Scheduling-Aware Routing for Supercomputers

Jens Domke
Institute of Computer Engineering
TU Dresden, Germany
Email: jens.domke@tu-dresden.de

Torsten Hoefler
Computer Science Department
ETH Zurich, Switzerland
Email: htor@inf.ethz.ch

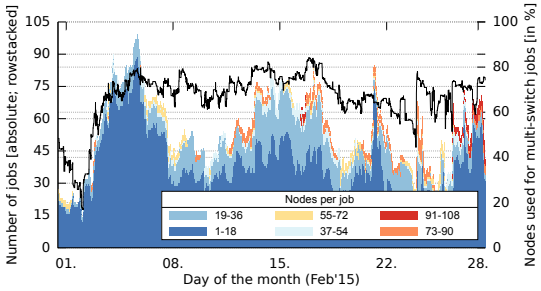
***Abstract*—The interconnection network has a large influence on total cost, application performance, energy consumption, and overall system efficiency of a supercomputer. Unfortunately, today's routing algorithms do not utilize this important resource most efficiently. We first demonstrate this by defining the *dark fiber* metric as a measure of unused resource in networks. To improve the utilization, we propose scheduling-aware routing, a new technique that uses the current state of the batch system to determine a new set of network routes and so increases overall system utilization by up to 17.74%. We also show that our proposed routing increases the throughput of communication benchmarks by up to 17.6% on a practical InfiniBand installation. Our routing method is implemented in the standard InfiniBand tool set and can immediately be used to optimize systems. In fact, we are using it to improve the utilization of our production petascale supercomputer for more than one year.**

***Index Terms*—High performance computing, Computer network management, Routing protocols, Unicast**

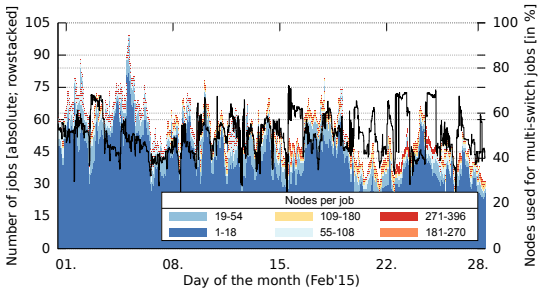
1. Introduction and Motivation

With Moore's law slowly coming to an end, high-performance computers (HPC) started to scale-out to larger systems. It can be expected that the number of network endpoints will grow significantly [1] which emphasizes the role of the interconnection network as one of the most critical components in a supercomputer. The network is largely controlled by routing algorithms which determine how to forward packets. These algorithms have to balance multiple, partially conflicting, requirements. For example, they shall provide the best forwarding strategy (in general an NP-hard problem) while minimizing the runtime of the routing in order to quickly react to failures of network components.

Routing algorithms have been the topic of many studies ranging from topology-specific routing algorithms [2], [3] through general deadlock-free algorithms [4], [5], more advanced deadlock-free algorithms balancing the routes [6], to advanced path-caching for quick failover [7]; a good overview is provided by Flich et al. [8]. Many advanced approaches for application-specific [9], [10] or topology-specific [11], [12], [13] routing and mapping assume idealized conditions such as a regular topology without faulty components, isolated bulk-synchronous applications communicating in synchronized phases, and the absence of system noise. Unfortunately, these assumptions are rarely true in practice [14].



(a) Taurus supercomputer (multi-island design with 2014 compute nodes connected by FDR/QDR InfiniBand)



(b) Tsubame2.5 petascale system (1408 compute nodes connected by a two full-bisection fat-tree QDR networks)

Figure 1. Batch job history (sampled every 10 min) of two petascale HPC systems for one month of operation; only multi-switch jobs shown; total system utilization (incl. single-node and single-switch jobs) usually higher

A brief analysis of the job mix on two production supercomputers emphasizes the complexity of real-world installations. Figure 1 shows all batch jobs that are using at least two network switches on two InfiniBand-based petascale supercomputers, Taurus [15] and Tsubame2.5 [16], in the timeframe of one month. We see that multiple parallel applications are spread throughout the system at any time and are thus contending for bandwidth on the shared network resources. In fact more than 66% and 50% of the compute jobs on Taurus and Tsubame2.5, respectively, are using multiple switches. But at the same time, communications do generally not cross from one parallel applications to another.

When computing (oblivious) routes for an interconnection

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

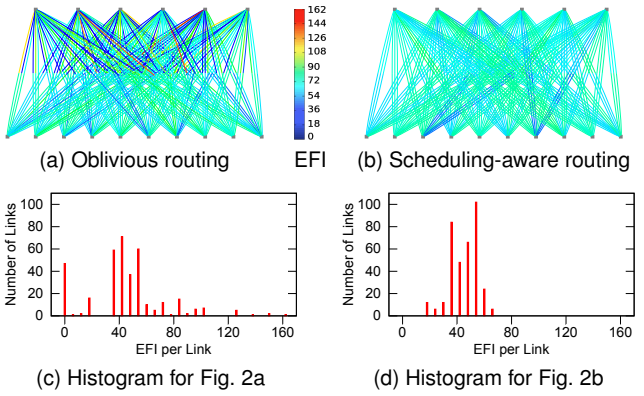


Figure 2. Comparison of EFI for inter-switch links (heat map in (a) and (b)); oblivious routing vs. scheduling-aware routing for three equal-sized batch jobs on a 2-level fat-tree; two colors of individual full-duplex links in (a) and (b) indicate different EFIs for the two opposite directions

network, the algorithm tries to balance the number of routes across all links so that each physical network cable has a similar number of routes crossing it. This number is called the Edge Forwarding Index (EFI), a theoretical upper bound for the worst-case congestion of a set of routes [17]. To illustrate our idea, we perform the following experiment: we route a two-level fat-tree network with an optimal oblivious strategy (ideally balanced EFI) and schedule three different applications (jobs) to the system. The allocation of application processes to nodes logically partitions the network into three pieces which are not crossed by communications. Now, we compute the effective EFI for each link by only counting routes connecting endpoints that belong to the same job. Figure 2a shows a heat map of the effective EFIs for each link in the network; Figure 2c shows the corresponding EFI histogram. We derived two conclusions directly from the figure: (1) there are hotspots of high route counts per link which can decrease communication performance of one or more jobs, and (2) some links are underutilized or not used at all which reduces the system's efficiency. A less obvious observation is that the total number of switches and links available for intra-job communication is suboptimal.

We now use this example to illustrate our idea of **scheduling-aware routing** (SAR) which aims to maximize the allocation of network resources per batch job. Hence, using SAR, intra-job communication throughput increases due to the increased available network hardware; this directly increases the utilization of the overall supercomputer. In other words, SAR exploits the knowledge of the batch system (the current job mix) to determine the application locality and guides the fabric manager in assigning routing paths in the network. Furthermore, SAR has a similarly low runtime complexity as other oblivious routing algorithms, and broad applicability to current and future interconnection technologies and supercomputers. Figures 2b and 2d show the same three-job example as before but using our SAR approach. It's easy to see that the maximum EFI (i.e., upper bound for congestion) reduced from more than 160 to a mere 60 and the overall balance is greatly improved.

Our main contributions and findings are:

- We define the notion of dark fiber which describes network hardware that is unused in a current system configuration (job mix).
- We introduce the new concept of scheduling-aware routing, which performs frequent path reconfigurations based on concurrently running applications.
- We extend the formalism of per-packet consistent updates for in-order delivery and deadlock-freedom and show how to achieve both conditions using current InfiniBand hardware.
- We implement a low-overhead scheduling-aware routing for InfiniBand-based HPC systems and demonstrate its benefits over state-of-the-art routing mechanisms using simulations and communication benchmarks.

2. Background, Assumptions, and Definitions

This section formalizes network-related terms, as well as new metrics, to evaluate and compare our scheduling-aware routing approach with other state-of-the-art algorithms.

First, we define an interconnection network as a multigraph where each pair of network devices is connected by one or more full-duplex links (or channels). These full-duplex links can be used in opposing directions without interference or throughput degradation. Furthermore, we assume that the link capacity, i.e., the maximum achievable throughput, in the network is uniform and constant over time.

Definition 1 (Interconnection network)**.** *An **interconnection network** $I := G(N, C)$ is a connected multigraph with the node set N. The channel set C consists of directed (multi-)channels, denoted by ordered pairs $(\cdot, \cdot)$. We call $n_x \in N$ a **terminal** if exactly one n_y exists with $(n_y, n_x) \in C$, otherwise n_x is called a **switch**. With $C^* \subseteq C$ we denote inter-switch links connecting two switches of I.*

Terminals can be further subdivided into **compute nodes**, executing scientific applications in a supercomputer, and **supplementary nodes**, responsible for storage, administration, etc. Network nodes communicate via **messages** of arbitrary length, which are transferred as payload of one or more **network packets** from sending node to receiving node. The packet routes, see Definition 2 for all nodes to all other nodes are calculated by a routing algorithm, see Definition 3, within the multigraph representing the network.

Definition 2 (Route or path)**.** *A **route** P_{n_x,n_y} of length $h \geq 1$ from node n_x to node n_y in the network $I = G(N, C)$ is defined as a sequence of links $(c_1, \ldots, c_h) =: P_{n_x,n_y}$. These links have to satisfy the following two conditions: $\{c_1, \ldots, c_h\} \subseteq C$, with $c_1 := (n_x, \cdot)$, $c_h := (\cdot, n_y)$, and if $c_q = (\cdot, n_z)$ then $c_{q+1} = (n_z, \cdot)$ for all $1 < q < h$.*

Definition 3 (Flow-oblivious, destination-based routing)**.** *A **routing function** $R : N \times N \to C$ for a network $I := G(N, C)$ assigns the next channel c_{q+1} on the path depending on the current node n_z and the destination node n_y, which is encoded in the packet header. The entirety of all paths in the network configured by a single routing function is called **routing configuration**.*

Furthermore, we presume that the routing algorithm must calculate **deadlock-free** routes [18], to be applicable to lossless interconnection networks, which are primarily used in modern supercomputers. This deadlock-freedom can be ensured if the routing satisfies the necessary and sufficient condition of Dally and Seitz's first theorem [4], i.e., the corresponding channel dependency graph for a routing needs to be acyclic. Generally speaking, the (deadlock-free) routing configuration is part of the switching rules for a network, which define a set of valid forwarding rules and packet modifications along the path to the destination. We call these switching rules a (global) **network state** from here on.

One problem while performing network updates, i.e., the transition between two network states, is to preserve per-packet or per-flow consistency [19], [20], and therefore preserve certain properties satisfied by each network state. Usually, "atomic" state transitions for the whole network are impossible, hence without consistency, these network updates can lead to security vulnerability, loss of packets, etc. We summarize the comprehensive theoretical groundwork given by Reitblatt et al. [19] in the two Definitions 4 and 5.

Definition 4 (Per-packet and per-flow consistency)**.** ***Per-packet consistency*** *of the network operation is achieved if and only if each packet injected into the network is processed by exactly one network state. If all packets of each injected message or flow are processed by the same network state, then we refer to this as* ***per-flow consistency****.*

Definition 5 (Per-packet consistent network update)**.** *Let u_s be a sequence of atomic updates u applied to a set of network switches. Then, for two (distinguishable) network states S and S', the transition $S \rightarrow_{u_s} S'$ is called a* ***per-packet consistent update*** *if and only if each packet is processed by either S or S', but not any inconsistent state in-between.* ***Per-flow consistent updates*** *are defined analogously.*

The property preservation of network updates, as stated by Theorem 1 of Reitblatt et al. [19], is a direct result when per-packet consistent updates are performed. However, this only holds for properties related to single packets, e.g., connectivity or loop-freedom. Other network properties, such as in-order delivery or deadlock-freedom, which are based on the correlation of multiple packets are harder to achieve. Hence, for our purpose of scheduling-aware rerouting of HPC systems, we extend Reitblatt's property preservation to be applicable to lossless and deadlock-free interconnects.

Definition 6 (Property Preserving Network Update)**.** *The transition $S \rightarrow_{u_s} S'$ between two network states S and S' is called a* ***property preserving network update*** *for a lossless interconnection network if the following conditions hold:*

1) *both network states, S and S', are based on deadlock-free routing configurations;*
2) *$S \rightarrow_{u_s} S'$ is a per-flow consistent update for reliable channels of communication, and per-packet consistent update for unreliable connections; and*
3) *a temporally simultaneous processing of packets, either by S and S', during a transition is deadlock-free.*

Attribute 2 of Definition 6 can be weakened to per-packet consistent updates when in-order delivery is not required for reliable connections. Assuming our scheduling-aware (re-)routing is a property preserving network update, then we can guarantee fault-free execution of parallel applications (w.r.t. network-related problems) and guarantee uninterruptible operation of the HPC system. Further network properties based on multiple packets, such as congestion control or quality of service, are beyond the scope of this paper.

Besides the commonly known network metrics throughput and latency, we will make use of two new metrics. The first metric extends the established edge forwarding index [17] to differentiate between "useful" intra-job paths and paths connecting compute nodes which do not work on the same scientific problem, see Definition 7. In this regard, a **batch job**, or short **job** j, is a (parallel) application scheduled onto a subset of compute nodes, $N_j \subseteq N$, which collectively work on a given task during a finite timeframe.

Definition 7 (Effective Edge Forwarding Index)**.** *The* ***effective edge forwarding index*** *γ^e of a switch port or outgoing link $c_q \in C^*$ is the sum of intra-job routes being forwarded via this port, i.e.,*

$$\gamma^e(c_q) := \sum_j \left|\{P_{n_x,n_y} \mid n_x, n_y \in N_j \wedge c_q \in P_{n_x,n_y}\}\right|$$

for all jobs j running on the system.

Our second metric describes the amount of superfluous links in the network, metaphorically speaking: the proportion of "dark fiber" to links usable for intra-job communication, see Definition 8. Please note, that our latter metric does not differentiate between copper or fiber cables, and we use fiber as a synonym for link. Furthermore, this metric is only based on intra-job paths, and we ignore any other traffic related to supplementary nodes, such as I/O traffic to and from a remote filesystem or administrative data paths.

Definition 8 (Dark Fiber Percentage)**.** *The* ***dark fiber percentage*** *is the percentage of links in the system, which are not used for intra-job routes, and can therefore be derived from γ^e in the following way:*

$$\theta := \frac{\left|\{c_q \in C^* \mid \gamma^e(c_q) = 0\}\right|}{|C^*|}$$

3. Example Implementation of SAR

In Section 3.1, we outline the basic components, including relevant features and limitations, available on our petascale Taurus HPC system, followed by Section 3.2 giving details on how we extend and combine these building blocks to implement a scheduling-aware routing (SAR).

3.1. Hardware/Software Building Blocks

3.1.1. InfiniBand and OpenSM. The InfiniBand™ technology [21], as defined by the architecture specification (IBTAspec) [22], is widely used in current HPC systems to network compute and storage nodes. Each network device

in the fabric is identified by one or more local identifiers (LIDs), configured via a lid mask control (LMC) parameter such that the total number of LIDs per device is 2^{LMC}.

The IBTAspec defines a layer, called verbs, between the software/operating system and the InfiniBand hardware. Communication between two IB devices is implemented via queue pairs (QPs) that establish a send and receive channel between one source LID and one destination LID for unicast traffic[1]. However, the verbs layer also allows modifications of QPs, e.g., performing path migrations (APM) or draining all outstanding work requests (SQD), which we will use in Section 3.3 to enforce property preserving network updates.

Applications such as Open MPI (cf. Section 3.1.3) post work requests (WRs) to a queue pair to initiate data transport to a remote node. InfiniBand's hardware offloading and kernel-bypass characteristics allow the processing of these WRs asynchronously to the application. The channel adapter (HCA; network interface card of IB) splits a scheduled message into packets, configures packet headers, and dispatches them. Achieving per-flow consistency is made more complicated because of this asynchronicity, which we will elucidate further in Section 3.3. InfiniBand ensures lossless processing of WRs for reliable channels through credit-based flow control, both on a per-link basis and end-to-end between send/receive queues. However, reliable channels require in-order delivery of all packets of one WR, otherwise the IB hardware drops messages and requests retransmission.

Moreover, IB supports virtual lanes (VLs), which are basically different sets of buffers within one port. InfiniBand network ports must support up to 15 VLs for data traffic—8 data VLs is common on current hardware—and one VL for management traffic. Data VLs can be used for quality of service or to avoid potential deadlock configurations in the forwarding tables of switches. Various routing approaches use this feature to combine the i^{th} VL of each port into virtual layers and assign paths to different layers to create a deadlock-free routing configuration [6], [23], [24].

The IB fabric is controlled by a subnet manager, which is responsible for assigning LIDs and LMCs to nodes, reacts to topological changes and failures in the fabric, and calculates the routing configuration for the fabric, among other things. An open-source version of this subnet manager is called OpenSM [25], which currently implements nine flow-oblivious routing algorithms for a variety of supported network topologies [14]. These algorithms can be categorized into topology-aware routings, such as Up*/Down* [5] or fat-tree [26], and topology-agnostic routings, e.g., layered shortest path routing (LASH) [24] or (deadlock-free) single-source shortest-path routing [6], [27]. Once a new routing configuration is calculated, OpenSM updates the linear forwarding tables (LFT) of all IB switches, and packets are forwarded according to these new switching rules. The LFTs are per-packet lookup tables with the destination as index for the array and egress port stored in it, i.e., egress port = LFT[LID]. A LFT update usually only happens in response to a topological change in the fabric, e.g., when network components fail or new nodes are being added.

1. Multicast traffic is beyond the scope of this paper, because it is performed unreliably in InfiniBand, and therefore generally not used by parallel applications.

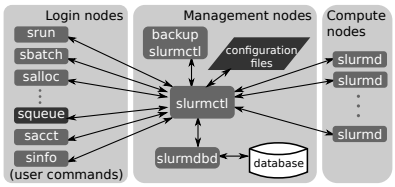


Figure 3. SLURM architecture (highlighted: squeue and configuration files which will be used by the filtering tool in Section 3.2.1)

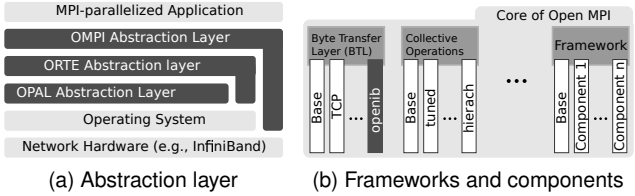


Figure 4. Open MPI's modular component architecture establishes an message passing interface between application and hardware to enable inter-process communication (highlighted: openib BTL for InfiniBand)

3.1.2. Simple Linux Utility for Resource Management. Many supercomputers utilize SLURM [28] as cluster resource manager and batch system. SLURM is open-source and offers a plugin interface, which makes it a preferred solution for many HPC sites, since it offers the ability for enhancements depending on individual needs and it offers the option to test research ideas. The SLURM architecture, see Figure 3, primarily consists of three parts: (1) user commands, (2) controller daemon, and (3) compute node daemons. The controller daemon manages most of the system's compute resources, and schedules batch jobs onto free compute nodes. However, the controller is partially or fully unaware of other resources, such as remote storage or interconnection network. For example, while SLURM is aware of the system's physical topology, i.e., node-to-switch mapping and inter-switch connectivity, it is unaware of the deployed routing configuration. The SLURM controller can be attached to a database, see Figure 3, which stores the historical and current state of job allocations for a system. We make use of this information in Section 3.2 to obtain a snapshot of simultaneously running jobs and their locality.

3.1.3. Open MPI. The message passing interface (MPI) specification [29] standardizes an API for parallel applications to perform inter-process communication, either locally on a compute node or via the interconnection network. One widely used open-source implementation of the MPI standard is Open MPI [30]. The Open MPI library assigns one rank ($\in [0, \ldots, n-1]$) to each processes within a single MPI-parallelized application, and therefore provides an abstract interface to the application to perform point-to-point or collective operations between ranks. Open MPI uses a modular component architecture (MCA) to support a variety of interconnection technologies and transport services, see Figure 4. The OMPI layer provides the MPI API to

the applications and implements communication algorithms for point-to-point and collective operations, monitoring and profiling interfaces, etc. One framework within the MCA is the byte transfer layer (BTL), responsible for point-to-point communication between MPI processes. The BTL for IB networks is called "openib", see Figure 4b. This openib component interfaces with the verbs layer, see Section 3.1.1, to set up QPs, schedule send or receive WRs, and handle events raised by the InfiniBand hardware, among other things. When Open MPI is configured with threading support (for POSIX threads), then the openib component spawns one asynchronous event thread per MPI process. In Section 3.3, we will use that asynchronous event thread to establish communication channels between OpenSM and MPI programs to achieve per-flow consistent network updates.

3.2. Scheduling-Aware Routing Optimization

3.2.1. Filter between Batch System and Subnet Manager. Performing a scheduling-aware routing optimization for a supercomputer is only beneficial for parallel applications when their compute nodes are attached to multiple switches. Therefore, we create a filtering tool, see Figure 5, which periodically polls SLURM for the current state of the system, i.e., information about batch jobs, including job state (pending, running, etc.), job identifier, and allocated compute nodes. This information is obtainable through SLURM's squeue command. Furthermore, we require the topology information, i.e., terminal-to-switch mapping, which is stored among SLURM's configuration files. This topology information is used to filter out a list of multi-switch jobs containing all currently running applications which have their compute nodes attached to at least two switches. The list of multi-switch jobs is compared to the list of the previous iteration while ignoring the actual job identifiers. Hence, we can avoid unnecessary calculations of LFTs, since replacing one job with another job on the same compute nodes will not result in a different routing configuration while using SAR, see Algorithm 1. If the two lists of multi-switch jobs differ, then our filtering tool writes a job-to-terminal mapping file for OpenSM, informs the OpenSM about the configuration change, and replaces the previous list with the new list for the next iteration. We extend OpenSM's signal handler to process the SIGUSR2 signal to inform the OpenSM about an updated job-to-terminal mapping.

We refrain from having a direct interface between SLURM and OpenSM for the following reasons: First off, portability, to easily support other batch systems or subnet managers, such as PBS [31] and derivatives, or the fabric manager for Intel Omni-Path [32]. Furthermore, the SLURM controller and OpenSM can run on different management nodes of a HPC system. Lastly, integrating the filtering functionality into the SLURM controller will increase the latency experienced by users when submitting many jobs.

3.2.2. Routing Optimization with DFSSSP. As basis for our SAR we choose the deadlock-free single-source shortest-path routing (DFSSSP) for three reasons: (1) DFSSSP is

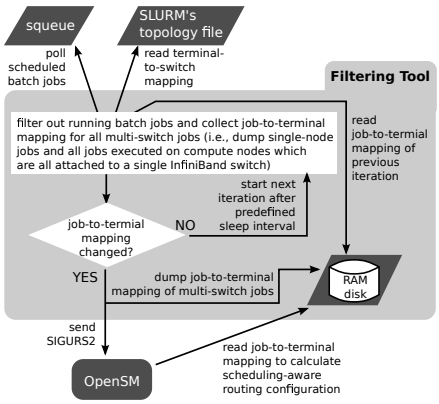


Figure 5. Flowchart of a filtering tool; collect information about currently running multi-switch batch jobs and initiate recalculation of LFT via SAR

deadlock-free and topology-agnostic, and therefore supports a wide variety of network topologies [6], (2) DFSSSP offers high global throughput for the complete HPC system and offers a well-balanced EFI across the system [14], and (3) DFSSSP already distinguishes three terminal types (compute, storage, and others nodes), and optimize their routing separately. Hence, SAR inherits the characteristics listed in (1) and (3) and improves upon (2).

DFSSSP routing computes the destination-based paths for the nodes of the interconnection network by applying a modified version of Dijkstra's shortest-path algorithm, optimized for multigraphs. DFSSSP iterates over all nodes of the network, and firstly applies Dijkstra's algorithm, followed by an update of the edge weights in the graph. This edge weight update, increasing the weight by $+1$ for every new path on a link, ensures the well-balanced EFI, which we mentioned above in reason (2). After calculating a path for every node-to-node combination, DFSSSP assigns these paths into virtual layers to achieve a deadlock-free routing configuration, similar to other routings, such as LASH [24].

We extend the DFSSSP routing to include knowledge about batch job locations, and enable the algorithm to optimize the path calculation for intra-job paths, shown as pseudo code in Algorithm 1. Our main modifications are outlined in lines 1–10, while subsequent lines are shown for sake of completeness and sketch the remainder of the already existing DFSSSP implementation in OpenSM. In the first part of Algorithm 1, we analyze the job-to-terminal mapping given by the filtering tool, see Section 3.2.1. Each node in the multigraph, representing the interconnection network, gets extended by a job array which stores all jobIDs currently running on this node. We presort these nodes descending by the size of largest job executed on the node, i.e., compute nodes belonging to the largest job running on the HPC system will be processed first in the following loop, see line 6. Hence, this sorting ensures that intra-job path balancing is increased and the number of overlapping paths is minimized. Since edge weights are only updated when actually used by an intra-job path, see lines 9 and 10 of Algorithm 1, the resulting paths should improve the network metrics compared to other

Algorithm 1: Scheduling-aware DFSSSP routing

Input: Network $I = G(N, C)$
Job-to-terminal mapping $B := [(\text{nodeName}, \text{jobID}), \ldots]$
Result: Scheduling-aware and deadlock-free routing configuration (P_{n_x,n_y} for all $n_x, n_y \in N$)

```
/* Process job-to-terminal mapping                          */
foreach node n ∈ N do
    n.jobList ← empty list []
    foreach pair (nodeName,jobID) ∈ B do
        if n.nodeName = nodeName then n.jobList.append(jobID)
/* Optimize routing for compute nodes                       */
N_sorted ← Sort N descending by the job size executed on n ∈ N
foreach node n_d ∈ N_sorted do
    Calculate one path P_{n_x,n_d} for every pair (n_x, n_d), with n_x ∈ N,
    with the modified Dijkstra algorithm (details in [6])
    foreach node n_x ∈ N do
        if n_x.jobList ⋂ n_d.jobList ≠ ∅ then
            Increase edge weight by +1 for each link in path P_{n_x,n_d}
/* Optimize routing for storage nodes                       */
foreach storage node n_d ∈ N do
    Calculate one path P_{n_x,n_d} for every pair (n_x, n_d), with n_x ∈ N
    Update edge weights for all links used by P(·, n_d)
/* Optimize routing for all other nodes                     */
foreach node n_d ∈ N ∧ n_d not processed before do
    Calculate one path P_{n_x,n_d} for every pair (n_x, n_d), with n_x ∈ N
    Update edge weights for all links used by P(·, n_d)
/* Create deadlock-free routing configuration               */
foreach route P_{n_x,n_y} calculated above do
    Assign P_{n_x,n_y} to one virtual layer without creating a cycle in the
    corresponding channel dependency graph (see [6] for details)
```

state-of-the-art oblivious routings, which we will demonstrate in Section 4. The rest of the base DFSSSP algorithm is kept in place to ensure deadlock-freedom and a well-balanced routing configuration towards the remote filesystem.

3.3. Property Preserving Network Update for IB

Changing the routing configuration of an InfiniBand network cannot be done atomically. OpenSM splits the LFT of each switch into multiple management datagram (MAD) packets, each carrying a payload of 64 bytes, to distribute new LFTs to all switches. This non-atomicity can cause out-of-order packet delivery for a flow, and therefore preventing per-flow consistency and disrupting InfiniBand's reliable connection service, see Section 3.1.1, resulting in application crashes in the worst case.

Multiple network update protocols for software defined networks (SDN) have been proposed [19], [20]. For example, the proposed two-phase updated installs passive routing configurations which are activated when the switch identifies a certain packet, so that subsequent packets follow the same path. Unfortunately, none of these protocols are applicable to IB due to missing features, i.e., the ability to deploy two routing configurations simultaneously, to distinguish between flows, or to tag packets. The only safe method of updating the forwarding rules of a path in InfiniBand is if we can guarantee that no packet or flow is using the entire path during the update process.

We propose a five-phase update protocol guaranteeing property preserving network updates for reliable connections, see Definition 6, within the limitations of the InfiniBand architecture specifications (IBTAspec) [22]. For our prototype implementation of this update protocol we use Open MPI in

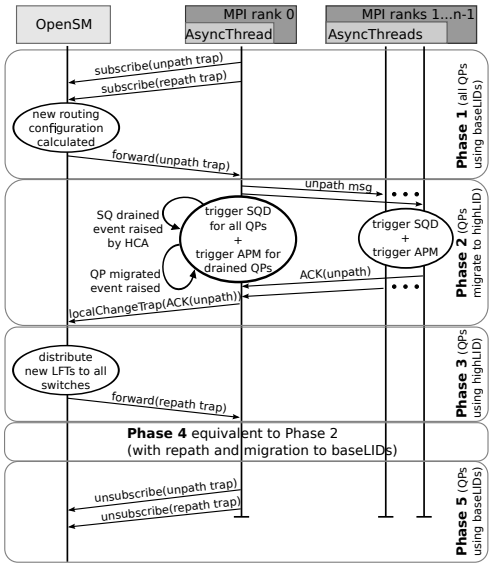


Figure 6. Sequence diagram of a five-phase update protocol to achieve property preserving network updates of InifiniBand networks

version 1.8.4 and OpenSM in version 3.3.18, however we are most certain that this protocol can be applied to other services in a supercomputer, e.g., for reliable communication with a remote filesystem. The sequence diagram in Figure 6 exemplifies the interaction between an MPI-parallel application and OpenSM, which performs the routing reconfiguration.

The initial assumptions for our update protocol to be applicable are the following, and will ensure deadlock-freedom prior to the network update:

1) OpenSM assigns two LIDs (we distinguish between baseLID and highLID, which equals to baseLID + 1) for each node in the network (via LMC=1);
2) The Routing configuration for baseLIDs is calculated by our Algorithm 1 (using VLs $0, \ldots, n-2$);
3) Routing configuration for highLIDs is calculated by Up*/Down* and uses VL $= n-1$; and
4) Packets are only sent from baseLIDs to baseLIDs or among highLIDs, but not a mixture of both.

During phase 1, an MPI-parallelized application, or the asynchronous event thread (AsyncThread) of rank 0 (see Section 3.1.3) to be precise, uses InformInfo MAD packets to subscribe for event forwarding, see IBTAspec Sections 13.4.8.3 and 13.4.11. All packets for inter-process communication within the MPI application are using baseLIDs for destination addressing. The MPI application uses a uMAD library to send userspace management datagram packets to communicate with the OpenSM. The uMAD library also allows the AsyncThread of rank 0 to periodically poll for MAD packets send by the OpenSM. The AsyncThread subscribes[2] only for unpath and repath traps, see IBTAspec

2. Theoretically, subscriptions and forwarded traps contain individual paths records, however our subscriptions apply for the whole subnet's LID range, and traps contain empty path records. This increases scalability by reducing the number of MADs handled by both sides, i.e., only one MAD, with either unpath or repath trap, has to be sent out to each subscriber.

Section 14.4.12. These traps are intended to inform terminals about usability changes of certain routes. Once the OpenSM calculates a new routing configuration for the baseLIDs, in consequence of our approach outlined in Section 3.2, it raises an unpath trap and forwards the trap to all subscribers, but it withholds to reconfigure any LFT. The AsyncThread of rank 0 forwards the unpath trap to all other ranks of the same application by the use of MPI calls, i.e., primarily MPI_Isend, MPI_Irecv, and MPI_Test. This tree-like approach is required, because the uMAD library cannot be used simultaneously by multiple AsyncThreads on one compute node.

In phase 2, all processes of all subscribed parallel applications modify their queue pairs, which belong to reliable connections, to enter the draining state, see IBTAspec Section 10.3.1.5. While in draining state, MPI processes can still post new WRs to the verbs layer, however these will not be send out before the QP reenters the "ready-to-send" state. Once the draining of an QP is complete, the IB HCA sends an event notification to the AsyncThread. After receiving this "QP drained"-notification, the AsyncThread triggers a path migration to the highLIDs, see IBTAspec Section 10.4, changes the QP back into the "ready-to-send" state, and then loads a new alternate path. The new alternate path is always the opposite of the current path, i.e., if the current path is baseLID→baseLID, then the new alternate path will be highLID→highLID. When all local QPs have been migrated to the highLID, then each process informs rank 0, and acknowledges the successful migration. The AsyncThread of rank 0 uses a "local changes" trap, see IBTAspec Section 14.3.13, informing the OpenSM about the successful draining of baseLID-related traffic, after it received acknowledgments by all ranks.

When OpenSM received confirmation by all subscribers, indicating that no reliable connections are using baseLIDs in the entire network, the protocol switches to phase 3. OpenSM reconfigures all switches with new LFTs, which were calculated during phase 1, for the baseLIDs. The routing configuration for the highLIDs is never changed, since it relies on the oblivious Up*/Down* routing, and is kept in place until a failure in the network requires fault-recovery and routing recalculation for baseLIDs and highLIDs[3].

Phase 4 starts after all switches have been reconfigured, and it is essentially a copy of phase 2 with the exception that the OpenSM requests the subscribers to migrate all their reliable connections from the highLIDs back to the baseLIDs. During phase 5, OpenSM listens for applications canceling their forwarding subscriptions and waits for new scheduling-aware rerouting requests. Upon completion of the MPI-based application the AsyncThread of rank 0 sends out unsubscribe InformInfo MADs to the OpenSM.

The whole draining and path migration process is transparent to the overlying application. At most, the application could experience a minor latency increase for a particular QP while it is draining, and experience reduced throughput while the highLIDs are used.

3. Property preserving network updates during fault-recovery in an InfiniBand network is beyond the scope of this paper.

4. Evaluation

4.1. Current Limitations and Problems

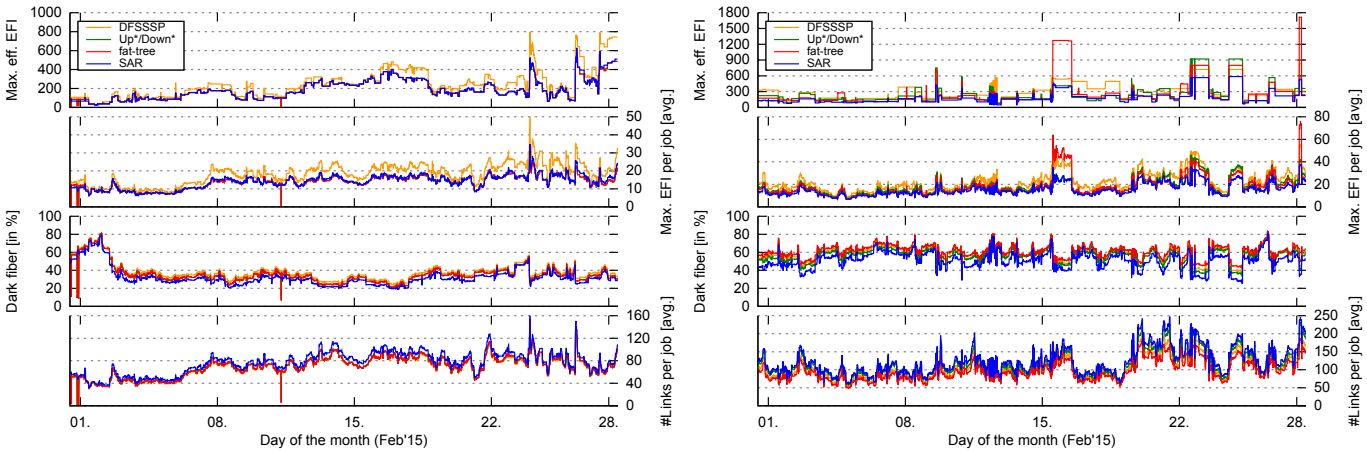
When this research was conducted, we experienced minor problems while implementing our network update protocol, shown in Figure 6. For the sake of completeness, we will list these limitations without going into much detail about possible solutions. First off, the communication between the AsyncThread of rank 0 and the OpenSM is potentially subject to packet loss, because (u)MAD packets to subscribe and forward traps use QP0 and QP1, special QPs configured for unreliable transport service, see IBTAspec Sections 3.9.4 and 3.9.5. However, both the uMAD library and OpenSM usually send MADs multiple times if they are not acknowledged in time. Second, Open MPI combined with the openib component does not support simultaneous calls to the MPI API from multiple threads of one process. Manually serializing MPI calls between the main application and the AsyncThread via a pthread mutex lock is our current workaround. And lastly, any attempts to modify QPs into the draining state was prevented by the tested firmwares for our IB devices. Depending on firmware version, either the verbs call was possible but had no effect, or the firmware rejected the verbs call. Therefore, we refrain from showing practical results of the property preserving network update protocol for our petascale system in the following sections. However, we successfully tested the entire update protocol (apart from the QP draining) presented in Sections 3.2 to 3.3 on a small test bed, see Section 4.3.4.

4.2. Theoretical Evaluation of Network Metrics

Our evaluation and comparison of the effective edge forwarding index and dark fiber percentage, see Definitions 7 and 8, is based on the same batch job history we presented in Figure 1 for the two HPC systems, called Taurus [15] and Tsubame2.5 [16]. Hence, we "replay" exactly the same job composition of each system and investigate the effects of different routing algorithms. These algorithms are Up*/Down*, fat-tree, and DFSSSP, all implemented in OpenSM [25], and we compare them against our SAR implementation. It is worth noting that the vendor of our Taurus HPC system originally suggested the usage of fat-tree routing before we deployed SAR, see Section 4.3. Furthermore, fat-tree routing is the current default on the Tsubame2.5 supercomputer.

4.2.1. Effective Edge Forwarding Index. We are analyzing two metrics with respect to the EFI. The first metric is $\gamma^e_{\max} := \max_{c_q \in C^*} \gamma^e(c_q)$, i.e., the maximum γ^e of all links, which implies hotspots of links shared by multiple jobs. The second metric is the maximal edge forwarding index $\gamma_{\max}(j)$ per job j, which should also be as low as possible, because it bounds the worst-case congestion within a job, i.e., the link with the most intra-job routes crossing it.

The results for $\gamma^e_{\max}$ are shown in the top plot of Figure 7a and 7b. As one can see, our scheduling-aware routing



(a) Taurus petascale supercomputer

(b) Tsubame2.5 petascale supercomputer

Figure 7. Replay of job history (Figure 1) for two HPC systems; four routings applied per sampling point; Metrics collected: maximal γ^e across all links (top), γ_m averaged for all jobs (2nd plot), dark fiber percentage (3rd plot), and used links averaged for all jobs (bottom); lower is better for first three plots

TABLE 1. IMPROVEMENTS BY OUR SCHEDULING-AWARE ROUTING COMPARED TO DFSSSP, FAT-TREE, AND UP*/DOWN* ROUTING

	Taurus HPC system						Tsubame2.5 HPC system					
Metric	DFSSSP		fat-tree		Up*/Down*		DFSSSP		fat-tree		Up*/Down*	
	max. / in %	avg. / in %	max. / %	avg. / %	max. / %	avg. / %	max. / %	avg. / %	max. / %	avg. / %	max. / %	avg. / %
$\max_{c_q \in C^*} \gamma^e(c_q)$	279.0 / 50.8	57.3 / 23.3	18.0 / 21.2	-1.1 / -0.6	18.0 / 21.2	-1.1 / -0.6	321.0 / 61.2	119.4 / 38.5	1186.0 / 71.2	80.5 / 29.7	354.0 / 48.7	69.9 / 26.8
$\text{avg}_{j \in J} \gamma_{\max}(j)$	16.0 / 39.0	4.3 / 23.4	1.6 / 11.4	-0.3 / -2.1	1.6 / 11.4	-0.3 / -2.1	18.9 / 46.0	6.7 / 30.1	38.0 / 49.8	2.7 / 14.8	10.6 / 29.9	2.4 / 13.2
θ [in %]	9.38	6.03	7.64	3.62	7.64	3.62	12.06	7.63	17.74	9.99	9.24	5.51
$\text{avg}_{j \in J}$ #links(j)	16.5 / 15.0	7.7 / 11.1	14.1 / 13.8	6.6 / 9.4	14.1 / 13.8	6.6 / 9.4	49.2 / 26.7	18.3 / 17.4	75.1 / 43.9	26.5 / 27.3	22.3 / 14.9	7.4 / 6.4

(blue line) outperforms DFSSSP for the Taurus system and outperforms the other three routing algorithms on the Tsubame2.5 supercomputer. For a quantitative comparison, we summarize the results in Table 1. The results in the table show the improvement by SAR for each metric (given as maximum and as average across all sample points within the month, as well as the improvement in percent). For example, the 279.0 and 56.41 for DFSSSP mean that SAR reduced $\gamma^e_{\max}$ by 279.0 (or 50.8%), at least once compared to DFSSSP routing, and reduced $\gamma^e_{\max}$ by 56.41 (or 23.1%), on average during the full month.

We see a reduction of $\gamma^e_{\max}$ when SAR is used for both HPC systems, especially for Tsubame2.5. This indicates that the worst-case link congestion is lowered. The tremendous differences between fat-tree and our SAR on Tsubame2.5, around day 16 and day 28, indicate very unfortunate job-to-node placements, which does not match the default fat-tree routing of Tsubame2.5. SAR slightly increases $\gamma^e_{\max}$ on Taurus in comparison to fat-tree and Up*/Down* routing, however on average SAR assigned only on more path onto the link with the highest load. This disadvantage in the worst-case bound of SAR is negligibly small, especially in the context of the other two metrics, link utilization and link availability. We also remark that Taurus had a fault-free and regular topology and we assume that SAR will outperform fat-tree and Up*/Down* if the system's network topology becomes more irregular over time due to network faults [14].

For the second metric, the maximal edge forwarding index per job (2nd plot in Figure 7), we see a similar behavior. Our SAR approach lowers the maximum number of inter-job routes compared to the other routings, which should accelerate the achievable throughput within the job, which we will showcase in Section 4.3.4.

4.2.2. Available Links per Job and Dark Fiber. While the EFI provides upper bounds for worst-case congestion, we now evaluate a more direct metric for utilization. For this, we analyze the number of switch-to-switch links that are available to each job and to the overall set of jobs. The dark fiber percentage θ is shown in the third plot of Figure 7a and 7b for both supercomputers, and is summarized in the third row of Table 1. Especially for Tsubame2.5, the improvement to θ by our scheduling-aware routing is clearly visible in the plot. We see that SAR increases the number of links available to running batch jobs by up to 17.74% for Tsubame2.5 in comparison to the default routing on

this system. For both systems, SAR consistently increases θ and thus enables a more efficient resource utilization. SAR utilizes between 3.6% and 9.9% more links on the month-long average than the other oblivious routing mechanisms.

Lastly, the bottom plots of Figure 7 show that SAR increases the number of intra-job links as well ("higher is better" applies to these two plots) directly leading to a higher effectively available bandwidth for the applications. Depending on the routing algorithm and HPC system, our SAR approach allows the average multi-switch job to use between 6 and 26 more links.

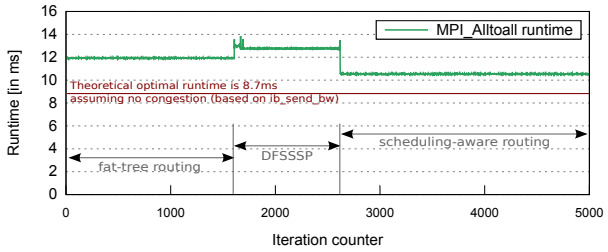


Figure 8. Runtime measurement for MPI_Alltoall (w/ 1 MiB in each send buffer) on 28 compute nodes using three different routing approaches

4.3. Practical Evaluation on a Production System

We are using our scheduling-aware routing, as presented in Section 3.2, now for more than one year on our petascale Taurus production system [15] at the TU Dresden. The system's InfiniBand fabric is designed as a multi-island network. Multiple smaller 2-level full-bisection fat-trees are connected by a 216-port director switch. The following four subsections show numbers collected while our system was open to regular users. However, we omit showing runtime comparisons to calculate the linear forwarding tables with SAR for the following reasons: (1) SAR and DFSSSP have the same runtime complexity of $\mathcal{O}\big(|N|^2 \cdot \log|N|\big)$, for the node set N, (2) measurements revealed a negligible runtime overhead introduced by our extensions, shown in Algorithm 1, and (3) the competitive runtime of DFSSSP routing has been shown before, e.g., refer to [6], [33], [34] for detailed comparisons.

4.3.1. Runtime of the Filtering Tool. Our current reconfiguration tool runs with a 5 min interval (cf. Figure 5) to minimize the chance that SAR reroutes for batch jobs that only run for a very short time. We measure the runtime of the tool including polling the SLURM controller and analyzing currently running jobs. The average time spent by our filtering tool is 16 s, with a low of 0.02 s. In 99.1% the runtime was below 2 min. We only experienced three occasions in over a year, where the runtime was above 10 min. The vast majority of the runtime of our tool results from the latency to obtain the list of scheduled jobs via the squeue command, which depends on the load of the SLURM controller.

4.3.2. New Routing Configurations per Day. The number of reconfigurations of the network per day, due to a substantive change in the job mix and job locations, ranges between 0 and 57, with an average of 14 reconfigurations per day. Therefore, approximately every two hours our system's forwarding tables are adjusted to increase the performance of the running parallel applications. Only for four days in over a year of operation no reconfigurations were needed, and three out of these four days were on weekends, and one time on Monday, i.e., matching the typical days of lower load on a production system in a university environment.

4.3.3. Runtime for Configuring LFTs of all IB Switches. The time it takes to reconfigure all switches in the IB network is of importance. Either because it defines the time window, where out-of-order packet delivery can happen if property preserving network updates are not enforced, or because it defines the timeframe during which the network throughput might be decreased due to the use of the highLIDs (routed by the less efficient Up*/Down*). We measured an average of 4.6 μs to send a LFT block to a switch and receive an acknowledgment. Instead of reconfiguring all LFT blocks of a switch at once, OpenSM iterates over the block numbers and sends out the first LFT block of each switch, before processing the second block, and so on. So, theoretically OpenSM could reconfigure one block number for all 210 switches in our Taurus system in under 1 ms, however we measure a runtime between 25 ms and 50 ms for this process. Despite this OpenSM-internal overhead, the whole LFT reconfiguration for our system is completed in ≈0.8 s on average. Hence, the likelihood for an out-of-order packet delivery, which cannot be compensated by InfiniBand's packet timeout/retry mechanism and therefore cause an application crash, is close to zero. In fact, we are unaware of any application crash caused by SAR on our HPC system, even though the system cannot use the full network update protocol discussed in Section 3.3 due to limitations in the HCA firmware.

4.3.4. Effects on MPI Traffic. The optimal latency between any two nodes in the system is not affected by our SAR approach, due to the fact that the underlying algorithm always calculates shortest paths. Therefore, any reconfiguration of the fabric with our Algorithm 1 will not increase the number of hops between nodes. Changes in the observable latency for small messages are possible, i.e., better or worse compared to algorithms for individual messages depending on the congestion in the system. However, from the two network metrics evaluated in Section 4.2, we can deduce that the likelihood of congestion is reduced and therefore the observable latency for small messages improves on average using SAR. Hence, we refrain from showing measured latency results, since the optimal latency does not change.

We conduct an MPI runtime measurement on our production system to showcase the impact of our scheduling-aware routing compared to other routing approaches. The benchmark emulates a bulk-synchronous application using an MPI_Alltoall collective operation followed by an MPI_Barrier to synchronize the time measurement. Every MPI process sends 1 MiB to every other process, and the

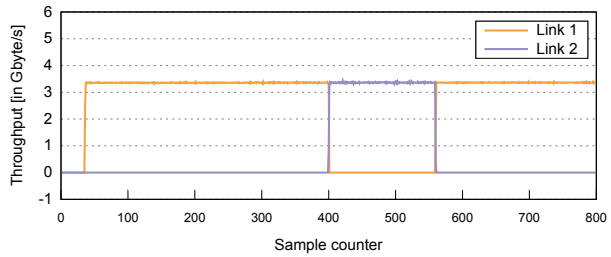


Figure 9. Visualization of our network update protocol (without QP draining) and path migration between 2 links on a testbed during high MPI load

runtime of the slowest process is written into a log file. This benchmark is scheduled to 28 nodes of one full-bisection fat-tree island of Taurus. We switched between three routing algorithms while the benchmark was running. Due to natural fragmentation of the production system, the 28 nodes ended up connected to ten different switches in the island.

Figure 8 summarizes the results: We ran the default fat-tree routing for the first three minutes. Then, we switched the subnet manager to standard DFSSSP routing, approximately between iteration 1,600 and 2,600. We observe a runtime increase of 7.1% due to the change from fat-tree routing to DFSSSP routing. For the last time segment, we started our SAR routing, which initiates the scheduling-aware routing for the whole system. The path optimizations by our scheduling-aware routing decreases the runtime for the benchmark by 17.6% compared to DFSSSP routing and 11.7% compared to fat-tree routing. We also plot the theoretical peak performance, assuming all data can be transferred without congestion and MPI library overhead. This shows that SAR essentially halves the congestion overhead caused by the highly optimized fat-tree routing, even on a regular fat-tree.

4.3.5. Property Preserving Network Updates on Testbed. Even so draining of the queue pairs currently does not work, as outlined in Section 4.1, we are able to successfully perform path migrations and update the routing configuration. To showcase that our network updates are safe in practice, we use a small test system, consisting of two IB QDR switches (connected by two links) and four nodes with IB FDR host channel adapters (two nodes connected to each switch). The modified OpenSM, which supports our network update protocol, is running on an administration node. We execute an MPI benchmark, which repeatedly performs MPI_Bcast with a 1 MiB send buffer, on two compute nodes to stress the fabric, and perform a scheduling-aware rerouting, see Figure 9. While running the benchmark, we query the performance counters of the inter-switch links approximately every 0.07 s, and use them to calculate the shown throughput. Additionally, we added an artificial delay of 10 s in OpenSM between sending unpath and repath traps, so that the path migration is easier to identify, see the area between sample 400 and 560. The path migration between the two links is clearly visible in Figure 9 and our update protocol has no negative impact on the throughput of the broadcast operation.

5. Related Work

The state-of-the-art routing method for supercomputers is static and flow-oblivious routing (e.g., [22], [35]), but a few alternatives, such as adaptive routing strategies (e.g., [32], [36], [37]) exist. Research shows that oblivious routing can result in local congestion, which increases application runtime, when the routes and communication pattern does not match properly [38]. Yet, its simplicity and practicality makes it the most-used routing mechanism, for example, in all InfiniBand and Ethernet networks. The objective of these flow-oblivious routings, such as the deadlock-free single-source shortest-path routing (DFSSSP) [6], Up*/Down* [5], or fat-tree routing [26], is to distribute the routes evenly across the interconnection network. However, this static and global balancing approach is only effective when the system is used by a single parallel application. SAR improves this flow-oblivious approach by including knowledge about the job-to-node mapping during the path calculation.

Optimizing communication performance for multi-user HPC system has been studied in the past resulting in many different approaches. Mellanox's proprietary traffic-aware routing, short TARA [39], monitors active applications and the network port counters, and attempts path adjustments when congestion is identified. However, TARA is a reactive optimization and depends on the time to collect and analyze all port counters. Furthermore, we are unaware of TARA's ability to enforce property preserving updates of the LFTs, hence out-of-order packet delivery is possible.

Application-aware routings, e.g., [9], [40], have been developed, but require detailed knowledge about traffic patterns and injection rates of each application. Additionally, this optimization approach has no notion of a timely behavior of the applications, and might assign too few links to each applications assuming they communicate simultaneously. In contrast, our scheduling-aware approach increases the network resources available to each application.

Optimizing job-to-node assignment via the batch system, while considering the network topology [41], [42], is feasible for regular networks, but assumes idealized routing or ignores it completely. Performing a routing-aware job-to-node mapping [43] by the batch job scheduler has been proposed as alternative. However, the computational complexity to solve the mapping problem is usually infeasible for an online scheduling of batch jobs, or fast, but imprecise, heuristics are used. In addition, this job-to-node mapping might increase the system's fragmentation, or it is hindered by the logical separation of the system through batch queues.

Multiple network update protocols have been developed in the past, but these are usually specific to certain technologies and their properties, such as for the broader gateway protocol (BGP) [44], [45] or for SDN networks [19], [20], to name but a few. Our property preserving network update is similar in this regard, but is customized for the conditions and requirements found in lossless InfiniBand fabrics.

6. Conclusion

The network of flow-oblivious routed supercomputers, e.g., the 47% InfiniBand-based systems of the Top500 list, suffers from underutilization when used simultaneously by many users, and application performance is reduced by local congestion in the network. Our scheduling-aware routing (SAR) approach for the interconnection network is able to mitigate these problems. As a result, the observable throughput for MPI-parallelized applications improves considerably, and is less sensitive to actual locality of the compute nodes within the supercomputer. Furthermore, the amount of dark fiber, i.e., installed but unused cables, is reduced.

We successfully integrated a filtering tool, periodically performing analyses of batch jobs, and a modified subnet manager (supporting SAR) into our IB-based petascale HPC system. Combining the knowledge of two different resource management domains has proven to be effective in theory, as well as in practice in over one year of usage on our system. We believe that SAR will be beneficial for other HPC centers which use obliviously routed networks, because of its low computational overhead and its transparency to parallel applications. Furthermore, the SAR approach could optimize pre-calculated alternatives for adaptive routing algorithms, theoretically reducing the flow migrations by the switches.

Changing the forwarding rules, not subject to error correction, within a interconnection network can cause unintended network packet delivery, such as out-of-order delivery, packet drops, and deadlocks, etc. Upper layer protocols, such as MPI, might fail, if a lossless interconnection technology is not able to handle these problems. Our proposed network update protocol for InfiniBand preserves required forwarding properties to ensure fault-free operation. Unfortunately, missing features within the hardware vendor's firmware prevent us currently from using this property preserving network update protocol. Nonetheless, we have not experienced any application crashes due to scheduling-aware reroutings of our HPC system, presumably because of resiliency features build into the transport layer of InfiniBand.

Acknowledgments

The authors would like to thank Prof. Nagel and his team for providing the batch job history of the Taurus HPC system installed at TU Dresden and for the opportunity to modify Taurus' routing algorithm over a longer period of time. Furthermore, we would like to thank Prof. Matsuoka and his team for giving us access to the batch job history of the Tsubame2.5 supercomputer located at the Tokyo Institute of Technology.

References

[1] P. Kogge, K. Bergman, and S. Borkar, "ExaScale Computing Study: Technology Challenges in Achieving Exascale Systems," University of Notre Dame, Department of Computer Science and Engineering, Notre Dame, Indiana, Tech. Rep. TR-2008-13, Sep. 2008.

[2] G. Rodriguez, C. Minkenberg, R. Beivide, R. P. Luijten, J. Labarta, and M. Valero, "Oblivious routing schemes in extended generalized Fat Tree networks," in *IEEE International Conference on Cluster Computing and Workshops, 2009 (CLUSTER '09)*, Aug. 2009, pp. 1–8.

[3] E. Zahavi, "Fat-tree Routing and Node Ordering Providing Contention Free Traffic for MPI Global Collectives," *J. Parallel Distrib. Comput.*, vol. 72, no. 11, pp. 1423–1432, Nov. 2012. [Online]. Available: http://dx.doi.org/10.1016/j.jpdc.2012.01.018

[4] W. J. Dally and C. L. Seitz, "Deadlock-Free Message Routing in Multiprocessor Interconnection Networks," *IEEE Trans. Comput.*, vol. 36, no. 5, pp. 547–553, 1987.

[5] M. D. Schroeder, A. Birell, M. Burrows, H. Murray, R. Needham, T. Rodeheffer, E. Satterthwaite, and C. Thacker, "Autonet: A High-speed, Self-Configuring Local Area Network Using Point-to-Point Links," *IEEE Journal on Selected Areas in Communications*, vol. 9, no. 8, Oct. 1991.

[6] J. Domke, T. Hoefler, and W. E. Nagel, "Deadlock-Free Oblivious Routing for Arbitrary Topologies," in *Proceedings of the 25th IEEE International Parallel & Distributed Processing Symposium (IPDPS)*. Washington, DC, USA: IEEE Computer Society, May 2011, pp. 613–624.

[7] J. C. Villanueva, T. Skeie, and S.-A. Reinemo, "Whitepaper: Routing and Fault-Tolerance Capabilities of the Fabriscale FM compared to OpenSM," Fabriscale, Tech. Rep., Jul. 2015. [Online]. Available: http://fabriscale.com/wp-content/uploads/2015/07/whitepaper_isc2015.pdf

[8] J. Flich, T. Skeie, A. Mejia, O. Lysne, P. Lopez, A. Robles, J. Duato, M. Koibuchi, T. Rokicki, and J. C. Sancho, "A Survey and Evaluation of Topology-Agnostic Deterministic Routing Algorithms," *IEEE Trans. Parallel Distrib. Syst.*, vol. 23, no. 3, pp. 405–425, Mar. 2012. [Online]. Available: http://dx.doi.org/10.1109/TPDS.2011.190

[9] M. A. Kinsy, M. H. Cho, T. Wen, E. Suh, M. van Dijk, and S. Devadas, "Application-aware deadlock-free oblivious routing," in *Proceedings of the 36th annual international symposium on Computer architecture*, ser. ISCA '09. New York, NY, USA: ACM, 2009, pp. 208–219. [Online]. Available: http://doi.acm.org/10.1145/1555754.1555782

[10] M. Palesi, R. Holsmark, S. Kumar, and V. Catania, "A Methodology for Design of Application Specific Deadlock-free Routing Algorithms for NoC Systems," in *Proceedings of the 4th International Conference on Hardware/Software Codesign and System Synthesis*, ser. CODES+ISSS '06. New York, NY, USA: ACM, 2006, pp. 142–147. [Online]. Available: http://doi.acm.org/10.1145/1176254.1176289

[11] B. Prisacari, G. Rodriguez, C. Minkenberg, and T. Hoefler, "Fast Pattern-Specific Routing for Fat Tree Networks," *ACM Trans. Archit. Code Optim.*, vol. 10, no. 4, pp. 36:1–36:25, Dec. 2013. [Online]. Available: http://doi.acm.org/10.1145/2555289.2555293

[12] H. Subramoni, S. Potluri, K. Kandalla, B. Barth, J. Vienne, J. Keasler, K. Tomko, K. Schulz, A. Moody, and D. K. Panda, "Design of a Scalable InfiniBand Topology Service to Enable Network-Topology-Aware Placement of Processes," in *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, ser. SC '12. Los Alamitos, CA, USA: IEEE Computer Society Press, 2012, pp. 70:1–70:12. [Online]. Available: http://dl.acm.org/citation.cfm?id=2388996.2389091

[13] A. Jokanovic, J. C. Sancho, G. Rodriguez, A. Lucero, C. Minkenberg, and J. Labarta, "Quiet Neighborhoods: Key to Protect Job Performance Predictability," in *2015 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. Hyderabad: IEEE, May 2015, pp. 449–459.

[14] J. Domke, T. Hoefler, and S. Matsuoka, "Fail-in-Place Network Design: Interaction between Topology, Routing Algorithm and Failures," in *Proceedings of the IEEE/ACM International Conference for High Performance Computing, Networking, Storage and Analysis (SC14)*, ser. SC '14. New Orleans, LA, USA: IEEE Press, Nov. 2014, pp. 597–608. [Online]. Available: http://dx.doi.org/10.1109/SC.2014.54

[15] ZIH, "Bull HPC-Cluster (Taurus)," 2016, https://doc.zih.tu-dresden.de/hpc-wiki/bin/view/Compendium/SystemTaurus.

[16] GSIC, "TSUBAME2 Hardware Architecture," 2016, http://tsubame.gsic.titech.ac.jp/en/hardware-architecture.

[17] M. C. Heydemann, J. Meyer, and D. Sotteau, "On Forwarding Indices of Networks," *Discrete Appl. Math.*, vol. 23, no. 2, pp. 103–123, May 1989. [Online]. Available: http://dx.doi.org/10.1016/0166-218X(89)90022-X

[18] W. Dally and B. Towles, *Principles and Practices of Interconnection Networks*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2003.

[19] M. Reitblatt, N. Foster, J. Rexford, C. Schlesinger, and D. Walker, "Abstractions for Network Update," in *Proceedings of the ACM SIGCOMM 2012 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication*, ser. SIGCOMM '12. New York, NY, USA: ACM, 2012, pp. 323–334. [Online]. Available: http://doi.acm.org/10.1145/2342356.2342427

[20] J. McClurg, H. Hojjat, P. Cerny, and N. Foster, "Efficient Synthesis of Network Updates," in *Proceedings of the 36th ACM SIGPLAN Conference on Programming Language Design and Implementation*, ser. PLDI 2015. New York, NY, USA: ACM, 2015, pp. 196–207. [Online]. Available: http://doi.acm.org/10.1145/2737924.2737980

[21] G. F. Pfister, "An Introduction to the InfiniBand Architecture," in *High Performance Mass Storage and Parallel I/O:Technologies and Applications*. Wiley-IEEE Press, 2002, pp. 616–632. [Online]. Available: http://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=5264600

[22] InfiniBand Trade Association, "InfiniBandTM Architecture Specification Volume 1 Release 1.3 (General Specifications)," Mar. 2015.

[23] K. S. Shim, M. H. Cho, M. Kinsy, T. Wen, M. Lis, G. E. Suh, and S. Devadas, "Static virtual channel allocation in oblivious routing," in *Proceedings of the 2009 3rd ACM/IEEE International Symposium on Networks-on-Chip*, ser. NOCS '09. Washington, DC, USA: IEEE Computer Society, 2009, pp. 38–43. [Online]. Available: http://dx.doi.org/10.1109/NOCS.2009.5071443

[24] T. Skeie, O. Lysne, and I. Theiss, "Layered Shortest Path (LASH) Routing in Irregular System Area Networks," in *IPDPS '02: Proceedings of the 16th International Parallel and Distributed Processing Symposium*. Washington, DC, USA: IEEE Computer Society, 2002, p. 194.

[25] "Mellanox OFED for Linux User Manual Rev. 2.0-3.0.0," Aug. 2013. [Online]. Available: http://www.mellanox.com/related-docs/prod_software/Mellanox_OFED_Linux_User_Manual_v2.0-3.0.0.pdf

[26] E. Zahavi, G. Johnson, D. J. Kerbyson, and M. Lang, "Optimized InfiniBand fat-tree routing for shift all-to-all communication patterns," *Concurr. Comput. : Pract. Exper.*, vol. 22, no. 2, pp. 217–231, Feb. 2010. [Online]. Available: http://dx.doi.org/10.1002/cpe.v22:2

[27] T. Hoefler, T. Schneider, and A. Lumsdaine, "Optimized Routing for Large-Scale InfiniBand Networks," in *17th Annual IEEE Symposium on High Performance Interconnects (HOTI 2009)*, Aug. 2009.

[28] A. B. Yoo, M. A. Jette, and M. Grondona, "SLURM: Simple Linux Utility for Resource Management," in *Job Scheduling Strategies for Parallel Processing: 9th International Workshop, JSSPP 2003, Seattle, WA, USA, June 24, 2003. Revised Paper*, D. Feitelson, L. Rudolph, and U. Schwiegelshohn, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2003, pp. 44–60. [Online]. Available: http://dx.doi.org/10.1007/10968987_3

[29] Message Passing Interface Forum, "MPI: A Message-Passing Interface Standard Version 3.1," Jun. 2015. [Online]. Available: http://www.mpi-forum.org/docs/mpi-3.1/mpi31-report.pdf

[30] E. Gabriel, G. E. Fagg, G. Bosilca, T. Angskun, J. J. Dongarra, J. M. Squyres, V. Sahay, P. Kambadur, B. Barrett, A. Lumsdaine, R. H. Castain, D. J. Daniel, R. L. Graham, and T. S. Woodall, "Open MPI: Goals, Concept, and Design of a Next Generation MPI Implementation," in *Proceedings, 11th European PVM/MPI Users' Group Meeting*, Budapest, Hungary, Sep. 2004, pp. 97–104.

[31] R. L. Henderson, "Job Scheduling Under the Portable Batch System," in *Proceedings of the Workshop on Job Scheduling Strategies for Parallel Processing*, ser. IPPS '95. London, UK, UK: Springer-Verlag, 1995, pp. 279–294. [Online]. Available: http://dl.acm.org/citation.cfm?id=646376.689372

[32] M. S. Birrittella, M. Debbage, R. Huggahalli, J. Kunz, T. Lovett, T. Rimmer, K. D. Underwood, and R. C. Zak, "Intel Omni-path Architecture: Enabling Scalable, High Performance Fabrics," in *2015 IEEE 23rd Annual Symposium on High-Performance Interconnects (HOTI)*. Santa Clara, CA: IEEE, Aug. 2015, pp. 1–9.

[33] J. Domke, T. Hoefler, and S. Matsuoka, "Routing on the Dependency Graph: A New Approach to Deadlock-Free High-Performance Routing," in *Proceedings of the 25th ACM International Symposium on High-Performance Parallel and Distributed Computing*, ser. HPDC '16. New York, NY, USA: ACM, 2016, pp. 3–14. [Online]. Available: http://doi.acm.org/10.1145/2907294.2907313

[34] T. Schneider, O. Bibartiu, and T. Hoefler, "Ensuring Deadlock-Freedom in Low-Diameter InfiniBand Networks," in *Proceedings of the 2016 IEEE 24rd Annual Symposium on High-Performance Interconnects*, ser. HOTI '16. IEEE Computer Society, Aug. 2016.

[35] Y. Ajima, T. Inoue, S. Hiramoto, and T. Shimizu, "Whitepaper: Tofu: Interconnect for the K computer," Fujitsu, Tech. Rep. Vol. 48, No. 3, Jul. 2012.

[36] B. Alverson, E. Froese, L. Kaplan, and D. Roweth, "Whitepaper: Cray XC Series Network," Cray Inc., Tech. Rep. WP-Aries01-1112. [Online]. Available: http://www.cray.com/sites/default/files/resources/CrayXCNetwork.pdf

[37] B. Arimilli, R. Arimilli, V. Chung, S. Clark, W. Denzel, B. Drerup, T. Hoefler, J. Joyner, J. Lewis, J. Li, N. Ni, and R. Rajamony, "The PERCS High-Performance Interconnect," in *2010 IEEE 18th Annual Symposium on High Performance Interconnects (HOTI)*, Aug. 2010, pp. 75–82.

[38] T. Hoefler, T. Schneider, and A. Lumsdaine, "Multistage Switches are not Crossbars: Effects of Static Routing in High-Performance Networks," in *Proceedings of the 2008 IEEE International Conference on Cluster Computing*. IEEE Computer Society, Oct. 2008.

[39] Mellanox Technologies, "Product Brief: Unified Fabric Manager Software," Mar. 2016. [Online]. Available: http://www.mellanox.com/related-docs/prod_management_software/PB_UFM_InfiniBand.pdf

[40] X. Zhong and V. M. Lo, "Application-Specific Deadlock Free Wormhole Routing on Multicomputers," in *Proceedings of the 4th International PARLE Conference on Parallel Architectures and Languages Europe*, ser. PARLE '92. London, UK, UK: Springer-Verlag, 1992, pp. 193–208. [Online]. Available: http://dl.acm.org/citation.cfm?id=646421.693827

[41] T. Hoefler and M. Snir, "Generic Topology Mapping Strategies for Large-scale Parallel Architectures," in *Proceedings of the 2011 ACM International Conference on Supercomputing (ICS'11)*. Tucson, AZ: ACM, Jun. 2011, pp. 75–85.

[42] H. Yu, I.-H. Chung, and J. Moreira, "Topology Mapping for Blue Gene/L Supercomputer," in *Proceedings of the 2006 ACM/IEEE Conference on Supercomputing*, ser. SC '06. New York, NY, USA: ACM, 2006. [Online]. Available: http://doi.acm.org/10.1145/1188455.1188576

[43] A. H. Abdel-Gawad, M. Thottethodi, and A. Bhatele, "RAHTM: Routing Algorithm Aware Hierarchical Task Mapping," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, ser. SC '14. Piscataway, NJ, USA: IEEE Press, 2014, pp. 325–335. [Online]. Available: http://dx.doi.org/10.1109/SC.2014.32

[44] S. Raza, Y. Zhu, and C.-N. Chuah, "Graceful Network State Migrations," *IEEE/ACM Trans. Netw.*, vol. 19, no. 4, pp. 1097–1110, Aug. 2011. [Online]. Available: http://dx.doi.org/10.1109/TNET.2010.2097604

[45] P. Francois, O. Bonaventure, B. Decraene, and P. A. Coste, "Avoiding disruptions during maintenance operations on BGP sessions," *IEEE Transactions on Network and Service Management*, vol. 4, no. 3, pp. 1–11, Dec. 2007.