Softening Up the Network for Scientific Applications

Celio Trois*, Luis C. Bona*, Marcos D. Del Fabro*, Magnos Martinello†,
Sarvesh Bidkar‡, Reza Nejabati‡, Dimitra Simeonidou‡
*Computer Science Department, Federal University of Parana, Curitiba, Brazil, Email: {ctrois, bona, didonet}@inf.ufpr.br
†Computer Science Department, Federal University of Espirito Santo, Espirito Santo, Brazil, Email: magnos@inf.ufes.br
‡High Performance Networks Group, University of Bristol, Bristol, United Kingdom,
Email: {sarvesh.bidkar, reza.nejabati, dimitra.simeonidou}@bristol.ac.uk

***Abstract*—Scientific applications demand huge computational power connected through fast networks. They are developed using parallel kernel methods, usually implemented with the Message Passing Interface (MPI), presenting well-behaved communication patterns across computing nodes. The current network technologies do not allow defining traffic forwarding policies considering the different application traffic, resulting in an unbalanced load on the network links. Moreover, the devices are not concerned if the traffic is latency-sensitive or bandwidth-intensive. To handle this, we present NetSA, a framework exploiting the communication patterns of scientific applications, considering latency and bandwidth constraints, as the key logic for evenly placing the application flows on the network available paths. Through NetSA, the scientific application developer can easily modify the network behavior to best fit the application communication requirements. We have performed experiments for optimizing the MPI communication primitives and applied our solution to speed up scientific applications, obtaining an execution time reduction up to 27%.**

I. Introduction

Scientists are using scientific applications (SciApps) to create and predict complex phenoms, for example, weather forecasting, prediction of natural disasters, bacterial profiling, animal genotyping, and so forth. Often, these applications have requirements such as model deterministic or stochastic, the existence of multiple simultaneous dependent phenomena, resolution, complexity, and ensemble size. The developer must evaluate these requirements and apply one or more computational methods (or kernels) to implement the scientific software. These kernels usually present well-behaved patterns to exchange data across the computing nodes.

For more accurate results, SciApps are increasingly demanding computational power, being executed in dedicated clusters connected through extremely fast networks for moving the huge amounts of generated data. A programming approach based on communications has become crucial, so the Message Passing Interface (MPI) library was born with the efforts of many industrial and academic, leading to a de facto standard for developing parallel and distributed programs [1].

In general, the performance of parallel and distributed computation is greatly affected by communication due to network congestion points or bottlenecks [24]. To tackle these bottlenecks, a multiple-path network usually provides more than one path between any pair of computing nodes. Link Aggregation Control Protocol (LACP, IEEE 802.3ad) and Equal-Cost Multi-Path (ECMP, IEEE 802.1Qbp) are used for load-balancing the communications across these paths.

These protocols apply a hash function on some packet header fields for choosing the output port to forward packets. The problem is that two or more long-lived flows can collide on their hash and end up on the same output port, creating a bottleneck, overwhelming the switch buffers and degrading the overall switch performance [2]. Furthermore, these protocols do not consider the different characteristics of network paths, where some may provide higher bandwidth while others exhibit lower latency.

Software-Defined Networking (SDN) has emerged to support new possibilities for network management, decoupling control and forwarding functions and enabling the network to become directly programmable according to the user requirements [20]. The existing works using SDN for forwarding communications through multiple paths use information collected from end-hosts or network devices [2], [6], [12]. To our knowledge, this is the first work using the well-behaved patterns expressed by SciApps to forward their communication flows through multiple network paths, considering latency and bandwidth requirements.

In this paper, we present NetSA, a framework to optimize the software-defined **Net**work for **S**cientific **A**pplications. NetSA exploits the communication patterns of SciApps as the key logic for balancing the application communications on the network paths. NetSA provides an API allowing the SciApp developer to tune the network forwarding according to his application. The framework keeps a database of communication patterns, annotated with constraints on latency-sensitive and bandwidth-intensive communications. When NetSA receives an API call, it identifies the pattern in its database, placing the communications according to existing constraints: latency-sensitive are forwarded through low-latency paths and bandwidth-intensive via higher bandwidth links.

The rest of this paper is structured as follows. Section II presents a brief literature review and related works, Section III shows the NetSA implementation details, and Section IV reports the impacts of the network topology on applications communications. Section V shows the experiments using our approach for balancing the Message Passing Interface (MPI)

collective communications and optimizing parallel kernels and SciApps; finally, the conclusion is reported in Section VI.

II. Background and Related Works

In this section, we explain some SDN concepts, bringing up related works where SDN was applied for optimizing user applications and SciApps. We also report proposals using the SciApps communication patterns to improve the performance.

SDN aims to overcome the conventional network limitations; in the existing pre-SDN network infrastructure, each device must be individually configured, eventually causing misconfigurations and errors. Furthermore, each device has a specific purpose, for example, switches are used to connect computers to a Local Area Network (LAN), routers interconnect several LANs and provide Internet connection, and firewalls filter out unwanted packets. SDN allows the network devices functionalities to be defined or modified after being deployed. It presents an open architecture that makes the network configuration and management flexible and programmable, enabling new features to be added without changing the network hardware [20].

OpenFlow [14] has been adopted by enterprises and academy, becoming a de facto SDN protocol for programming the SDN devices. The packet forwarding is performed considering flows, using multiple packet headers fields instead of just destination addresses. OpenFlow-enabled switches maintain flow-tables with entries containing a matching pattern, actions, and priorities. When a network device receives a new packet, it searches its flow tables looking for an entry that matches the packet header fields. If multiple matching entries are found, the one with the highest priority is used, and its actions are performed on all packets belonging to that flow (e.g. forward to an output port). If no match is found, the device forwards the packet to the network controller. The controller manages the switches flow tables by adding, modifying, or removing their entries, and querying flows statistics. By creating matching on flows, it is possible to accommodate network traffic through different physical topology paths; even traffic from the same parallel application.

Many proposals are using SDN to modify the forwarding for improving the application performance. B4 [12] allows to define application priority and uses multipath routing/tunneling to optimize link usage. Mahout [6] detects elephant flows at the end hosts and uses placement algorithms to compute paths for them. NoF [21] raises the network programming level by allowing the application specialists to program the network through high-level constructs. U-Chupala et al. [22] have proposed a bandwidth and latency aware routing to improve the overall network performance.

We have also seen works modifying the network topology for increasing the throughput for reducing the latency. Vassoler et al. [23] have used fast magneto-optical switches for modifying the physical topology in server-centric datacenters. Fujiwara et al. [9] have used free-space optics that transforms a laser light in an optical cable to a laser beam in the air, and vice versa on top of the cabinets to reduce both end-to-end network latency and total cable length.

Related to SciApps, Takahashi et al. [18] have implemented some SDN enhanced MPI primitives; they have developed an MPI *Bcast* (broadcast) for eliminating duplicate packets in the network and an MPI *Allreduce* that makes a communication plan to forward the reducing through distinct paths. The MPI directives were also modified to dynamically adjust the computing resources based on the SciApp requirements [10]. Date et at. [7] have discussed the integration of SDN with high-performance computing (HPC) infrastructure, reporting some specific HPC proposals for dealing with computing relocation, bandwidth, and latency. Finally, in a previous work [19], we studied the impacts of the SDN programmability on SciApps.

SciApps are usually implemented using kernels such as Dense Linear Algebra, Sparse Linear Algebra, Structured or Unstructured Grids, and Fast Fourier Transforms. These kernels present well-behaved communication patterns used for designing new chip-multiprocessors topologies, network-on-chip, encompassing coherence protocols [3], thread mapping [8], and improving application performance [16], [5].

Our proposal differs from the existing works by using the programmability introduced by SDN to dynamically adjusting the SciApps communications on the network paths, considering their traffic patterns annotated with latency and bandwidth constraints.

III. NetSA: Network for Scientific Applications

NetSA is a framework to program the software-defined **Net**work for **S**cientific **A**pplications. It uses the well-behaved communication patterns expressed by SciApps as the basis for balancing the applications flows through the available network paths. It is composed of a Network Programming Module (NPM), a Topology Information Module (TIM), a Communication Pattern Database (CPD), and an Application Programming Interface (API). Figure 1 shows the NetSA framework components and their interactions.

A. Communication Patterns

The Communication Pattern Database (CPD) stores the computing nodes communication as traffic matrices. Each matrix cell keeps the amount of traffic (packets and bytes) transmitted between every pair of ingress and egress nodes. NetSA provides an option for measuring and recording the communication patterns; in this operation, the switches flow table statistics are collected and based on source and destination addresses, NetSA computes the number of transmitted bytes and packets, storing this information in XML format. The recording operation is performed only once per each different traffic matrix; for all the upcoming executions, NetSA uses the information stored in the CDP.

The traffic matrix may also keep, in each cell, the communication constraints, allowing the SciApp developer to inform which communications are latency-sensitive or bandwidth-intensive. This information may also be set through latency

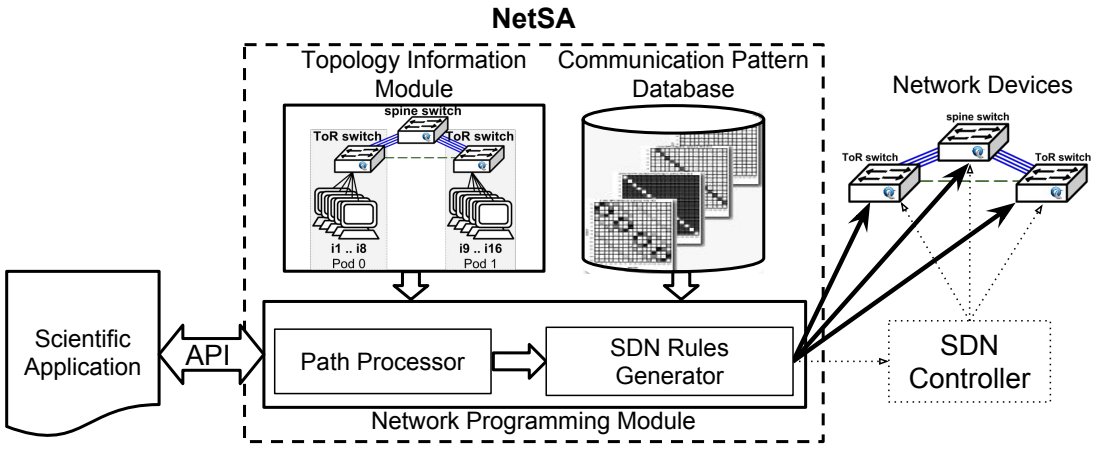


Fig. 1: NetSA Framework.

and bandwidth thresholds (in bytes per packet). NetSA classifies the cells as bandwidth-intensive if their average packet size (APS) values ($\frac{total\ number\ of\ bytes}{total\ number\ of\ packets}$) are greater than the defined bandwidth threshold; similarly, the cells with APS values lower than the latency threshold are classified as latency-sensitive.

```
<?xml version="1.0" encoding="UTF-8"?>
<traffic_matrix>
    <matrix id="100" name="MPI_Allreduce">
        ...
        <src_host>i9
            <dst_host>i1
                <num_bytes>42752734</num_bytes>
                <num_packets>35346</num_packets>
                <bandwidth intensive="true"></bandwidth>
            </dst_host>
            <dst_host>i2
                <num_bytes>9394</num_bytes>
                <num_packets>81</num_packets>
                <latency max_num_hops="2"></latency>
            </dst_host>
        ...
        </src_host>
        ...
    </matrix>
</traffic_matrix>
```

Fig. 2: MPI_Allreduce XML traffic matrix containing latency and bandwidth constraints.

We will use the MPI_Allreduce primitive to explain NetSA; this MPI function combines values from all nodes and distributes the result back to all nodes. Figure 2 shows an example of XML, generated by NetSA, for storing the traffic matrix for this primitive. The communication from *i9* to *i1* is classified as bandwidth-intensive, so the NetSA should forward it through bandwidth-intensive paths, while from computing node *i9* to node *i2* the communication is classified as latency sensitive and has to be allocated to a path with no more than two hops.

A graphical representation of the MPI_Allreduce traffic matrix is shown in Figure 3. In this figure, the bandwidth threshold was configured to 1100 and the latency set to 120. The bandwidth-intensive cells are shown in blue (dark gray in B&W), the cells in green (light gray in B&W) are defined as latency-sensitive, the gray ones have no constraints. The

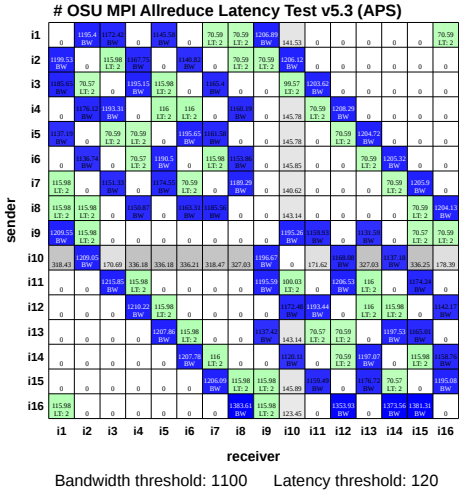


Fig. 3: MPI_Allreduce traffic matrix Average Packet Size (bytes/packet) graphical representation.

cells are normalized to the highest value; darker cells indicate higher values and white cells no communication occurred.

B. Topology Information Module

The Topology Information Module (TIM) keeps the topology information such as the network links, the computing nodes locations, and network devices and ports. In this module, the links can also be manually annotated as latency-sensitive and bandwidth-intensive.

Although in the current version of NetSA the topology is manually informed, it is possible to obtain and compute this information through the SDN controller or the Link Layer Discovery Protocol (LLDP)[1].

C. NetSA API

NetSA provides a simple API allowing the developer to inform which pattern the application is going to communicate.

[1]http://www.ieee802.org/1/files/public/docs2002/lldp-protocol-00.pdf

This API includes a single function (*int net_set_pattern(int pattern_id);*) receiving the *pattern_id* and returning 1 if the network was successfully programmed or 0 on error.

```
...
if(net_set_pattern(100)){ // mpi_allreduce
   MPI_Allreduce(sendbuf, recvbuf, size, MPI_FLOAT, MPI_SUM,
                 MPI_COMM_WORLD);
} else {
   printf("Error: could not program the network.");
}
...
```

Fig. 4: Example of calling NetSA API.

Figure 4 shows an example where the API is called to program the network for the *MPI_Allreduce* communication primitive (*pattern_id=100*).

D. Network Programming Module

The Network Programming Module (NPM) is responsible for interpreting the NetSA API calls and generating the SDN messages for programming the network devices. This module includes a Path Processor Module (PPM) and a SDN Rules Generator Module (RGM).

Upon receiving an API call, the NPM uses the argument *pattern_id* to read the traffic matrix from CPD, identifying which nodes have exchanged information, as well as their latency and bandwidth constraints. The topology information is acquired from TIM. With this information, NPM allocates the nodes communications among the network paths and sends messages for programming the network devices with these forwarding rules. Due to the well-behaved communication patterns expressed by the SciApps, and to avoid the time-consuming task of continuously reading the network state, NetSA uses the stored bandwidth and latency information.

PPM first generates low-priority rules for communicating all-to-all computing nodes; preventing any pair of nodes to be incommunicable. So, the higher-priority rules are created, using Dijkstra weighted shortest-path [22] for allocating the traffic matrix cells identified as bandwidth-intensive. Finally, the latency-sensitive communications are accommodated, with the highest priority, on the latency annotated links. Whenever a communication is established in a path, its weight is increased.

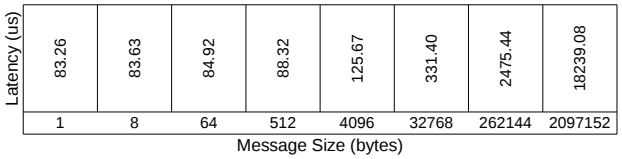
After the communication being placed, the RGM creates the OpenFlow messages for programming the network devices' flow tables. In the implemented version, the switches are directly programmed through OpenFlow protocol, but this module can be modified for using any controller's API.

For understanding how throughput and latency can be affected by the network topology, we have analyzed the communications for different message sizes, being exchanged through paths presenting a different number of hops. This investigation is presented in the next section.

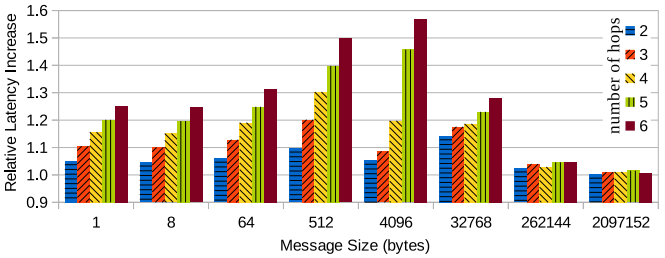
IV. Characterizing the Impact of Path Length on Communications

To understand the impact of network topology on throughput and latency, we have studied the MPI point-to-point communication primitives, using the OSU Micro-Benchmarks (OMB)[2] for observing the variations in latency and throughput when the traffic was forwarded through paths varying from one to six hops. We have chosen this maximum value because the current datacenter network topologies are designed with a maximum of six hops between any pair of nodes [25].

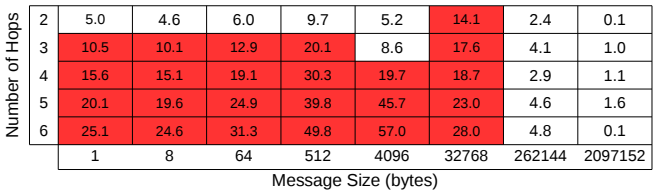
We have executed all point-to-point latency tests (*osu_latency*, *osu_latency_mt*, and *osu_multi_lat*), but due to the similarity of the results, we only present the values of *osu_latency* test. This test is carried out in a ping-pong fashion; the sender sends a message with a certain data size to the receiver and waits for a reply from it. The receiver receives the message from the sender and sends back a reply with the same data size.



(a) Latency measured for a single hop.



(b) Relative latency increase.



(c) Percentage increase in latency.

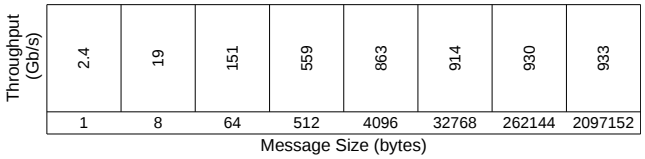
Fig. 5: Latency for different hop count and message size.

Figure 5a shows the latency values (in micro-seconds) for the different tested message sizes when the communication is performed through a single hop (only one switch). Figure 5b shows the relative increase in latency, compared to the single hop value, varying the number of hops from two to six.

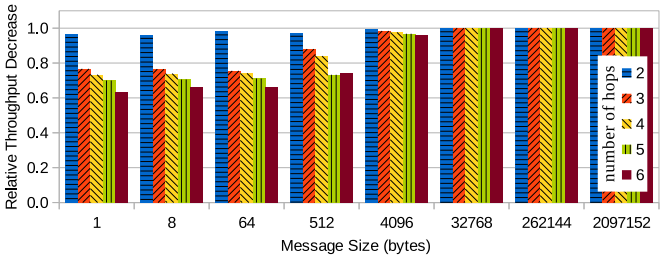
Figure 5c shows the percentage increase in latency, where the cells in red represent an increment higher than 10%. This figure facilitates the increased latency visualization; for example, considering an application with latency-sensitive communication, tolerating an increase up to 10% in latency, and transmitting messages with 512 bytes. In this case, the application traffic MUST be forwarded through a path with a maximum of two hops.

[2]http://mvapich.cse.ohio-state.edu/benchmarks/

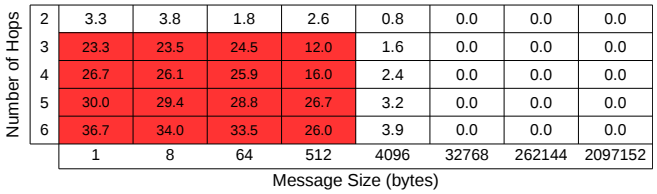
Just as happened with the latency tests, the values obtained for all OMB bandwidth tests were similar, so we show only the results of *osu_bw*. This test has a sender sending out a fixed number of back-to-back messages and waiting for a reply from a receiver. The receiver sends the reply only after receiving all these messages. The bandwidth is calculated based on the elapsed time and the number of bytes sent by the sender.



(a) Throughput values for a single hop.



(b) Relative throughput decrease.



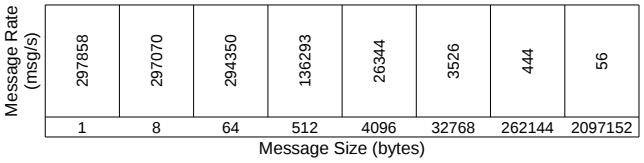
(c) Percentage decrease in throughput.

Fig. 6: Throughput for different hop count and message size.

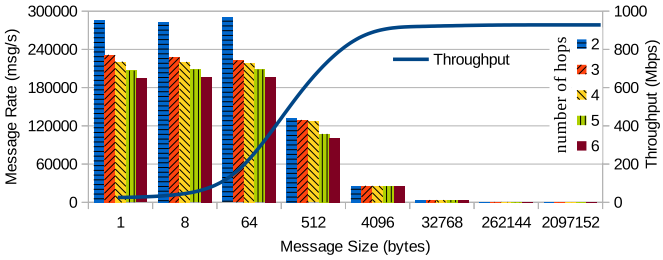
Figure 6a shows the measured throughput (GB/s) for transmitting the different messages size through one hop and Figure 6b plots relative decrease in throughput for hop variation. Figure 6c present the percentage increase compared to the single hop values. For messages larger than 4096 bytes, the link becomes saturated, and throughput stops decreasing. We can also see that for messages larger than 2048 bytes, the throughput percentage increase is lower than 10%, even if they are forwarded through six hops. Thus, we conclude that the number of hops does not significantly affect bandwidth-intensive communications.

We have also executed the OMB Multiple Bandwidth / Message Rate test (osu_mbw_mr); this test evaluates the aggregate bandwidth and message rate between multiple pairs of processes. Each of the sending processes sends a fixed number of messages back-to-back to the paired receiving process before waiting for a reply from the receiver.

In Figure 7a, it can be seen the message rate (msg/sec) for different message size forwarded through a single hop. There is a large variability on the throughput for small messages $[1, 8, 64, 512]$, varying from $[2.4, 19, 151, 559]$ Mbps respectively, as expected. Whereas for larger messages ($\geq$ 4096 bytes), it achieves the capacity of the link, decreasing the message rate variation. Figure 7b shows that the message rate falls sharply when small messages ($\geq$ 64 bytes) are forwarded through a higher number of hops. It is possible to see that for these messages, through one hop, the message rate is around 300,000 msg/sec; however, when the test is performed via six hops, this rate decreases to less than 200,000 msg/sec. Thus, if the SciApp depends on exchanging a huge number of small messages, then it is mandatory to send them along paths with the least number of hops.



(a) Message rate values for a single hop.



(b) Message rate variation chart.

Fig. 7: Message rate for different hop count and message size.

V. Evaluation

For evaluating our proposal, we have performed three different experiments. In the first experiment, we have used NetSA for improving the MPI collective primitives by balancing their communications through the network paths. In the second experiment, our approach is applied for reducing the execution time of NAS Parallel Benchmarks (NPB) [13], and in the third experiment, we have optimized the network for SciApps implemented using the OpenLB library [11]. In order to obtain meaningful and realistic results, the experiments were executed in a real testbed, described in the next section.

As a baseline, we have measured all experiments with a single hop topology, meaning that all computers were connected to a single switch. We have also measured the experiments with network devices configured with Link Aggregation Control Protocol (LACP) for aggregating the Ethernet interfaces into a single logic interface. Lastly, the switches were programmed with NetSA.

A. Testbed

Our testbed is composed of 16 Lenovo PCs with processor Intel quad-core 3.2Ghz, 8GB RAM, 1TB HD, 1 Gigabit Ethernet, running Linux Debian 8.2, and MPI mpich-3.2. Three Pica8 P-3290 OpenFlow switches running the operating system PicOS v2.6.4. Each switch has 48 Gigabit Ethernet ports, four 10 Gigabit optical SFP+ ports, supporting OpenFlow 1.4

through Open vSwitch v2.0 integration (*http://openvswitch.org/*). The experiments were performed using OSU Micro-Benchmarks (OMB) v5.3, NAS Parallel Benchmarks v3.3.1, and OpenLB v1.0.

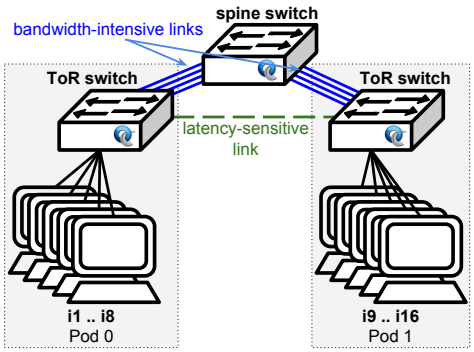


Fig. 8: Testbed topology annotated with bandwidth and latency information.

The testbed topology is shown in Figure 8; it is composed of one spine and two top-of-rack (ToR) switches. The sixteen computing nodes (*i1..i16*) are divided between the two ToR switches. The spine switch is connected to the ToR through four links, annotated as bandwidth-intensive (identified in blue). The topology also includes a link connecting the ToR switches; this link is annotated as latency-intensive (in green).

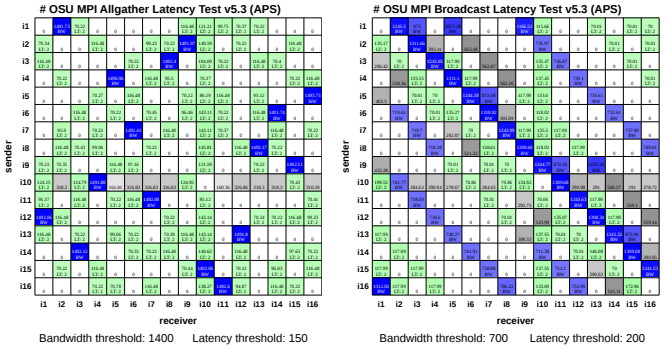
B. Collective Communications

MPI collective primitives are used in most applications, being responsible for a significant fraction of the communication time [15]. We have selected the three OMB benchmarks that most exchanged data through the network to conduct this experiment (MPI_Allgather, MPI_Allreduce, and MPI_Bcast). These benchmarks measure the average latency of collective operations across their processes, for various message lengths, over a large number of iterations. They were executed in our evaluation testbed using 16 processes (one process per computer). For increasing the amount of traffic transmitted through the spine switch, when evaluating the primitive MPI_Allgather, the computing nodes were informed in a random order.

Before running NetSA, we have measured and stored these benchmarks traffic matrices, annotating some cells as bandwidth-intensive and latency-sensitive, as shown in Figure 3, Figure 9a, and Figure 9b.

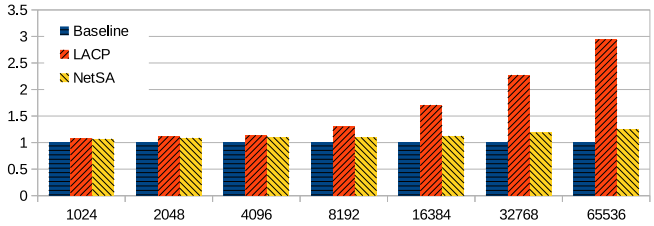
We have tested different thresholds, choosing those that provided the lowest average latency for each test. The allocation of bandwidth-intensive traffic (in blue) had a significant impact on results, meaning that, when they were properly balanced, the benchmarks have achieved a latency time close to the baseline. However, for the evaluated MPI primitives, allocating the latency-sensitive communications on the latency link (in green) did not influence the result significantly. This happened because these communications are dependent on bandwidth-intensive messages.

Figure 10a shows the OSU MPI_Allreduce latency test normalized to the baseline. It is possible to see that, when

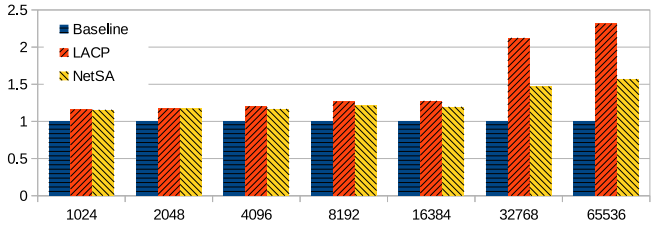


(a) MPI_Allgather (random). (b) MPI_Bcast.

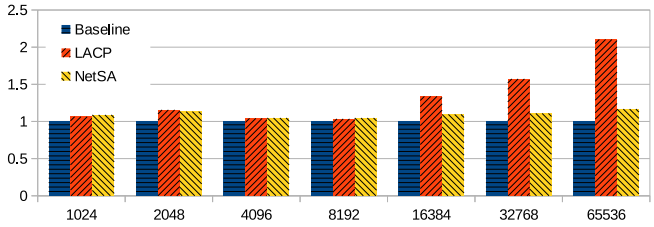
Fig. 9: MPI collective primitives traffic matrices with cells annotated with bandwidth and latency constraints.



(a) MPI_Allreduce latency test relative increase.



(b) MPI_Allgather latency test relative increase.



(c) MPI_Bcast latency test relative increase.

Fig. 10: OMB MPI collective latency tests.

the network was configured using LACP, for messages larger than 8192 bytes, there is a significant increase in the average latency, taking nearly 200% longer than the baseline for the largest message. When the network devices were programmed with NetSA, in the worst case, the average latency took 25% more than the baseline.

The Figure 10b and Figure 10c present the relative latency increase for MPI_Allgather and MPI_Bcast respectively. Both charts show a considerable increase for message larger than 8192 bytes. In both tests, in the worst case, LACP has

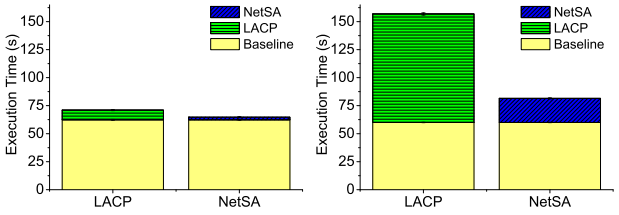
increased latency time more than twice, while NetSA has increased 57% and 17% for MPI_Allgather and MPI_Bcast.

These results show that a proper allocation of communication is fundamental for reducing the latency, and consequently, reducing the overall execution time of MPI operations.

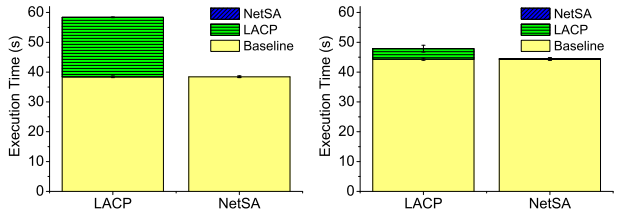
C. Parallel Benchmarks

We have modified four tests from NAS Parallel Benchmarks (NPB), including calls to NetSA API, for programming the network according to their communication patterns. NPB are set of programs designed to help evaluate the performance of parallel supercomputers, consisting of five kernels and three pseudo-applications. For this experiment, we have also used the topology described in Section V-A.

We have selected the benchmarks that most exchanged information among their computation nodes: *bt*, *cg*, *ft*, and *lu*. The *bt* is a pseudo application used to compute block tridiagonal matrices; *cg* is a Sparse Linear Algebra kernel used to compute an approximation to the smallest eigenvalue of a large, sparse, symmetric positive definite matrix. The *ft* kernel solves a 3-D partial differential equation using Fast Fourier Transform, and *lu* is a pseudo application that uses iterative methods for solving linear systems.



(a) NPB *bt* pseudo application. (b) NPB *ft* parallel kernel.



(c) NPB *cg* parallel kernel. (d) NPB *lu* pseudo application.

Fig. 11: NAS Parallel Benchmarks execution times.

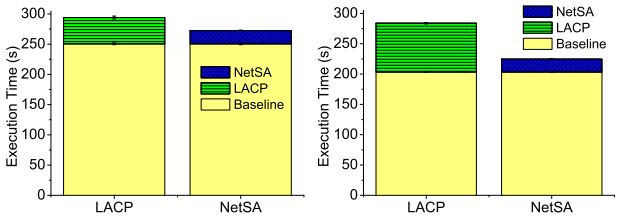
These benchmarks were executed 30 times and their execution times were recorded; these values are shown in Figure 11. For Figure 11 and Figure 12, the baseline comprises the computation time coupled with the communication time through a single hop. The overhead imposed by the topology and the manner the communications are balanced on the links by LACP and NetSA are displayed on the top of each bar.

For *cg* and *lu*, the execution time using NetSA was close to the baseline; the increment for *bt* and *ft* was respectively 4% and 36%. When the flows were distributed through LACP, the baseline times for *bt*, *cg*, *ft*, and *lu* incremented respectively 14%, 52%, 161%, and 8%.

To confirm that the LACP was not equally dividing the communications among the paths, we analyzed the switch ports statistics when running the *ft* kernel and we realized that three ports had transmitted approximately 3 Gigabytes and the other port forwarded merely 2.6 Megabytes. With NetSA, all ports have transmitted around 2.2 Gigabytes.

D. Scientific Applications

In addition to the experiments on NPB, we have modified two scientific applications, including calls to NetSA API, for amending the forwarding according to their communication patterns. The first application examines a steady flow past a 3D cylinder placed in a channel. The cylinder is offset somewhat from the center of the flow to make the steady-state symmetrical flow unstable [17]. The second application implements a backward facing step, being used to simulate flows through rough-walled rock fractures [4]. Both applications were implemented using the OpenLB library [11].



(a) 3D cylinder. (b) 3D backward facing step.

Fig. 12: Scientific applications execution times.

The 3D cylinder application had a 8% increase in the execution time when the flows were placed with NetSA and 17% with LACP. By examining the ports of switches, we realize that again LACP did not balance the communications; some ports have transmitted about 11 Gigabytes, while other ports only 3.7 Gigabytes. The execution time of the second application has increased about 27% when the communications were placed with LACP, comparing with NetSA.

For these applications, as well as for the *ft* kernel, even when NetSA properly distributed the communications on the existing links, the overhead was considerable. This happened because the amount of information exchanged between the two Pods was higher than the capacity of existing links. This problem could be mitigated by including new bandwidth links from ToR switches to the spine switch.

We have also measured the overhead introduced by NetSA API calls. Among the tested applications, the time for programming the switch flow tables, on average, was 0.47 seconds. This overhead is slightly higher for applications that need to install more rules. For instance, the *bt* application demanded the installation of 48 rules, which is twice the number of rules compared to other applications. The average time for programming the rules for *bt* was around 0.54 seconds.

These experiments have shown that even in the case of extremely simple topology, SciApps performance is highly affected by unequal loads in the network paths. However, NetSA provides a way to program the network to meet

the pattern communications to be suitably balanced through network paths. Moreover, NetSA allows more agility for setting up routes/flows on the underlying physical network topology, exploring the existing redundant paths for flows that are throughput oriented or providing shortest paths for those that need low latency guarantees.

VI. Conclusion

This paper presents NetSA, a framework for placing the scientific applications communications through the network available paths. Our approach relies on (i) storing the application traffic matrices and annotating them with bandwidth and latency constraints; (ii) providing an API to enable the scientific application developer easily modify the network forwarding for his needs, and (iii) using this information for balancing the communications on the network available paths. These points allowed NetSA to allocate flows with the SDN centralized logical view, improving SciApps performance.

The experiments demonstrate the effectiveness of our approach in a real testbed by improving the MPI collective operations and accelerating up to 27% the execution time of scientific applications.

As future work, we intend to provide a scheme for detecting the communication patterns and automatically optimize the network for them. We also plan to enable NetSA for running (or scheduling) multiple concurrent applications, as well as, propose a strategy for optimizing the computing nodes placement, based on these patterns.

Acknowledgment

The authors would like to thank CAPES for partial funding of this research, CNPq under Grant 456143/2014-9, and the Brazilian Ministry of Communications for partial funding it via "Digital Inclusion: Technology for Digital Cities" project. Also, the research has been partially funded by the European Commission H2020 program under grant agreement no. 688941 (FUTEBOL), as well from the Brazilian Ministry of Science, Technology, Innovation, and Communication (MCTIC) through RNP, CTIC and FAPES (under grants no. 0410/2015 and no. 0444/2015).

References

[1] S. Achour and W. Nasri, "A performance prediction approach for mpi routines on multi-clusters," in *2012 20th Euromicro International Conference on Parallel, Distributed and Network-based Processing*. IEEE, 2012, pp. 125–129.

[2] M. Al-Fares, S. Radhakrishnan, B. Raghavan, N. Huang, and A. Vahdat, "Hedera: Dynamic flow scheduling for data center networks," in *Proceedings of the 7th USENIX Conference on Networked Systems Design and Implementation*, ser. NSDI'10. Berkeley, CA, USA: USENIX Association, 2010, pp. 19–34.

[3] N. Barrow-Williams, C. Fensch, and S. Moore, "A communication characterisation of splash-2 and parsec," in *International Symposium on Workload Characterization (IISWC)*. IEEE, 2009, pp. 86–97.

[4] S. Briggs, B. W. Karney, and B. E. Sleep, "Numerical modelling of flow and transport in rough fractures," *Journal of Rock Mechanics and Geotechnical Engineering*, vol. 6, no. 6, pp. 535–545, 2014.

[5] L. Chen, X. Huo, and G. Agrawal, "A pattern specification and optimizations framework for accelerating scientific computations on heterogeneous clusters," in *International Parallel and Distributed Processing Symposium (IPDPS)*. IEEE, May 2015, pp. 591–600.

[6] A. R. Curtis, W. Kim, and P. Yalagandula, "Mahout: Low-overhead datacenter traffic management using end-host-based elephant detection," in *Proceedings INFOCOM*. IEEE, April 2011, pp. 1629–1637.

[7] S. Date, H. Abe, D. Khureltulga, K. Takahashi, Y. Kido, Y. Watashiba, U. Pongsakorn, K. Ichikawa, H. Yamanaka, E. Kawai *et al.*, "An empirical study of sdn-accelerated hpc infrastructure for scientific research," in *2015 International Conference on Cloud Computing Research and Innovation (ICCCRI)*. IEEE, Oct 2015, pp. 89–96.

[8] M. Diener, E. Cruz, L. Pilla, F. Dupros, and P. O. Navaux, "Characterizing communication and page usage of parallel applications for thread and data mapping," *Performance Evaluation*, vol. 88, pp. 18–36, 2015.

[9] I. Fujiwara, M. Koibuchi, T. Ozaki, H. Matsutani, and H. Casanova, "Augmenting low-latency hpc network with free-space optical links," in *2015 IEEE 21st International Symposium on High Performance Computer Architecture (HPCA)*, Feb 2015, pp. 390–401.

[10] G. Galante and L. C. Bona, "Supporting elasticity in openmp applications," in *2014 22nd Euromicro International Conference on Parallel, Distributed, and Network-Based Processing*. IEEE, 2014, pp. 188–195.

[11] V. Heuveline and J. Latt, "The openlb project: an open source and object oriented implementation of lattice boltzmann methods," *International Journal of Modern Physics C*, vol. 18, no. 04, pp. 627–634, 2007.

[12] S. Jain, A. Kumar, S. Mandal, J. Ong, L. Poutievski, A. Singh, S. Venkata, J. Wanderer, J. Zhou, M. Zhu *et al.*, "B4: Experience with a globally-deployed software defined wan," in *ACM SIGCOMM Computer Communication Review*, vol. 43, no. 4. ACM, 2013, pp. 3–14.

[13] H. Jin, H. Jin, M. Frumkin, M. Frumkin, J. Yan, and J. Yan, "The openmp implementation of nas parallel benchmarks and its performance," NASA Technical Report NAS-99-011, Tech. Rep., 1999.

[14] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "Openflow: enabling innovation in campus networks," *ACM SIGCOMM Computer Communication Review*, vol. 38, no. 2, pp. 69–74, 2008.

[15] R. Rabenseifner, "Automatic mpi counter profiling of all users: First results on a cray t3e 900-512," in *Proceedings of the message passing interface developers and users conference*, vol. 1999, 1999, pp. 77–85.

[16] E. Rubin, E. Levy, A. Barak, and T. Ben-Nun, "Maps: Optimizing massively parallel applications using device-level memory abstraction," *ACM Trans. Archit. Code Optim.*, vol. 11, no. 4, pp. 1–22, Dec. 2014.

[17] M. Schäfer, S. Turek, F. Durst, E. Krause, and R. Rannacher, *Benchmark Computations of Laminar Flow Around a Cylinder*. Wiesbaden: Vieweg+Teubner Verlag, 1996, pp. 547–566.

[18] K. Takahashi, D. Khureltulga, B. Munkhdorj, Y. Kido, S. Date, H. Yamanaka, E. Kawai, and S. Shimojo, "Concept and design of sdn-enhanced mpi framework," in *Software Defined Networks (EWSDN), 2015 Fourth European Workshop on*. IEEE, 2015, pp. 109–110.

[19] C. Trois, L. C. E. de Bona, M. D. D. D. Fabro, and M. Martinello, "Carving Software-Defined Networks for Scientific Applications with SpateN," in *41st Conference on Local Computer Networks (LCN)*. IEEE, 2016, pp. 606–610.

[20] C. Trois, M. D. D. D. Fabro, L. C. E. de Bona, and M. Martinello, "A survey on sdn programming languages: Towards a taxonomy," *IEEE Communications Surveys Tutorials*, vol. PP, no. 99, pp. 1–25, April 2016.

[21] C. Trois, M. Martinello, L. C. E. de Bona, and M. D. Del Fabro, "From software defined network to network defined for software," in *Proceedings of the 30th Annual ACM Symposium on Applied Computing*, ser. SAC '15. ACM, 2015, pp. 665–668.

[22] P. U-Chupala, K. Ichikawa, H. Iida, N. Kessaraphong, P. Uthayopas, S. Date, H. Abe, H. Yamanaka, and E. Kawai, "Application-oriented bandwidth and latency aware routing with open flow network," in *Cloud Computing Technology and Science (CloudCom), 2014 IEEE 6th International Conference on*, Dec 2014, pp. 775–780.

[23] G. L. Vassoler, F. R. de Souza, P. P. P. Filho, and M. R. N. Ribeiro, "Hybrid reconfiguration for upgrading datacenter interconnection topology," in *IEEE Photonics Conference 2012*, Sept 2012, pp. 782–783.

[24] Y. Watashiba, S. Date, H. Abe, Y. Kido, K. Ichikawa, H. Yamanaka, E. Kawai, and S. Shimojo, "Efficacy analysis of a sdn-enhanced resource management system through nas parallel benchmarks," *The Review of Socionetwork Strategies*, vol. 8, no. 2, pp. 69–84, 2014.

[25] F. Yao, J. Wu, G. Venkataramani, and S. Subramaniam, "A comparative analysis of data center network architectures," in *International Conference on Communications (ICC)*. IEEE, 2014, pp. 3106–3111.