DiG: Data centers in the Grid

Hardik Soni, Damien Saucez, Thierry Turletti

DiG: Data centers in the Grid

Hardik Soni
Inria
Sophia Antipolis, France
Email: hardik.soni@inria.fr

Damien Saucez
Inria
Sophia Antipolis, France
Email: damien.saucez@inria.fr

Thierry Turletti
Inria
Sophia Antipolis, France
Email: thierry.turletti@inria.fr

***Abstract*—We are witnessing a considerable amount of research work related to data center and cloud infrastructures but evaluations are often limited to small-scale scenarios as very few researchers have access to a real infrastructure to confront their ideas to reality. In this demo we will reveal our experiment automation tool, DiG (*Data centers in the Grid*), which explicitly allocates physical resources in grids to emulate data center and cloud networks. DiG allows one to utilize grid infrastructures to evaluate research ideas pertaining to data centers and cloud environments at massive scale and with real traffic workload. We have automated the procedure of building target network topologies while respecting available physical resources in the grid against the demand of links and hosts in the experiment. We will present a showcase where DiG automatically builds a large data center topology composed of hundreds of servers executing various Hadoop intensive workloads.**

I. Introduction

Most SDN experiments having data center and cloud scenarios are performed with traffic traces using emulators (e.g., Mininet [1], Maxinet [2]) or simulators (e.g. ns-3 [3]) due to restricted access to real production environments of companies like Amazon, Google, or Facebook. Therefore, experiment results may be biased or noisy due to modeling techniques of simulators or unaccounted and excessive usage of physical resources in case of emulation.

Many tools exist like Mininet [1] and Maxinet [2] for running SDN experimentations. Among them, Maxinet is the closest to our work. However, it targets scalable emulation to create SDN enabled data center environments and relies on synthetic traffic generation models. Maxinet is built using Mininet, which has the capability to run real world applications to generate traffic. However, at the scale of hundreds of hosts, running such applications on emulated hosts consume computing resources and hinders emulation's scaling capability of network experiments. Neither Maxinet nor Mininet provides guarantee on allocation of computing power (i.e., CPU cores) for emulated hosts to scale the entire experiment with a minimum amount of physical resources.

In this work, we aim (1) to create data center topologies while respecting computing and network resource constraints and (2) to run real world data center applications on top of it. Since very few researchers have access to production data center environments and the majority of them has access to grid computing environments like Grid5000 [4], the primary goal of our system is to build test environments for SDN enabled data center and cloud networks in grid physical infrastructures. With DiG (*Data centers in the Grid*) we can build overlay experimental networks by explicitly allocating available physical resources, like CPU and link capacity, to the requirements of experimental network topologies, allowing to run real world data center applications on top of a grid with performance guarantees.

II. System Description

A. Overlay Experimental Network with Resource Guarantee

The DiG system is able to create experimental networks that carry real traffic between nodes running protocol stacks as in real world data center networks. To achieve this, DiG implements a layer 2 overlay network on a grid infrastructure. Having a layer 2 overlay network as an experimental network provides a bare-metal network environment to the SDN controllers and switches.

DiG instantiates data center servers and switches by running virtual machines (VMs) and OpenFlow enabled switches on grid nodes. While creating an overlay network, it is important to take into account the available computing power of the grid nodes and the physical link capacity between each pair of grid nodes. In most of the cases, the physical network connectivity along with the computing power of the grid nodes are known by the experimenters.

B. DiG Technical Description

DiG maps the experimental network on a physical grid network while satisfying the computing power requirements of all the nodes in the experimental network and not exceeding the computing capacity of grid nodes. Similarly, layer 2 overlay links are mapped by satisfying the demand of all the links in experimental topology while not overloading physical links. So, the problem is reduced to the resolution of a Virtual Network Embedding (VNE) [5] problem with constraints on nodes computing power and links capacity.

DiG runs in three phases to implement experimental networks on grid infrastructures. Each phase generates an output in the form of text files and these files are used in the next phase as an input. This makes the system more flexible and facilitates modifications and integration of different phases implementations. The names of the three phases are *Experimental Network Embedding*, *Configuration Generation*, and *Deployment*. Note that each phase can be run in an independent way with appropriate input files, without the need of executing other phases.

1) Experimental Network Embedding: DiG solves a VNE problem using the ALEVIN [6] framework, which is used to generate the mapping between the experimental network and the grid physical infrastructure. ALEVIN is fed with the

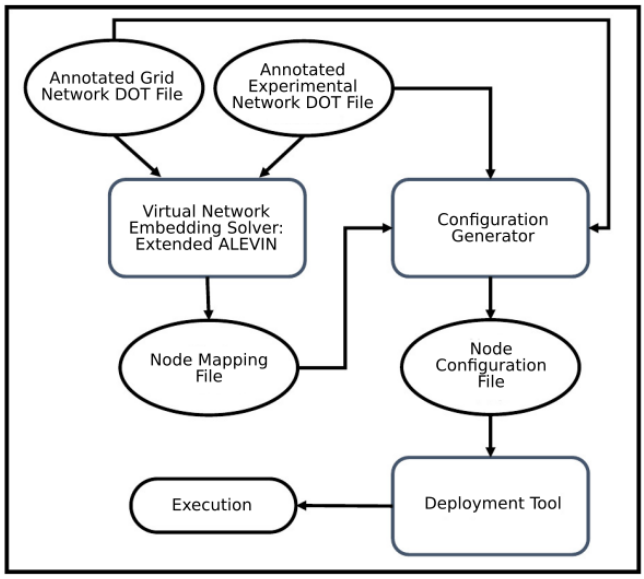

Fig. 1. DiG Module interaction

experimental and grid networks described in DOT language [7] in a text file. The experimental network is annotated with CPU cores requests for nodes, link capacity requests and other application-specific attributes like Hadoop node type for automation usage. The grid network is annotated with CPU core capacity for grid nodes, link bandwidth capacity for physical links and IP addresses for automation purpose. ALEVIN uses CPU cores and link bandwidth attributes for both experimental and grid networks and it generates a node mapping file as shown in Fig. 1. The node mapping file is a text file that identifies the set of experimental network nodes mapped on each physical node.

2) Configuration Generator for Experimental Network: The Configuration Generator phase takes as input the mappings generated from the Experimental Network Embedding phase along with the descriptions of the experimental and grid networks in DOT format. However, the mapping file can be generated by any means, and not necessarily with the technique presented in Sec. II-B1. This allows (1) running different tools and algorithms for the network embedding step and (2) relaxing the strong dependency on the performance of embedding algorithms.

The Configuration Generator phase prepares the configuration files for each physical host based on the mapping. It contains the meta-data to instantiate the mapped part of the experimental network on physical hosts. The meta-data contains the appropriate commands and the parameters to instantiate the virtual machines and to map the virtual hosts to the physical nodes. It also contains the necessary information (e.g., source-destination UDP port numbers, IPs of grid nodes, tunnel unique IDs etc.) to create layer 2 tunneling protocol (i.e., L2TPv3) endpoints and links capacity information satisfying the experimental network bandwidth demand based on the mapping. Along with the experimental network configuration files, this phase generates files to bring up basic network utility (e.g., assigning IP to experimental network interface, routing etc.) in the hosts.

3) Deployment of Experimental Network: The last phase consists of the deployment of the experimental network using configuration files on the physical machines. DiG instantiates the virtual hosts in the experimental network on grid nodes, creates OpenFlow [8] switches interconnected with L2TPv3 tunnels and controls the link bandwidth according to the requirements of the experimental network to emulate. It is also responsible to launch applications on virtual hosts of the experimental network. The Linux Traffic Control utility (*tc*) is used to control the bandwidth at the tunnel interfaces according to the links capacity requirements of the experimental network.

C. Management Network

As mentioned above, the deployment phase launches applications in virtual hosts. DiG uses a designated node called *Manager node* in the grid infrastructure to launch the deployment phase in a centralized way. All the communications required for deployment purpose and management of experimental network are carried out on a dedicated management network isolated by experimental networks, as depicted in Fig. 2.

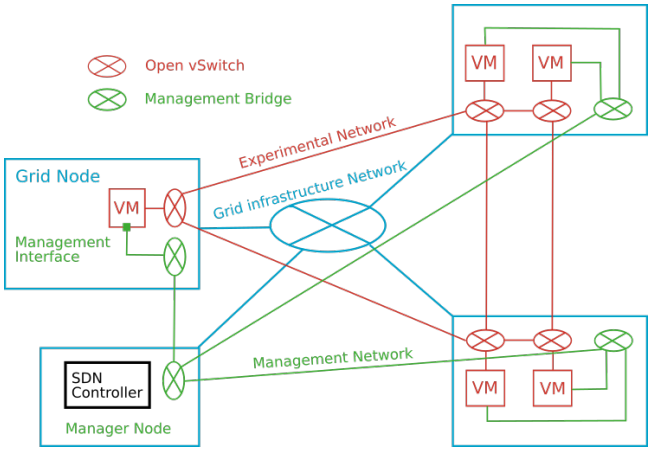

Fig. 2. Experimental Overlay Network with Management Network

Each virtual host in the experimental network includes a management network interface. A management bridge is created on all the grid nodes including the manager node, as shown in Fig. 2. The virtual hosts of the experimental network running on a grid node are connected to the management bridge on the grid node through their management interface. The management bridge on each grid node is connected to the management bridge on the manager node. The Deployment tool is executed on the manager node; it uses the management network to dispatch the commands to launch applications on different VMs in the experimental network. This approach prevents any possible management traffic interfering in the experimental network that could distort experimental results.

III. Showcase

The primary goal of DiG is to create experimental network environments with resource guarantee to imitate real world SDN-based data center and ISP topologies with high level of realism. Such network environments can be used for instance to test and evaluate performance of SDN controllers or routing algorithms with different real time traffic or topologies.

In this demo, we will showcase how to automatically emulate an OpenFlow data center composed of hundreds of servers in Grid5000. The second phase of the demonstration will deploy and run Hadoop benchmark programs. Hadoop is used in many real world data centers and many benchmark suites exist (e.g., HiBench [9]). Interestingly, Hadoop MapReduce applications generate a substantial amount of traffic during the data shuffling phase and particularly the TestDFSIO and TeraSort are MapReduce benchmark applications. Hence, they can be a primary choice for data-center workload generation to demonstrate the effectiveness of DiG.

IV. Experimental requirement

1) Stable high speed Internet connection without blocking SSH traffic.
2) Power supply plug.
3) Large monitor for better visuals (to be provided, possibly by the organizers).

V. Conclusion

In this paper, we present the DiG tool to create easily SDN data center and ISP networks on a Grid infrastructure. DiG runs network embedding algorithms to emulate data center infrastructures in a Grid with performance guarantees. DiG automatically creates L2 overlay experimental networks and hosts based on the output of the embedding algorithms and can launch any off-the-shelf application on the experimental hosts to generate workload on the data center to evaluate. We demonstrate DiG by automatically running Hadoop benchmarks in an emulated data center of hundreds of nodes. DiG is available to the community at URL http://team.inria.fr/diana/software/.

In the future we plan to use DiG to study SDN controller performance as carried out in [10] but with real applications and traffic. Similarly, DISCO [11], ElastiCon [12], Hedera [13], DIFANE [14], DevoFlow [15], Kandoo [16], Onix [17] or Beehive [18] could be tested and evaluated on the DiG system with real network conditions and traffic.

Acknowledgments

This work was partially supported by the ANR Reflexion Project (ANR-14-CE28-0019). Experiments presented in this paper were carried out using the Grid'5000 testbed, supported by a scientific interest group hosted by Inria and including CNRS, RENATER and several Universities as well as other organizations (see https://www.grid5000.fr).

References

[1] B. Lantz, B. Heller, and N. McKeown, "A network in a laptop: Rapid prototyping for software-defined networks," in *Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks*, ser. Hotnets-IX. New York, NY, USA: ACM, 2010, pp. 19:1–19:6. [Online]. Available: http://doi.acm.org/10.1145/1868447.1868466

[2] P. Wette, M. Draxler, A. Schwabe, F. Wallaschek, M. Zahraee, and H. Karl, "Maxinet: Distributed emulation of software-defined networks," in *Networking Conference, 2014 IFIP*, June 2014, pp. 1–9.

[3] "ns-3 Project page," http://www.nsnam.org/, [Online; accessed 11-August-2015].

[4] "Grid'5000 Project page," http://www.grid5000.fr/, [Online; accessed 11-August-2015].

[5] A. Fischer, J. Botero, M. Till Beck, H. de Meer, and X. Hesselbach, "Virtual network embedding: A survey," *Communications Surveys Tutorials, IEEE*, vol. 15, no. 4, pp. 1888–1906, Fourth 2013.

[6] M. Beck, C. Linnhoff-Popien, A. Fischer, F. Kokot, and H. de Meer, "A simulation framework for virtual network embedding algorithms," in *Telecommunications Network Strategy and Planning Symposium (Networks), 2014 16th International*, Sept 2014, pp. 1–6.

[7] "The DOT Language," http://www.graphviz.org/doc/info/lang.html/, [Online; accessed 11-August-2015].

[8] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "Openflow: Enabling innovation in campus networks," *SIGCOMM Comput. Commun. Rev.*, vol. 38, no. 2, pp. 69–74, Mar. 2008. [Online]. Available: http://doi.acm.org/10.1145/1355734.1355746

[9] "HiBench," https://github.com/intel-hadoop/HiBench, [Online; accessed 11-August-2015].

[10] A. Tootoonchian, S. Gorbunov, Y. Ganjali, M. Casado, and R. Sherwood, "On controller performance in software-defined networks," in *Proceedings of the 2Nd USENIX Conference on Hot Topics in Management of Internet, Cloud, and Enterprise Networks and Services*, ser. Hot-ICE'12. Berkeley, CA, USA: USENIX Association, 2012, pp. 10–10. [Online]. Available: http://dl.acm.org/citation.cfm?id=2228283.2228297

[11] K. Phemius, M. Bouet, and J. Leguay, "Disco: Distributed sdn controllers in a multi-domain environment," in *Network Operations and Management Symposium (NOMS), 2014 IEEE*, May 2014, pp. 1–2.

[12] A. A. Dixit, F. Hao, S. Mukherjee, T. Lakshman, and R. Kompella, "Elasticon: An elastic distributed sdn controller," in *Proceedings of the Tenth ACM/IEEE Symposium on Architectures for Networking and Communications Systems*, ser. ANCS '14. New York, NY, USA: ACM, 2014, pp. 17–28. [Online]. Available: http://doi.acm.org/10.1145/2658260.2658261

[13] 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. 19–19. [Online]. Available: http://dl.acm.org/citation.cfm?id=1855711.1855730

[14] M. Yu, J. Rexford, M. J. Freedman, and J. Wang, "Scalable flow-based networking with difane," *SIGCOMM Comput. Commun. Rev.*, vol. 41, no. 4, pp. –, Aug. 2010. [Online]. Available: http://dl.acm.org/citation.cfm?id=2043164.1851224

[15] A. R. Curtis, J. C. Mogul, J. Tourrilhes, P. Yalagandula, P. Sharma, and S. Banerjee, "Devoflow: Scaling flow management for high-performance networks," *SIGCOMM Comput. Commun. Rev.*, vol. 41, no. 4, pp. 254–265, Aug. 2011. [Online]. Available: http://doi.acm.org/10.1145/2043164.2018466

[16] S. Hassas Yeganeh and Y. Ganjali, "Kandoo: A framework for efficient and scalable offloading of control applications," in *Proceedings of the First Workshop on Hot Topics in Software Defined Networks*, ser. HotSDN '12. New York, NY, USA: ACM, 2012, pp. 19–24. [Online]. Available: http://doi.acm.org/10.1145/2342441.2342446

[17] T. Koponen, M. Casado, N. Gude, J. Stribling, L. Poutievski, M. Zhu, R. Ramanathan, Y. Iwata, H. Inoue, T. Hama, and S. Shenker, "Onix: A distributed control platform for large-scale production networks," in *Proceedings of the 9th USENIX Conference on Operating Systems Design and Implementation*, ser. OSDI'10. Berkeley, CA, USA: USENIX Association, 2010, pp. 1–6. [Online]. Available: http://dl.acm.org/citation.cfm?id=1924943.1924968

[18] S. H. Yeganeh and Y. Ganjali, "Beehive: Towards a simple abstraction for scalable software-defined networking," in *Proceedings of the 13th ACM Workshop on Hot Topics in Networks*, ser. HotNets-XIII. New York, NY, USA: ACM, 2014, pp. 13:1–13:7. [Online]. Available: http://doi.acm.org/10.1145/2670518.2673864