over the widely-deployed system-wide deterministic single-path routing in the current InfiniBand [12], [13].

Our design also addresses another important problem that exists in the current OpenFlow-based SDN: the limited size of a flow table. With the current technology, the size of a flow table is limited to a few thousand entries [14]. As such, flow tables will not be able to support all communications in large-sized networks. By having flow tables in addition to the forwarding table, our design allows flow tables to be used to direct traffic for the most important flows while other packets can follow default routes.

B. The Flow Concept

Our SDN-enhanced InfiniBand does not require any modification to InfiniBand data packets. The "flow" in the SDN-enhanced InfiniBand is defined using the existing header fields in the current InfiniBand data packets. Depending on the level of control that the SDN-enhanced InfiniBand would provide, a selection of potential choices with increasing complexity is described below. The selection of the flow definition is a design choice, depending on the functionality required as well as the hardware constraints. The terminology used is described in Section II-D and [7].

- A flow may be defined by the bit pattern in the DLID field. In this case, the flow-table functionality will be similar to the forwarding table. However, application specific routing is enabled.
- A flow may be defined by the bit pattern in the DLID and SLID fields. This allows per-application routes based on both source ID and destination ID to be used in the system.
- A flow may be defined by the bit pattern in the DLID, SLID, and the SL (service level) field. This allows per application routes based on source ID, destination ID, and service level to be used in the system.
- A flow may be defined by bit patterns in the DLID, SLID, SL, and DestQP fields. This allows different communications within an application between the same source and destination nodes to follow different routes.
- A flow may be defined by bit patterns in the DLID, SLID, SL, DestQP, PSN fields. This allows different packets for the same message to follow different routes.

C. Switches in SDN-enabled InfiniBand

To support per-flow resource management functionality, enhancements must be made to InfiniBand switches. Figure 5 shows the block diagram of an InfiniBand SDN switch and its interaction with the enhanced SM in SDN-enabled InfiniBand. The switch distinguishes between data packets and control packets. The enhancement is only used to process data packets. The processing of control packets is the same as the current InfiniBand. First, a flow table similar to that of an OpenFlow switch is added that will match flow entries and perform the specified actions. Second, the packet processing logic must be modified such that if a packet matches a flow in the flow table, the actions specified in the flow table will be performed on the packet. If the packet fails to match any flow such as with traditional InfiniBand or jobs with dynamic traffic patterns, the routing will be determined by the forwarding table. Third, the SMA software must be enhanced with the added functionality to understand the enhanced subnet management protocol. In addition to performing the traditional subnet management protocol that mainly relates to managing the forwarding table, the enhanced subnet management protocol also performs OpenFlow-like functions that relate to managing the flow table.

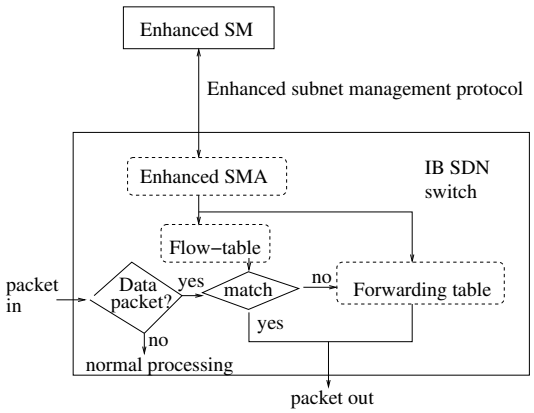

Figure 5. InfiniBand SDN switch

D. Control Packet Modification

For the purposes of integrating per-flow resource management into InfiniBand, a new class of subnet management packet is defined by using one of the InfiniBand vendor values for OpenFlow. The current InfiniBand specification defines several classes of management packets, such as the subnet management classes and the performance management class [7]. The new class assumes that the forwarding tables have been setup and is routed based on DLID. Despite functionality of the new MAD class being very similar to the LID routed MAD class [7], creating a new class was chosen over using the LID routed MAD class since the LID routed

MAD class has already been defined for other purposes. A new type allows for more flexibility in the values used in each field, rather than having to work around existing values.

The OpenFlow class of MAD packets also preserves the formats of both InfiniBand and OpenFlow control packet structures. The 256 byte InfiniBand MAD packet has a 232 byte data field, of which the first 56 bytes will be used by the fixed fields of the OpenFlow structure `ofp_flow_mod`. The remaining 176 bytes will be used for the variable length OpenFlow `ofp_match` and `ofp_instruction_header` structures (refer to [3] for more details of the OpenFlow structures). This space should be sufficient for OpenFlow matches and actions, since InfiniBand has significantly fewer definitions for flows than Ethernet does. While this may not be the most effective use of space, it allows for our design to keep up with the development of OpenFlow and potentially reuse software developed for OpenFlow in our SDN-enhanced InfiniBand.

Figure 4 shows the new class of management packets that is created for SDN-enhanced InfiniBand. The value in the parenthesis in each field is the value set for the new OpenFlow class of management packets. Among the header fields are the BaseVersion with a value of 1, MgmtClass with a value of $0x09$, ClassVersion with a value of 1, and TransactionID with a variable value. The BaseVersion and ClassVersion fields are required for the current version of MAD packets. The MgmtClass value identifies the packet class. The value $0x09$ is the first value available to developers to add new functionality, which we use for the new OpenFlow class. The response bit, R, may have a value of 0 or 1, depending on whether the OpenFlow control packet requires a response. The Transaction ID field will be generated from LRH and BTH fields to identify the transaction. Finally, the data field will contain the whole OpenFlow control packet.

The existing InfiniBand already has the capability to collect network state information to the SM. With our design, the flow-table entries are pre-established. Thus, we only need OpenFlow control packets to allow the SM to manipulate the flow table entries in switches within its domain. In the OpenFlow standard, such packets are `ofp_flow_mod` packets [3]. Since SDN-enhanced InfiniBand only uses a subset of control functions in OpenFlow, not all fields in the standard Openflow control packets are necessary. The most important field used is the match field that specifies flows and the corresponding actions.

E. Enhanced Subnet Manager to Control the New Switches

The main change to the subnet manager (SM) software (e.g. OpenSM [15]) will be the additional functionality to handle OpenFlow operations, performing the functions that are performed in OpenFlow controllers. Since the existing InfiniBand already obtains the network topology information, the additional functionality mainly includes the following:

- Maintaining global information on jobs and the network state.
- Interacting with SDN applications and switches, computing the flow table entries for each application, and setting up flow table entries on each switch.

These are a subset of the functionalities supported by a typical OpenFlow controller. There are two options to implement these functionalities. The first option is to integrate OpenFlow controller operations into the SM. The SM will send MAD packets to set up the flow tables in the switches under its control.

The other option is to separate the OpenFlow controller functions from the InfiniBand SM. In this case, a separate entity will implement the OpenFlow controller functions and will interact with SM to provide the functions. This would require the design of a protocol for the SM to communicate with the SDN controller. This method may allow SDN-enhanced InfiniBand to directly interact with peering OpenFlow-based SDN.

IV. A USE CASE AND PERFORMANCE STUDY

Here we describe a use case of SDN-enhanced InfiniBand. The use case illustrates the operations in a cluster with SDN-enhanced InfiniBand and shows how OpenFlow-style per-flow resource management capability can be exploited to achieve high communication performance.

Consider an HPC cluster whose interconnect has a fat-tree topology with our proposed SDN-enhanced InfiniBand. Let a flow be defined by SLID and DLID, which allows a per-flow routing to be decided by the source and the destination. To ease exposition, we will assume that the SM and the SDN controller are logically separated. Additionally, we assume that the cluster uses an augmented version of a job scheduler, such as the widely used SLURM scheduler with the tree plugin [16] to allocate compute nodes for jobs. The scheduler is slightly enhanced so that it can interact with the SDN controller by informing the SDN controller about the job to be allocated and deallocated, and the logical communication patterns of the jobs, and starting a job only after the interconnect has been configured for the job. The SDN controller allocates paths for all of the communications of a new job and informs the scheduler that the network is ready for the job. The SDN controller deallocates network resources after a job completes.

We will assume that the logical pattern for important communications for a job is known. Note that this is not an unreasonable assumption: studies have shown that the logical communications of most scientific applications are easily analyzable [12], [13]. In addition, for applications with unknown communication patterns, one can either use the default routes or assume the worst-case all-to-all logical communication for all processes of a job. Also, since SDN-enhanced InfiniBand provides default routes, the logical

communication pattern does not have to support all communications, but rather just the important ones.

Since the SDN controller maintains the global network link usage, the cluster can provide custom routing for each application that optimizes for network load balancing. The SDN controller can employ a simple heuristic to allocate the path for each communication in a job by selecting the least loaded path among all possible shortest paths for each application or even each communication phase within an application. Our evaluation with job traces from production HPC clusters indicates that with this simple heuristic, the network load balancing is greatly improved over the widely-used DmodK routing for fat-trees in the current InfiniBand.

Let us now turn to the system operations. For system initialization, SDN-enhanced InfiniBand keeps the forwarding table and does not modify the initialization sequence. The SM collects the topology information, allocates LIDs for each device, computes the paths (e.g. using the DmodK routing) for each source-destination pair, and installs the forwarding table entries to realize the routes. By setting up forwarding table entries, the default routes for the SDN-enhanced InfiniBand are established.

Once the system is initialized, job requests can be submitted and processed by the augmented job scheduler. Figure 6 shows the operations in the use case. When a job request arrives, the scheduler will allocate a set of compute nodes to the job. The scheduler informs the SDN controller about the job allocation and communication pattern, which at this time can be mapped to a physical communication pattern. The SDN controller then computes a custom route for each communication in the pattern. The custom routing is job specific. The computation of the custom routing takes into consideration various constraints such as the availability of flow table entries in each switch and optimizes load balancing in the network. Once the paths are determined, the SDN controller then computes the flow-table entries that realize the allocated custom paths for the job. The SDN controller sets up the flow table entries in a switch by sending OpenFlow MAD packets to switches. When an OpenFlow MAD packet is received by a switch, the switch extracts the entry and places it into the flow table. After all of the flow table entries are established for a job, the SDN controller can inform the job scheduler that the job can now be launched.

During the job execution, messages are split into packets. When a packet enters a switch, it will be matched against the entries in the flow table. Should the packet match with an entry, the switch processes the packet according to the entry's actions. If the packet does not match with any entry, it will be routed through the default path.

When a job finishes execution, the scheduler informs the SDN controller the completion of the job. If custom routing was established for the job, the SDN controller issues commands to remove the associated flow table entries in the affected switches, and update the flow table status.

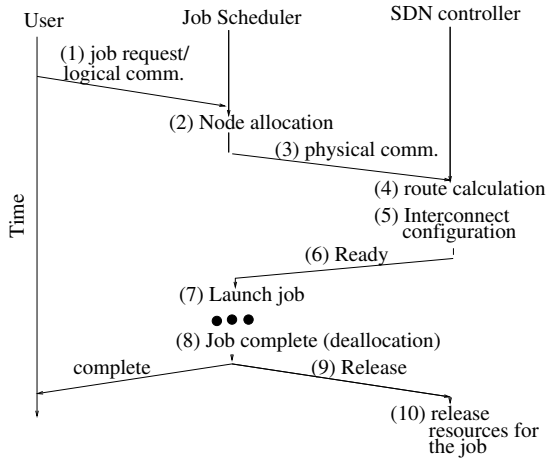

Figure 6. Operations in the use case

A. Performance Evaluation

In this sub-section, we evaluate the potential benefit of using SDN-enhanced InfiniBand in the setting described in the use case by simulating the job traces from production HPC clusters. Load balancing metrics are compared between systems with traditional and SDN-enhanced InfiniBand. The results indicate that SDN-enhanced InfiniBand achieves significantly better network load balancing. Next, we will describe the settings and methodology for the performance comparison, and report the results.

1) Job Traces: The job traces that we used in our study are maintained in the Parallel Workloads Archive [17]. They are in the Standard Workload Format (SWF) [18]. Each entry of a trace contains 18 fields, but we only use the job id, submit time, wait time, run time, number of allocated nodes, and status fields.

We used three job traces from production HPC systems in our study: CEA[1] Curie [19], LLNL[2] Thunder [20], and LLNL Atlas [21]. The traces have different workload characteristics. The Curie supercomputer has general-purpose scientific computing workloads [19]. Thunder was designated as a "capacity" cluster for small to medium jobs [20], while Atlas was a "capability" cluster [21] for larger jobs. Table I lists the general information for the traces and the systems, including the number of compute nodes, the number of CPUs per node, the number of jobs, the duration of the trace logs, and the average utilization of the system. The original traces had been filtered by Feitelson [22], [23]. In our simulation, we further remove uninteresting jobs that had run times of 0, were canceled while waiting in the queue (status value of -1), or requested more than the number of nodes in the system. In this study, we choose to present

[1]Commissariat à l'énergie atomique et aux énergies alternatives
[2]Lawrence Livermore National Laboratory

the results for the first 1000 jobs from our filtered traces, because they are representative for the full-trace simulations runs. These 1000 jobs spans over 5 to 16 days, depending on the trace.

Machine	CEA Curie	LLNL Thunder	LLNL Atlas
Nodes	5904	1024	1152
CPUs/Node	2	4	8
Number of Jobs	773,138	128,662	60,332
Time (months)	20	5	8
Utilization (%)	29.3	87.9	64.1

Table I
GENERAL INFORMATION ABOUT THE JOB TRACES AND THE SUPERCOMPUTERS THAT THE JOB TRACES WERE COLLECTED

2) Interconnect Topology and Routing: We assume that the interconnect topologies are fat-trees. More specifically, for the CEA Curie trace with 5904 compute nodes, we set the topology as a 30-port 3-level full bisection bandwidth extended generalized fat-tree, XGFT(3; 15, 15, 30; 1, 15, 15) using the notation in [4]. This tree supports $15*15*30=6750$ compute nodes, of which, the first 5904 nodes are used. For LLNL Thunder, the topology is set to be a 16-port 3-level full bisection bandwidth extended generalized fat-tree, XGFT(3; 8, 8, 16; 1, 8, 8), supporting $8*8*16=1024$ nodes, the exact number of nodes present in Thunder. For Atlas, the topology is a 18-port 3-level full bisection bandwidth extended generalized fat-tree, XGFT(3; 9, 9, 18; 1, 9, 9) with $9*9*18=1458$ compute nodes. The first 1152 continuous compute nodes are used for simulation, the other ports are unused.

For traditional InfiniBand, we assume DmodK routing is used [4]. For SDN-enhanced InfiniBand, the routes are allocated as follows: we assume that the important logical communication within each job is known. Each logical communication within a job is assigned a weight for the load that it introduces. The SDN controller maintains the usage weight for each link in the network. When allocating a path for a logical communication in a job, the SDN controller selects the path with the smallest maximum weight along its links from among all possible shortest paths. The SDN controller updates the global network state when the link usage state changes.

3) Job Allocation: Both traditional InfiniBand and SDN-enhanced InfiniBand are assumed to use the same SLURM scheduler with the tree plugin [16]. In this job scheduling scheme, compute nodes are selected to be as contiguous as possible to minimize the interference among jobs. More specifically, compute nodes are allocated by first identifying the nearest common ancestor switch that can satisfy a job request. After that, the best-fit algorithm is used to allocate compute nodes of the underlying leaf switches beneath the selected nearest common ancestor switch, while minimizing the number of segments used and fragments created.

4) Logical Communication in Each Job: Let N be the number of ranks in a job. To quantify network load balancing, the communication information of each job is needed. Studies have shown that the vast majority of HPC applications that run at scale have low-dimension stencil patterns [24], [25] such as 2-dimension nearest neighbor (2DNN). Another important class of HPC application has irregular communication patterns. To mimic these HPC communication workloads, our study considers the following patterns:

- Ring: In the ring pattern, process i communicates with processes $i+1$ and $i-1$ with wrap around.
- 2-dimension nearest neighbor (2DNN): In this pattern, we first generate a 2D $\lceil\sqrt{N}\rceil \times \lceil\sqrt{N}\rceil$ grid. If $N < \lceil\sqrt{N}\rceil \times \lceil\sqrt{N}\rceil$, there may have some un-occupied points in the last row of the grid. Process (i,j) communicates with four neighbor processes in the grid $((i-1,j)$, $(i+1,j)$, $(i,j-1)$, and $(i,j+1))$ with wrap around.
- 3-dimension nearest neighbor (3DNN): In this pattern, we first generate a 3D $\lceil\sqrt[3]{N}\rceil \times \lceil\sqrt[3]{N}\rceil \times \lceil\sqrt[3]{N}\rceil$ grid. If $N < \lceil\sqrt[3]{(N)}\rceil \times \lceil\sqrt[3]{(N)}\rceil \times \lceil\sqrt[3]{(N)}\rceil$, there may have some un-occupied points in the last plane of the grid. Each process communicates with six neighbor processes in the grid with wrap around.
- Random4 pattern: each process communicates with 4 other randomly selected processes.
- Random8 pattern: each process communicates with 8 other randomly selected processes.

In some of the experiments, we assume that all jobs have the same logical communication pattern (one of the above). In others, we consider situations when each job can have a different pattern selected randomly from the set of patterns (dynamic).

5) Performance metrics: The network load balancing is quantified with two metrics: *per job maximum load* (PJML) and *system wide maximum load* (SWML). At the high level, per job maximum load (PJML) is defined as the maximum link load that a job experiences during its execution while the system wide maximum load (SWML) is the maximum link load in the system at a given time. More specifically, the PJML for each job and SWML at a given time is computed by first estimating the link load of each communication from process s to process d in the logical communication pattern for the job, which is then mapped to physical nodes allocated. For each communication (s,d), we assume that the source node s will introduce at most 1 unit of traffic among all communications in the job whose source is s. Similarly, each destination node d can receive at most 1 unit of traffic among all communications whose destination is d. Let s_{out} be the number of outgoing communications from source s and d_{in} be the number of incoming communications to destination d in the communications for the job. Communication (s,d) will be assigned a weight

$$w_{(s,d)} = \min(\frac{1}{s_{out}}, \frac{1}{d_{in}})$$

During the job execution, this amount ($w_{(s,d)}$) of communication is assumed to occur in every link along the path from s to d. Based on this, the traffic incurred by each job and the whole system is simulated. SWML and PJML captures both intra-job and inter-job network contention.

6) Simulation: Each job trace provides the start and end time for each job. During the simulation, the simulator maintains the traffic load on all links. When a job starts, the traffic load for that job is added to the links used by the job, and is removed when the job finishes. Furthermore, the simulator monitors and updates the PJML for each job and the SWML for the whole system. Note that the network state changes only when a job starts or finishes. Hence, to simulate M jobs, only $2M$ updates to the SWML are needed.

Figure 7 shows the pseudo-code of our simulation to compute the performance statistics for SDN-enhanced InfiniBand. For traditional InfiniBand systems, it can be computed by replacing the SDN routing with the DmodK routing. The simulator takes three sets of input parameters: the job trace file, the system topology, and the type of logical communication pattern for the jobs in the given trace. The type of logical communication parameter may be one of the types listed earlier (*Ring*, *2DNN*, *3DNN*, *Random4*, and *Random8*). In this case, all jobs have the same logical communication pattern. The parameter may also be *dynamic*, in which case, each job uniform-randomly selects its communication pattern among Ring, 2DNN, 3DNN, and Random4.

The simulator reads through each job record in the job trace and creates two entries: a job start entry and a job finish entry. A job entry contains the job id, the number of allocated nodes, and a timestamp. The job start entry timestamp indicates the start time, which is the sum of the submit time and the wait time. The job finish entry timestamp represents the corresponding start time plus the job's run time. The entries are sorted by the timestamps for simulation. For a job start entry, the simulator computes the node allocation, generates the logical communications for the job, converts the logical communications to physical communications, decides the path for each communication, and adds load to the links in the path (lines 7 to 11). After the network load for this job is added to the network, the simulator updates the PJML for each active job and recomputes the SWML. For a job finish entry, the simulator removes the network traffic for the job that finished, records the PJMLs and recomputes SWML. The simulation completes when all jobs are simulated or the number of jobs simulated reaches a target.

For each trace, we simulate and record the SWML over the duration of all of the jobs and the PJML for each job over six patterns (Ring, 2DNN, 3DNN, Random4, Random8, and dynamic). The simulation is done using TopSim [26],

```
Data: Job trace, Interconnect topology(XGFT), type of
      logical communication pattern
Result: SWML during the trace time, PJML for each job
1  while (job trace is not empty) do
2      Get an entry from the job trace;
3      if (the entry is job start entry) then
4          Run job allocation algorithm to determine its
           compute node allocation;
5          Generate the logical communication pattern for this
           job;
6          Convert logical communications to physical
           communications;
7          for each communication in the job do
8              Compute the weight for the communication;
9              Compute the SDN route according to the current
               network usage state;
10             Add the weight for the communication to each
               link along the route;
11         end
12         Recompute the updated PJML for each active job;
13         Recompute the SWML, record the time and change;
14     end
15     if (the entry is job finish entry) then
16         Remove the traffic for this job from the network;
17         Record PJML for this job;
18         Recompute and update the SWML, record the time
           and change;
19     end
20 end
```

Figure 7. Computing SWML and PJML for SDN-enhanced InfiniBand

a topology simulator developed jointly by Los Alamos National Laboratory and FSU. Next, we will show two representative results for the first 1000 jobs in each trace. Other results as well as the plots for the whole traces have a similar trend.

7) Simulation Results: Figure 8 shows the SWML for the Curie trace assuming all jobs have the 3DNN logical communication pattern. Figure 9 shows the results of the dynamic communication pattern (each job randomly selects a pattern from Ring, 2DNN, 3DNN, Random4). Machines under the dynamic traffic pattern are more prone to network congestion due to the randomness of the communication pattern. Using the current InfiniBand with the system-wide DmodK routing scheme, the maximum SWML value goes up to 1.67 for the 3DNN pattern and 2.80 for the dynamic pattern. Using the SDN-enhanced InfiniBand with application specific routing that optimizes load balancing, the SWML is consistently close to 1 throughout the simulation. The one day gap in the Curie trace was a result of system upgrades.

Figure 10 and Figure 11 show the results for 3DNN and dynamic patterns for the LLNL Thunder trace, respectively. The DmodK routing results in up to 1.95 SWML for the 3DNN pattern and 2.47 for dynamic patterns while SDN-enhanced InfiniBand significantly reduces the SWML value to 1.00 for 3DNN and 1.20 for dynamic. This trace has a two

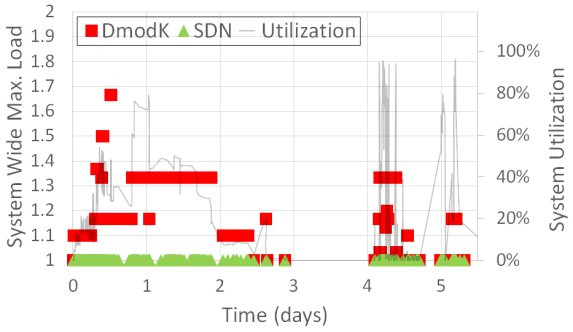

Figure 8. Curie, 3DNN

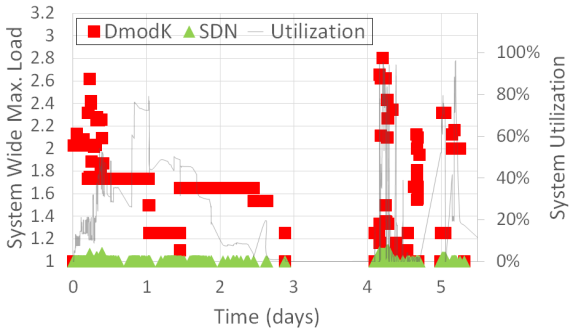

Figure 9. Curie, Dynamic

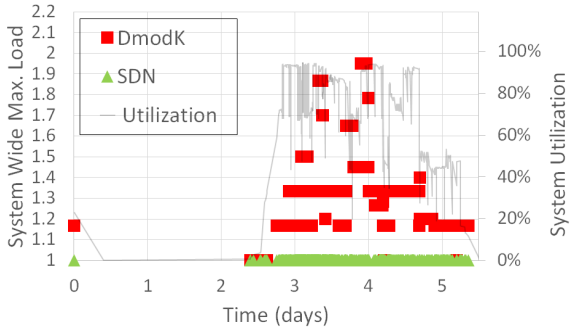

Figure 10. Thunder, 3DNN

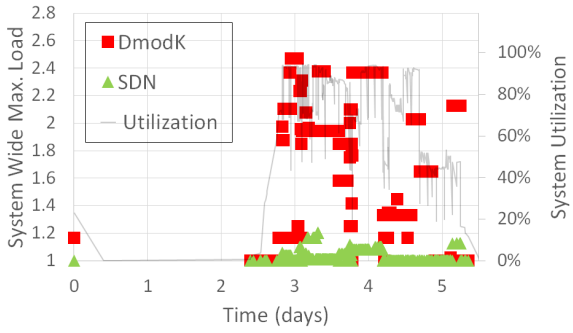

Figure 11. Thunder, Dynamic

day period where the utilization is 0% for unknown reasons. Additionally, there are many jobs requesting a single node in the trace, which did not contribute to the SWML value.

Figure 12 and Figure 13 illustrate that SDN routing continues to outperform DmodK routing. DmodK routing resulted in SWML of 1.83 with the 3DNN pattern and 2.86 for the dynamic pattern. SDN routing resulted in SWML of 1.00 for the 3DNN pattern and 1.17 for the dynamic pattern. Since Atlas was designed to run large jobs [21], resulting in bursts of high utilization.

Table II presents the average and maximum PJML of the first 1,000 jobs in the Curie, Thunder, and Atlas traces. The maximum PJML is equivalent to the SWML as both capture the maximum link load observed among all jobs. We define the δ as the percentage difference between the DmodK and SDN routing in terms of PJML. For the 3DNN pattern, the δ of the average PJML with DmodK and SDN-enhanced routing ranges from 2.1% to 8.0% while the maximum PJML rises significantly to 95.0% for Thunder. Similarly for the dynamic pattern, the δ of the average PJML with DmodK and SDN-enhanced routing ranges from 6.7% to 20.0% while the maximum SWMLs reaches 154.5% for Curie. From the table, it is clear that SDN-enhanced routing results in PJML values that are lower than those of conventional InfiniBand. Note that average PJML can be less than 1 because the traces have many jobs that request a single node, resulting in a PJML value of 0.

These results demonstrate that SDN-enhanced InfiniBand improves load balancing across different HPC workloads. Per-flow resource management is an effective mechanism for improving the communication performance in HPC clusters.

V. Related Work

Since the concepts of SDN and OpenFlow have been introduced, SDN has been widely accepted in industry and the research community. Extensive research and development has been carried out in this area. Most results, however, are not in the HPC domain. The HPC community has also started to explore SDN and OpenFlow capabilities for effective MPI communications through new MPI libraries that can take advantage of SDN capabilities [27]. Arap [28] investigated techniques to explore SDN capability for

Trace	Metric	3DNN pattern			Dynamic pattern		
		DmodK	SDN	δ (%)	DmodK	SDN	δ (%)
Curie	Avg. PJML	0.96	0.94	2.1	1.04	0.94	10.6
	Max. PJML	1.67	1.00	67.0	2.80	1.10	154.5
Thunder	Avg. PJML	0.93	0.90	3.3	0.96	0.90	6.7
	Max. PJML	1.95	1.00	95.0	2.47	1.20	105.8
Atlas	Avg. PJML	0.95	0.88	8.0	1.07	0.89	20.0
	Max. PJML	1.83	1.00	83.0	2.86	1.17	144.4

Table II
Per Job Maximum Load (PJML) for the traces

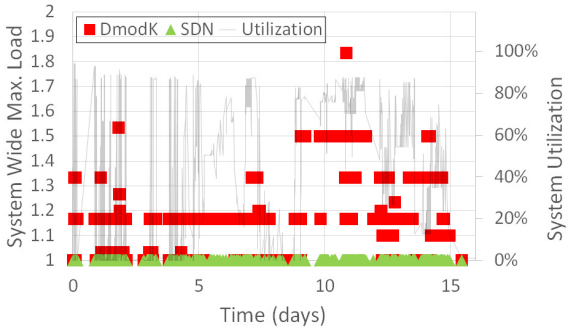

Figure 12. Atlas, 3DNN

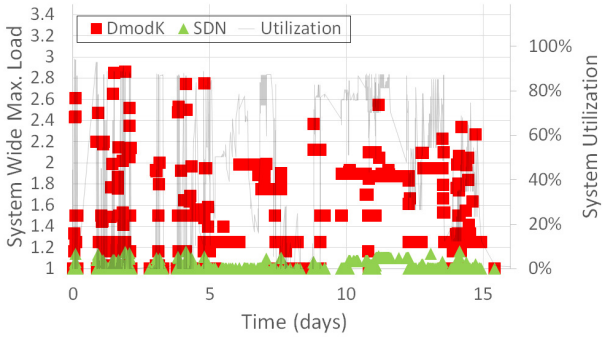

Figure 13. Atlas, Dynamic

efficient MPI collective communications; Takahashi [29] evaluated the performance of MPI-allreduce operation on an SDN cluster. Similarly, Dashdavaa [30] implemented and evaluated MPI_bcast with SDN enhancements. In industry, Mellanox has produced switches that can be configured as either InfiniBand switches or OpenFlow switches [31]. However, such a switch does not provide OpenFlow functionality when it is configured as an InfiniBand switch. To our knowledge, this is the first paper that investigates introducing the OpenFlow-style SDN capability into InfiniBand and evaluates the potential benefits.

VI. Conclusions

We investigate schemes to incorporate OpenFlow-style per-flow resource management into the current InfiniBand, illustrate a use case, and evaluate the potential benefits of the SDN-enhanced InfiniBand. We show that with limited modifications to the current InfiniBand standard, the per-flow resource management capability can be introduced to overcome the routing deficiency in the current InfiniBand and significantly improve the communication performance. This makes a strong case that the OpenFlow-style capability should be introduced in InfiniBand for building efficient HPC clusters.

Acknowledgements

This material is based upon work supported by the U.S. Department of Energy, Office of Science, Office of Advanced Scientific Computing Research, under Award Numbers 0000219853 and DE-SC0016039.

References

[1] Hans Meuer, Martin Meuer, Erich Strohmaier, Jack Dongarra, and Horst Simon. Top 500 list. http://www.top500.org/list/2015/11.

[2] Darren J Kerbyson and Kevin J Barker. Automatic identification of application communication patterns via templates. *ISCA PDCS*, 5:114–121, 2005.

[3] Open Networking Foundation. Openflow switch specification, March 2015. Version 1.5.1 (Protocol version 0x06).

[4] German Rodriguez, Cyriel Minkenberg, Ramon Beivide, Ronald P Luijten, Jesus Labarta, and Mateo Valero. Oblivious routing schemes in extended generalized fat tree networks. In *Cluster Computing and Workshops, 2009. CLUSTER'09. IEEE International Conference on*, pages 1–8. IEEE, 2009.

[5] J. C. Martinez, J. Flich, A. Robles, P. Lopez, and J. Duato. Supporting fully adaptive routing in infiniband networks. In *Parallel and Distributed Processing Symposium, 2003. Proceedings. International*, pages 10 pp.–, April 2003.

[6] Open Networking Foundation. SDN Architecture. Technical report, June 2014. White paper, ONF TR-502.

[7] InfiniBand Trade Association. *InfiniBand Architecture Specification: Release 1.3*. InfiniBand Trade Association, March 2015.

[8] InfiniBand Trade Association. Infiniband®roadmap. http://www.infinibandta.org/content/pages.php?pg=technology_overview, 2016.

[9] Mellanox Technologies. Infiniband: the production sdn. Technical report, 2012. White Paper.

[10] T. Hoefler, T. Schneider, and A. Lumsdaine. Multistage switches are not crossbars: Effects of static routing in high-performance networks. In *2008 IEEE International Conference on Cluster Computing*, pages 116–125, Sept 2008.

[11] Xin Yuan, Wickus Nienaber, and Santosh Mahapatra. On folded-clos networks with deterministic single-path routing. *ACM Trans. Parallel Comput.*, 2(4):27:1–27:22, January 2016.

[12] Gregory Johnson, Darren J. Kerbyson, and Mike Lang. Optimization of infiniband for scientific applications. In *Proceedings of the 2008 IEEE International Symposium on Parallel and Distributed Processing*, pages 1–8. IEEE Computer Society Press, 2008.

[13] Xin Yuan, Rami Melhem, and Rajiv Gupta. Compiled communication for all-optical tdm networks. In *Supercomputing, 1996. Proceedings of the 1996 ACM/IEEE Conference on*, pages 25–25, 1996.

[14] Andrew R. Curtis, Jeffrey C. Mogul, Jean Tourrilhes, Praveen Yalagandula, Puneet Sharma, and Sujata Banerjee. Devoflow: Scaling flow management for high-performance networks. In *Proceedings of the ACM SIGCOMM 2011 Conference*, SIGCOMM '11, pages 254–265, New York, NY, USA, 2011. ACM.

[15] The opensm open source project on open hub. https://www.openhub.net/p/opensm.

[16] Slurm workload manager. http://slurm.schedmd.com/.

[17] Steve J Chapin, Walfredo Cirne, Dror G Feitelson, James Patton Jones, Scott T Leutenegger, Uwe Schwiegelshohn, Warren Smith, and David Talby. Benchmarks and standards for the evaluation of parallel job schedulers. In *Job Scheduling Strategies for Parallel Processing*, pages 67–90. Springer, 1999.

[18] The standard workload format. http://www.cs.huji.ac.il/labs/parallel/workload/swf.html.

[19] Joseph Emeras. The cea curie log. http://www.cs.huji.ac.il/labs/parallel/workload/l_cea_curie/index.html.

[20] Moe Jette. The llnl thunder log. http://www.cs.huji.ac.il/labs/parallel/workload/l_llnl_thunder/index.html.

[21] Moe Jette. The llnl atlas. http://www.cs.huji.ac.il/labs/parallel/workload/l_llnl_atlas/index.html.

[22] Dror Feitelson. Parallel workloads archive. http://www.cs.huji.ac.il/labs/parallel/workload/.

[23] Dror G Feitelson and Dan Tsafrir. Workload sanitation for performance evaluation. In *Performance analysis of systems and software, 2006 IEEE international symposium on*, pages 221–230. IEEE, 2006.

[24] P. G. Raponi, F. Petrini, R. Walkup, and F. Checconi. Characterization of the communication patterns of scientific applications on blue gene/p. In *Parallel and Distributed Processing Workshops and Phd Forum (IPDPSW), 2011 IEEE International Symposium on*, pages 1017–1024, May 2011.

[25] X. Yuan, S. Mahapatra, M. Lang, and S. Pakin. Lfti: A new performance metric for assessing interconnect designs for extreme-scale hpc systems. In *Parallel and Distributed Processing Symposium, 2014 IEEE 28th International*, pages 273–282, May 2014.

[26] Scott Pakin, Xin Yuan, and Michael Lang. Predicting the Performance of Extreme-Scale Supercomputer Networks. *The Next Wave (http://www.nsa.gov/research/tnw/)*, 20, 2013.

[27] Keichi Takahashi, Dashdavaa Khureltulga, Baatarsuren Munkhdorj, Yoshiyuki Kido, Susumu Date, Hiroaki Yamanaka, Eiji Kawai, and Shinji Shimojo. Concept and design of sdn-enhanced mpi framework. In *Software Defined Networks (EWSDN), 2015 Fourth European Workshop on*, pages 109–110. IEEE, 2015.

[28] Omer Arap, Geoffrey Brown, Bryce Himebaugh, and Martin Swany. Software defined multicasting for mpi collective operation offloading with the netfpga. In *Euro-Par 2014 Parallel Processing*, pages 632–643. Springer, 2014.

[29] K. Takahashi, D. Khureltulga, Y. Watashiba, Y. Kido, S. Date, and S. Shimojo. Performance evaluation of sdn-enhanced mpi allreduce on a cluster system with fat-tree interconnect. In *High Performance Computing Simulation (HPCS), 2014 International Conference on*, pages 784–792, July 2014.

[30] Khureltulga Dashdavaa, Susumu Date, Hiroaki Yamanaka, Eiji Kawai, Yasuhiro Watashiba, Kohei Ichikawa, Hirotake Abe, and Shinji Shimojo. Architecture of a high-speed mpi_bcast leveraging software-defined network. In *Euro-Par 2013: Parallel Processing Workshops*, pages 885–894. Springer, 2013.

[31] Mellanox. Mellanox openstack and sdn/openflow solution reference architecture. https://www.mellanox.com/sdn/pdf/Mellanox-OpenStack-OpenFlow-Solution.pdf, September 2013.