2016 IEEE 41st Conference on Local Computer Networks

Carving Software-Defined Networks for Scientific Applications with SPATEN

Celio Trois, Luis Bona, Marcos D. Fabro
Univ. Federal do Parana (UFPR) - Curitiba - Brazil
Email: {ctrois, bona, didonet}@inf.ufpr.br

Magnos Martinello
Univ. Federal do Espirito Santo (UFES) - Espirito Santo - Brazil
Email: magnos@inf.ufes.br

***Abstract*—Scientific applications (SciApps) are broadly used in all science domains. For more accurate results, they have been increasingly demanding computational power and extremely agile networks. These applications are usually implemented using numerical methods presenting well-behaved patterns to exchange data across its computing nodes. This paper presents SPATEN, a tool that exploits the spatial communication patterns of SciApps as the fundamental logic to drive the network programming. SPATEN classifies the SciApps nodes communications and balances the elephant flows across the available network paths. As a proof of concept, we carried out a set of experiments in real testbeds, demonstrating that network programming may affect the performance of SciApps significantly. Also, a balanced flow allocation can speed up SciApps to near-optimal execution times.**

I. INTRODUCTION

Scientists have been executing scientific applications (SciApps) in different domains, including bioinformatics, astrophysics, weather forecasting, and genome research, among others. These applications are implemented with parallel and distributed programming, exchanging huge amounts of data among their computing nodes [2].

The wide majority of SciApps is implemented using well-known numerical methods, so-called dwarfs [3]. A computational dwarf can be defined as "a pattern of communication and computation common across a set of applications." They have been used for optimizing the communication on networks-on-chip, GPUs, multicore processors, multiprocessor architectures, and user application performance [9], [4].

The performance of SciApps is highly dependent on the computing nodes interconnection bandwidth. However, the common assumption is that these applications run on a fixed number of computing nodes [5] where the network has been considered as a static resource, working as a connectivity service that can not be controlled or modified.

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 [11]. We have found in literature proposals using SDN to improve specific user applications [6], [12]. However, to the best of our knowledge, no existing work routes the data flows through the network using the communication patterns for improving the performance of SciApps.

In this work, we propose SpateN, a tool that exploits the **S**patial **PAT**terns as the key logic to **E**nhance the **N**etwork programming. SPATEN classifies the elephant flows and balances them across the available network paths. We assume that the computing nodes have well-behaved communication patterns, meaning that a given application executed in a set of nodes has a strong trend to transmit the same amount of data across the same nodes. Also, we consider that these applications run on a dedicated cluster where SPATEN has full access to the programmable switches.

As a proof of concept, we carried out a set of experiments in real testbeds to demonstrate that (i) communication patterns are fundamental to achieve the network programmability demanded by SciApps; (ii) SPATEN is able to overcome the challenges introduced by SDN; and (iii) a balanced network load allows to keep near-optimal SciApps execution time.

The rest of this paper is structured as follows. Section II presents a brief literature review, while in Section III, our approach named SPATEN is described. Section IV reports our experiments and finally, the conclusion and future works are reported in Section V.

II. BACKGROUND

In this section, a characterization of SciApps communication patterns is presented. Also, we discuss some SDN benefits, issues, and related works.

The SciApps are usually implemented using known numerical methods that present well-behaved communication and computation patterns, so-called computational dwarfs [3], also named motif or kernel. The dwarfs are broadly used for designing new chip-multiprocessors communication topologies, network-on-chip, and thread mapping aiming to speed up user applications. The communication patterns can be characterized by their spatial behaviors, indicating where the nodes are located in the network topology and how much information they have exchanged. These characteristics can be used as input to modify the Software-Defined Network for balancing the communication across its paths.

One benefit brought by SDN is the ability to perform the forwarding operation considering flows (multiple packet headers fields) and not just destination addresses. Thus, it is possible to accommodate traffic from different applications through different physical topology paths aiming to optimize applications [12], [6].

DOI 10.1109/LCN.2016.42

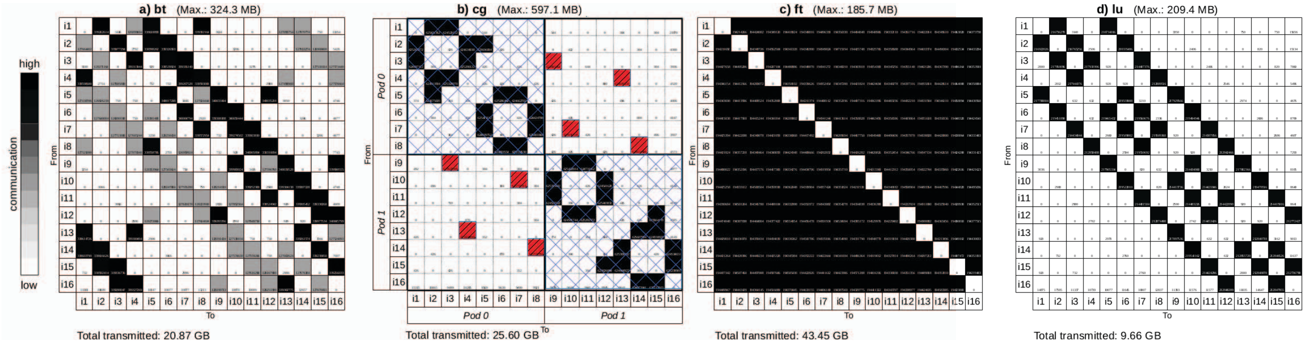


Fig. 1: Spatial behavior of four scientific applications: *bt*, *cg*, *ft*, and *lu*.

While SDN brought improvements for the networks, it also posed new problems. One issue introduced by SDN is the time to populate the switches forwarding tables. When an SDN-enabled device receives a new packet, if no match is found, the device forwards the packet to the controller (reactive approach); the controller manages the switch flow tables by adding, modifying, or removing the entries. Querying the controller reactively is time-consuming, so to deal with this issue, some SDN programming languages [11] rely on proactive approaches for installing the rules on switches ahead of time.

Another issue brought by SDN is the time taken for finding the matching rules at the switch flow tables. The number of table entries is increased because SDN enables specifying matching on specific flows using multiple packet header fields (microflows). To alleviate this problem, SDN includes wildcard matching rules. Ternary Content Addressable Memories (TCAMs) are being used for speeding up wildcarding table lookup operations. Although fast, TCAM is expensive and power-hungry, so SDN vendors are using a combination of TCAM and SRAM or DRAM, but they are slower than TCAMs for wildcarding.

Our approach uses the programmability introduced by SDN along with SciApps spatial behavior information for balancing the application flows through the network paths, avoiding both issues shown above.

III. SpateN: SPATial Network Programming

SpateN is a tool that uses spatial information to program the network for SciApps. For running an application, the scientist (user) must inform the application and the number of computing nodes. Based on previously stored spatial behaviors, SpateN generates the rules, proactively installs them on the SDN switches, and when the network is ready, it starts the application.

For developing SpateN, we have investigated the communication behavior of NAS Parallel Benchmarks [7] applications, selecting those that most exchanged information among the computing nodes: bt, cg, ft, and lu. Their spatial behaviors were recorded and stored in the SpateN database. The network topology annotated with links bandwidth and latency is also stored in its database.

A. Spatial Behavior

Spatial behavior can be formalized as a traffic matrix M_B, where each position $M_B[i][j]$ holds the number of bytes transmitted from node i to node j. SpateN has an option for measuring and recording the application traffic matrix; in this operation, SpateN installs forwarding rules matching both, source and destination addresses, on all top-of-rack (ToR) switches before executing the application. After the application has finished, SpateN reads the flow table statistics from ToR switches and computes its traffic matrix. We have opted to create an option for reading/storing the traffic matrix, but it could also be done online [8].

Figure 1 shows the spatial behavior of chosen applications. It is possible to see that they feature different communication patterns; *ft* exchanges almost the same amount of data across all nodes, transmitting a maximum of 185.7 MB through a pair of nodes. *cg* is the application that most exchanged data, considering the pairs of nodes (597.1MB); however, as we can see on its spatial behavior matrix, only a few pairs of nodes have communicated. Figure 1 also presents the total amount of data transmitted from all nodes.

For graphical visualization, we have normalized the matrix to its maximum value, showing it in a gray gradient, where cells in black are the most communicative pair of nodes, and white indicates that no communication happened.

B. Classifying the Communication

This section presents details of how spatial behavior is used to classify communication. Traffic flows are typically classified as either short-lived (mice flows) or throughput-bound (elephant flows) [10]; SpateN uses the spatial behavior to detect the pairs of hosts exchanging elephant flows.

The *cg* application was chosen to explain the communication classification. The matrices are divided into Pods where each Pod is the set of computers connected to the same ToR switch. Figure 1b shows the *cg* spatial behavior matrix divided in Pods, considering two Pods of eight nodes. The nodes in the same Pod (crosshatched) are not classified because their communications occur within the ToR switch. The remaining matrix cells are labeled as *unclassified*.

To classify the cells, we have used the Muhammad et al. approach [1], where the flow is classified as an elephant whenever it is consuming 10% of the link bandwidth per second. In Figure 1b, the cells identified in red with thin diagonal lines have been classified as elephants; the rest of the cells remained *unclassified*.

C. Routing

The Dijkstra's weighted shortest-path algorithm [8] is used for placing the previously classified elephant flows. SPATEN finds a path and creates the rules (*R1*) for switches, with a higher priority, matching source (i) and destination (j) addresses.

After placing the elephant flows, SPATEN computes the paths among all hosts using an approach similar to Mice-Trap [10], reducing the number of matching rules by grouping the flows by the destination address. These rules (*R2*) are created with a lower priority.

SPATEN proactively fills the switches flow tables with *R1* and *R2* before starting the application, avoiding both problems reported in Section II.

IV. EVALUATION

To evaluate our proposal, the experiments were structured into two parts. The first part is devoted to understanding *network programmability impact on the SciApps performance*. In the second part, the experiments investigate the feasibility of *accelerating applications by balancing their elephant flows*.

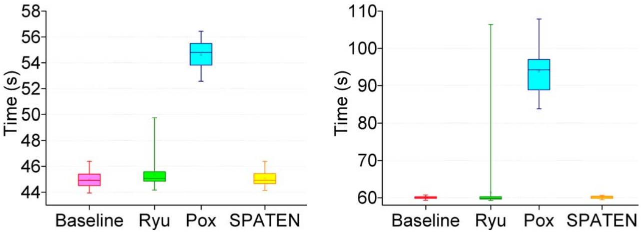
For all experiments, the applications were executed 30 times and their execution times were recorded. To avoid the pitfalls introduced by simulation and emulation tools, and be sure that the obtained results are correct and accurate, all experiments were executed in a real testbed. As the baseline, we have firstly measured all experiments with the switches configured as L2/L3 mode[1], using the simplest possible topology: all computers connected to a single switch.

Our testbed was 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 implementation 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, and a Firebolt3 chipset supporting up to 2048 flow entries in its TCAM memory. This switch can operate in two modes of operation: L2/L3 mode and Open vSwitch (OVS) mode. The OVS mode supports OpenFlow 1.4, through Open vSwitch v2.0 integration[2]. The evaluation was performed using the NAS parallel benchmarks v3.3.1 [7].

A. Impact of Network Programmability

To understand the impact of network programmability, we have used a single switch programmed with SPATEN, Ryu[3], and Pox[4], two well-known SDN controllers, forwarding the flows with their default reactive learning switch. We have executed the most rule-intensive applications, *ft* and *lu*, using 16 computers, measuring their execution time and investigating the installed matching rules. Figure 2a shows the execution times for *lu* application.

When the switch is programmed with Ryu controller, the application execution time was close to the baseline. However, the first execution time was higher due to the time for querying the controller. When controlled by Pox, the application took longer to execute; this is explained because the controller is creating and installing rules for every new flow (microflow). Furthermore, Pox installs the rules using expiration timeouts. When these timeouts expire, the rules are removed and the controller has to be queried again. With SPATEN, the execution time achieves the baseline, because it proactively installs the necessary rules before starting the application.



(a) *lu* execution time. (b) *ft* execution time.

Fig. 2: Execution time of *ft* and *lu* applications executed in 16 computers connected to a single switch.

Figure 2b shows the measured times for *ft*. When controlled with Ryu the application execution average time was 1.4 seconds slower. However, in the first execution it took 106.4 seconds (77% longer than the baseline) to install the all-to-all nodes matching rules. When the switch was controlled by Pox, the *ft* execution time was much higher, taking an average of approximately 94 seconds to finish its execution. The application execution time using SPATEN was similar to the baseline.

We note that the Ryu installs the matching rules using source and destination MAC addresses. So, for the *ft* application, it has installed 240 rules on the switch flow table. On the other hand, Pox has ranged from 117 to 334 rules. The higher number is because Pox creates microflow rules, and the idle and hard timeouts were responsible for the variation. The number of rules installed by SPATEN was 16 because they match only the destination addresses.

An important remark is the scalability regarding network states for installing rules based on the $\{$source, destination$\}$ tuple. It implies that the necessary number of rules grows exponentially and can be calculated as $n \times (n-1)$, where n is the number of nodes. Considering 48 computing nodes connected to all P-3290 Ethernet ports and a controller installing rules for communicating all-to-all nodes, a total of 2256 flow entries

---

[1] Layer 2 / Layer 3: The switch runs as a non-SDN switch.

[2] http://openvswitch.org/

[3] https://osrg.github.io/ryu/

[4] https://github.com/noxrepo/pox

will be necessary, exceeding the 2048 entries TCAM size. In our testbed switches, we have observed that the RTT for a ping message goes from 0.3ms for rules stored in TCAM to 4ms when they are located in DRAM. The throughput goes from 936Mbits/s when rules are stored in TCAM to only 4Mbits/s when stored in DRAM.

B. *Programming the Network with* SPATEN

In order to prove that SPATEN can optimize the SciApps, we have used the two applications that exchanged more traffic considering the pair of nodes (*cg* and *bt*). They were executed in 16 computers connected through the topology shown in Figure 3. The topology is composed of three switches, one spine and two top-of-rack (ToR) switches. Each ToR switch have eight computers connected to its Gigabit Ethernet ports, and they are connected to the spine through four Gigabit links.

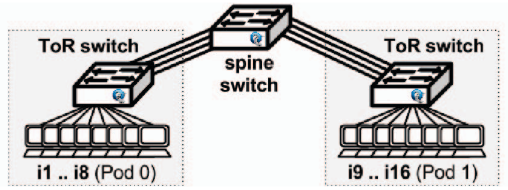
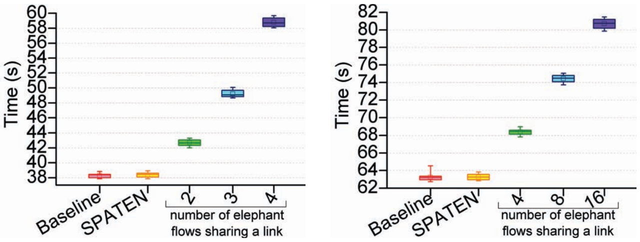


Fig. 3: Real testbed for SPATEN proof of the concept.

To assess the outcome, we compared the application execution time programmed with SPATEN against a single switch in L2/L3 mode (baseline). We also measured the execution time when the application flows were unbalanced on the available links.



(a) *cg* execution time.

(b) *bt* execution time.

Fig. 4: Execution time of *cg* and *bt* applications executed in 16 computers connected to the given topology.

Figure 4a shows the measured execution time of this experiment for the *cg* application, while Figure 4b illustrates the *bt* application. For both applications, the execution time had a very small increase, 100 milliseconds on average, when the network was programmed with SPATEN, compared to the baseline. On the other hand, when elephant flows were allocated to share a link, the execution time increased considerably. For instance, *cg* execution time doubles in Figure 4a, when four elephant flows are sharing a link.

V. CONCLUSION

We have presented SPATEN, a tool that improves the performance of scientific applications (SciApps) by taking advantage of their well-behaved communication patterns as the main insight for programming the network.

Our approach relies on the application of spatial behavior to classify the elephant flows, proactively allocating these flows in a balanced way along network paths, eliminating the time for querying the controller, and reducing the number of installed matching rules. Our experiments demonstrate the effectiveness of our approach in keeping the execution time of SciApps to near-optimal times in a real testbed.

In future work, we intend to apply SPATEN to scientific workflows with multiple execution phases, as well as explore the possibility to run (or schedule) multiple concurrent applications.

ACKNOWLEDGMENTS

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.

REFERENCES

[1] M. Afaq, S. Rehman, and W.-C. Song, “Large flows detection, marking, and mitigation based on sflow standard in sdn,” *Journal of Korea Multimedia Society Vol*, vol. 18, no. 2, pp. 189–198, 2015.

[2] S. Ahern, S. R. Alam, M. R. Fahey, R. J. Hartman-Baker, R. F. Barrett, R. A. Kendall, D. B. Kothe, R. T. Mills, R. Sankaran, A. N. Tharrington *et al.*, “Scientific application requirements for leadership computing at the exascale,” Oak Ridge National Laboratory (ORNL); Center for Computational Sciences, Tech. Rep., 2007.

[3] K. Asanovic, R. Bodik, B. C. Catanzaro, J. J. Gebis, P. Husbands, K. Keutzer, D. A. Patterson, W. L. Plishker, J. Shalf, S. W. Williams *et al.*, “The landscape of parallel computing research: A view from berkeley,” Technical Report UCB/EECS-2006-183, EECS Department, University of California, Berkeley, Tech. Rep., 2006.

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

[5] R. d. R. Righi, V. F. Rodrigues, C. A. da Costa, G. Galante, L. C. E. de Bona, and T. Ferreto, “Autoelastic: Automatic resource elasticity for high performance applications in the cloud,” *IEEE Transactions on Cloud Computing*, vol. 4, no. 1, pp. 6–19, Jan 2016.

[6] 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.

[7] 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.

[8] J. Ru, S. Wei, and Z. Hongke, “Traffic matrix-based routing optimization,” in *Proceedings of the 2015 International Conference on Computer Science and Intelligent Communication*, 2015, pp. 429–432.

[9] 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.

[10] R. Trestian, G. M. Muntean, and K. Katrinis, “Micetrap: Scalable traffic engineering of datacenter mice flows using openflow,” in *2013 IFIP/IEEE International Symposium on Integrated Network Management (IM 2013)*, May 2013, pp. 904–907.

[11] 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.

[12] 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.