2016 IEEE International Conference on Cloud Engineering

SENDIM for Incremental Development of Cloud Networks: Simulation, Emulation & Deployment Integration Middleware

Pradeeban Kathiravelu
INESC-ID Lisboa / Instituto Superior Técnico
Universidade de Lisboa, Portugal
pradeeban.kathiravelu@tecnico.ulisboa.pt

Luís Veiga
INESC-ID Lisboa / Instituto Superior Técnico
Universidade de Lisboa, Portugal
luis.veiga@inesc-id.pt

Abstract

Cloud networks are tested over simulation, emulation, and physical environments at different stages of development. Configuration management tools manage deployments and migrations across different cloud platforms, mitigating tedious system administration efforts. However, currently a cloud networking simulation cannot be migrated into an emulation, or vice versa, without rewriting and manually re-deploying the simulated application. This paper presents *SENDIM*[1], a **S**imulation, **E**mulation, a**N**d **D**eployment **I**ntegration **M**iddleware for cloud networks. As an orchestration platform for incrementally building Software-Defined Cloud Networks (SDCN), *SENDIM* manages the development and deployment of algorithms and architectures the entire length from visualization, simulation, emulation, to physical deployments.

I. Introduction

In a software development and testing process for cloud and network systems, automation and management tools focus on either deployment aspects, or early development stages such as simulation. While configuration management tools automate the deployment, reducing the manual efforts of DevOps [1], they focus only on product deployment aspects and do not manage the early design and development configuration efforts. Though approaches leveraging design patterns such as service-oriented architecture (SOA) [2] are used in managing multiple different components and products, they do not target the existing SDCN applications.

A framework that covers the entire development process to increase the productivity by minimizing the installation hell imposed due to the migration across multiple platforms should be researched and implemented for network systems. The configuration management approach can be extended to generalize the application logic such that a simulation can easily be migrated to an emulation or a physical deployment. An integrated framework should enable incremental development and deployment of applications, as well as migration between different physical deployments.

This paper presents *SENDIM*, an integrated cloud management architecture that coordinates heterogeneous SDCN systems and components, and manages cloud deployments, by extending the principles of SDN. *SENDIM* offers an orchestration process for building, analyzing, and migrating efficient cloud network architectures with different topologies, design dimensions, and deployment environments. *SENDIM* abstracts configurations and logic away from the application to provide seamless migration across multiple environments and different visualizations, such as simulations and emulations.

SENDIM provides ***Software-Defined Deployment***, offering programmable deployments. It minimizes manual deployments by automating deployments and migrations across different platforms and infrastructures. We foresee a shorter time to deliver as *SENDIM* minimizes the development overhead caused by incompatible APIs, programming languages, and granularity among the simulators and emulators, and administrative overheads caused by the deployment migration.

II. Background and Related Work

Integrating simulation, emulation, and deployments among heterogeneous systems has been proposed and researched in order to mitigate the administration overhead, in systems such as sensor networks [3]. Docker is an open source platform that provides an orchestration framework to build and execute distributed systems across different infrastructures, locally and over clouds [4]. Leveraging OSGi modular architecture for their extensibility, OpenDaylight [5] and ONOS [6] controllers cover many aspects of Software-Defined Networking (SDN), providing network orchestration and service management.

Configuration management tools such as Chef, CFEngine, and Puppet manage the configurations of cloud-scale deployments [1]. They automate the configuration and reconfiguration of the servers and virtual machines (VMs) in the cloud, eliminating the requirement to manually change or configure the servers, or write automation scripts. As they target the system administrators and cover mostly the deployment stage, configuring and deploying cloud applications for the early stage developments and quality assurance are still lacking. A complete middleware platform should be developed offering an integrated management architecture and deployment process for the SDCN applications, from visualization, simulation, emulation, to physical deployments, leveraging the global view of the network available to the SDCN controllers.

III. Solution Architecture

SENDIM is an integration middleware that optimizes the evaluation of cloud network algorithms and architectures. It minimizes the development efforts in testing the early research ideas to deployment, by offering a unified interface for simulation, emulation, and physical deployment of the cloud applications. It extends the motivation behind the SDN paradigm to the development aspects of applications that are designed to be executed on large scale SDCN systems consisting of many data centers. Our previous work

[1]Sendim is a northeastern Portuguese town close to the Spanish border, where the rare Mirandese language is spoken.

xSDN [7], an expressive simulator for network flows, has been extended to provide the cloud network simulation capabilities as well as a domain specific language with an easy-to-learn syntax to *SENDIM*.

A. Software-Defined Cloud Orchestration

SENDIM offers a software-defined cloud orchestration, an integrated software engineering process for SDCN, that builds up on the interoperability offered by the software-defined systems, offering programmable deployments improving the testing and quality assurance of the developments since early stage, managing the entire development process from design to production. As shown by Figure 1, software engineering process progresses in two dimensions: development dimension and deployment dimension. Development dimension consists of incrementally developing the application in different realizations, from design, simulation, emulation, to virtual and physical environments. The application is deployed in multiple environments such as development, testing, staging, to production environments.

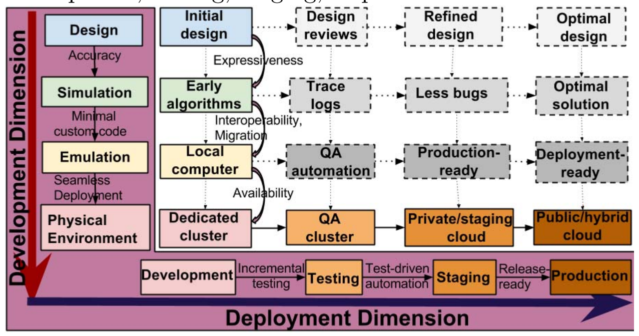

Fig. 1. Software-Defined Deployments

In the development dimension, transition from higher to lower realizations imposes different challenges at each level. Simulators should provide accuracy in expressing the chosen design in simulation. Porting from simulation to emulation should have minimal custom code developed. A seamless deployment is mandatory when migrating from emulation to physical deployment. While simulations are suitable for a few application scenarios, emulations or physical test deployments are mandatory for other. *SENDIM* leverages this for practical use case scenarios such as network load balancing and usage throttling algorithms, in building a system that scales itself across multiple realizations, with minimal human intervention.

Deployment dimension is test-driven in the enterprises. Development environment to testing environment should offer incremental testing, enabling smooth transition to staging environment. Staging is an environment identical to production, where release-ready applications are tested and migrated to production after final verifications. Multiple physical deployment environments exist based on the enterprise requirements. Development environment usually is a dedicated developer cluster. Testing environment consists of a QA cluster. Applications are often tested in a private or staging cloud, before moving to the production cloud.

In the deployment dimension, design is improved with revisions, to a refined and widely optimal design. Simulations are used to trace bugs with more logs and debugs in testing environment; with less bugs, it leads to an optimal solution. Emulations are migrated to testing environment with a QA automation. Emulations are production-ready in the staging environment phase, to deploy in production.

SENDIM offers a software-defined deployment by separating the deployment logic from the details of the given environments, by building the applications as bundles and creating the revisions as changesets and updated bundles, that can be deployed in the application hosted in the respective environment. By decoupling the deployed application from the details of the individual development and deployment environments and realizations, *SENDIM* automates and manages the entire matrix composed of these two dimensions. Hence, product deployments and updates are orchestrated from a central location more efficiently with less manual efforts.

Each application has an Initiator workflow, that deploys the initial version of the application into the controller and optionally simulate or emulate the system in the specified environment. It is also possible to deploy and execute the application into multiple environments at once using *SENDIM* deployers. The entire application is deployed into the controller for the initial or first deployment of the algorithm. Subsequent iterative deployments only need to deploy the changesets. Pseudocode of the application initializer workflow is presented in Algorithm 1.

Algorithm 1 *SENDIM* Application Initialization

```
1:  procedure DEPLOY($deplDescriptor$, $sysDescriptor$, $controller$, $appBundles$)
2:      $deployment \leftarrow parse(deplDescriptor)$
3:      $deploy(controller, appBundles)$
4:      if $(deployment.env.isPhysical())$ then
5:          while $(TRUE)$ do
6:              $execute(deployment.env)$
7:          end while
8:      else if $(deployment.env.isSimulation())$ then
9:          $simulationSandbox \leftarrow construct($
                        $deployment.env, sysDescriptor)$
10:         while $(TRUE)$ do
11:             $execute(simulationSandbox)$
12:         end while
13:     else if $(deployment.env.isEmulation())$ then
14:         $emulator \leftarrow deployment.env$
15:         $descriptor \leftarrow convert(emulator, sysDescriptor)$
16:         $construct(emulator, descriptor)$
17:         while $(TRUE)$ do
18:             $execute(emulator)$
19:         end while
20:     end if
21:     $clearDistributedObjects()$
22: end procedure
```

While physical deployments are identified and used to execute the workflows, emulations and simulations first need to construct the virtual networked systems before executing the network flows on them. In case of emulations, the system descriptors usually need to be converted to have the script for the respective emulator, whereas the simulations execute native inside *SENDIM*, based on the system descriptor.

B. SENDIM *Middleware Platform*

OSGi modular architecture is leveraged, with in-memory data grids for a distributed execution [8]. Figure 2 provides a higher level deployment view, depicting how the user applications are deployed into either the simulation or emulation environment or deployed in physical networks, along with the cross-platform deployment migrations.

The *SDCN parser* parses the *system descriptors* to compose the software-defined systems for the simulation, where the *SDD parser* parses the *deployment descriptor* for software-defined deployments. The user applications are deployed and executed on the environment defined in a *deployment descriptor* file, abiding to the deployment policies. While the SDCN is expressed in the system descriptor files following the *SENDIM* DSL, the descriptors are automatically converted into the respective script for the emulators by the relevant Converter implementations. The relevant emulator is triggered with the network constructed and flows initialized, when the emulation environment is given as the execution environment.

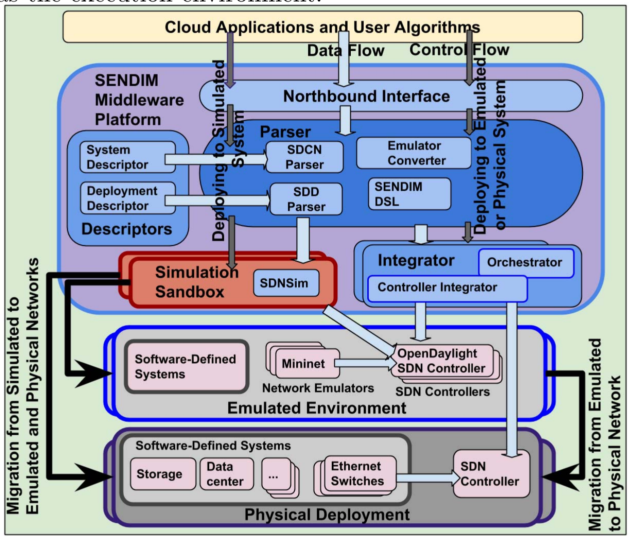

Fig. 2. *SENDIM* Architecture and Deployments

The simulation sandbox provides simulation for software-defined networks and systems. While emulators and converters are connected through Converters and Integrators, simulation is handled in the *SDNSim* simulation sandbox of *SENDIM*. Controller integrator is an API for an integration with SDN controllers, together with which *SDNSim* provides a software-defined deployment. Orchestrator provides extension to the SDN controller to connect with software-defined systems.

SENDIM implementation includes the development of the middleware platform as well as the applications and scripts that are developed on top of *SENDIM* as use cases. *SENDIM* middleware includes the core *SENDIM* including the orchestrator and *SENDIM* core that provides a northbound API for the application developers. Moreover, it also includes extensions deployed into the controller and emulators for connecting them to *SENDIM* middleware core by extending their east/westbound APIs. The extension points are for providing the emulation parameters for the emulator or for invoking the REST APIs to inject the *SENDIM* control flow. Application changesets are deployed exploiting the northbound API of the controller, where the integration of the controller to the emulators, simulators, and physical deployments leverages the southbound API.

Emulators are integrated to *SENDIM* through the converters and emulator extensions. Converter converts the system descriptors into the scripts for the respective emulator. Emulator extensions enable remote invocations of the respective emulators to remotely execute, control, pause, and exit the executing instances. Multiple emulator instances execute from different deployment environments, leveraging the single distributed controller environment in a tenant-aware manner, hence enabling an easy migration across multiple platforms, from simulations to emulations and vice versa.

Mininet has been leveraged as the default emulator. As emulations consume more resources than simulations, it is essential that the simulated platform should be able to be emulated in the *SENDIM* system with the given resources. With minimal overheads during the transition and execution in addition to the base emulation, the overhead imposed by the middleware is designed to be negligible once the emulator is integrated. If the resources are available, a network system can be preconfigured to resemble the emulated system, such that a seamless migration is possible between the chosen emulation to physical deployment environment, without any modification to the controller.

IV. Evaluation

SENDIM prototype was evaluated for integrated simulation and emulation capabilities, on a computer with Intel® Core™ i7-4700MQ CPU @ 2.40GHz × 8 processor, 8 GB memory, and Ubuntu 14.04 LTS 64 bit operating system. Multiple (up to 6) identical computers in a cluster were further used in evaluating the migration of the applications across different physical deployment environments.

Complex networks with small-world datacenter topology [9] were modeled using the DSL of *SENDIM*, and a routing was simulated across multiple hosts in shortest path. Varying number of switches and hosts with varying degrees up to 100 were used for multiple executions. In addition to the automated migrations across the deployment dimension offered by the configuration management systems, it was observed that *SENDIM* was also efficient in migrations across the development dimension.

A. *Emulation to Simulation Migrations*

Figure 3 shows the time taken for Mininet to emulate the network system, SDNSim to simulate the system, and *SENDIM* configured to resort to SDNSim simulations when the emulations take more than 1 minute. While SDNSim was able to construct the system with nodes and links within a few seconds even for large systems with up to 100,000 nodes, Mininet consumed up to 300 seconds even for a network of 1800 nodes and links. Hence, SDNSim showed a 100-fold performance compared to Mininet in visualizing the network.

When increasing the network size further up to 1900, Mininet failed to create the links, though it successfully emulated the nodes. For networks beyond the size of 8000, Mininet fails with an OSError when attempting to construct the nodes. When emulating a network of 1800 nodes with Mininet, Network Manager consumed as high as 99.6% of CPU during the links construction phase, and up to 100% of CPU consumption by the Open vSwitch daemon (ovs-vswitchd) was noticed during the final stages of emulation. However, simulation of the same network with *SDNSim* consumed only up to 6% of CPU.

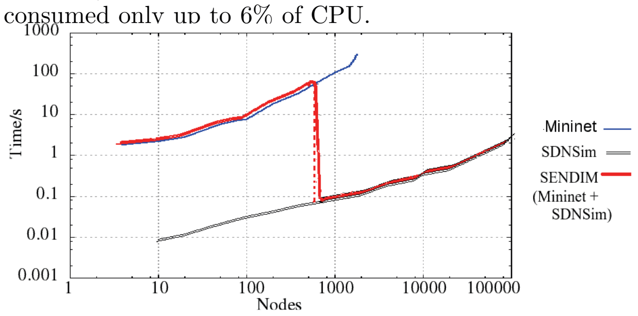

Fig. 3. Network Construction with Mininet and *SENDIM* SDNSim

Results as accurate as those produced by Mininet emulation were observed with SDNSim simulations for simple network tasks such as routing algorithms, cloud resource allocation, and network load balancing. This allows *SENDIM* to resort to simulations to visualize large scale data centers in the early stages of development.

SENDIM offers best of both worlds, as it offers the accuracy of emulations when resources are available, and simulate for larger networks with limited resources, transitioning seamlessly across both realizations. Moreover, this also indicates the potential of a seamless migration from simulations to emulations by either increasing the computing resources, or by reducing the simulated system size.

B. Simulation to Emulation Migrations

Mininet converter for *SENDIM* was evaluated for its efficiency in migrating the SDNSim simulations into Mininet emulations, with the efforts reduced in developing separate custom code for simulations and emulations. Based on the deployment descriptor, the system was either simulated in SDNSim simulation sandbox, or automatically converted to Mininet scripts by the *SENDIM* converter and emulated using Mininet without manual coding or deployment. Here the system descriptions in XML were converted to relevant Python scripts for Mininet.

Simulation to emulation migration was experimented with varying number of nodes including switches and hosts and varying average degree. System descriptors expressed the complex systems which are parsed into the network as well as the flows and custom properties. Figure 4 shows the time taken to migrate an application from simulation to emulation, by converting the system descriptor scripts, initiating Mininet with the network consists of nodes, which are either switches or hosts, and links across those nodes as specified in the descriptor, and start the Mininet execution of simple routing across the nodes. Time taken for the migration increases with the number of nodes, as the converter has to read, parse, and convert the descriptor files into Mininet scripts, and start the execution. Links are expressed as neighbor nodes to each of the nodes, and hence fine-grain manipulation of flows is possible with accurate representation of real SDCN systems.

There was also an increase in time with the increase in the number of average degrees per node, which is visible more in larger networks with more nodes. As depicted by Figure 4, *SENDIM* was able to handle 10,000 nodes within a few seconds. Larger and complex networks of 100,000 nodes were successfully migrated, though there was a clearly above linear increase in the time taken. This was because of the memory and processing required for the parsing of large descriptor files which are as large as 26 MB, and writing Mininet scripts that are of up to 18 MB for 100,000 nodes with up to 100 links for each node.

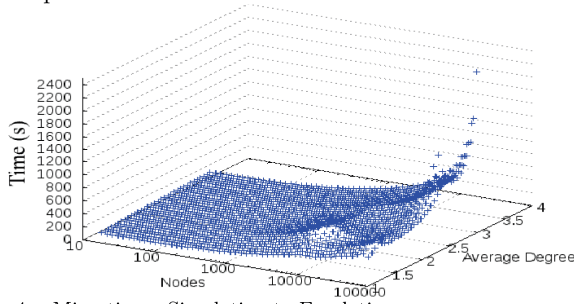

Fig. 4. Migrating a Simulation to Emulation

V. Conclusion and Future Work

SENDIM integrates and automates multiple phases of development process, by enabling seamless migration through loose coupling of algorithms from their deployment environments and development realizations such as simulations and emulations. The experimental evaluation elaborated the potential benefits of the middleware platform in increasing productivity and efficiency of the SDCN application development cycle. A complete cloud platform can be orchestrated using the proposed architecture, providing an end-to-end guarantee on testing the user applications.

Acknowledgements: This work was supported by national funds through Fundação para a Ciência e a Tecnologia with reference UID/CEC/50021/2013.

References

[1] D. Spinellis, "Don't install software by hand," *Software, IEEE*, vol. 29, no. 4, pp. 86–87, 2012.

[2] R. Perrey and M. Lycett, "Service-oriented architecture," in *Applications and the Internet Workshops, 2003. Proceedings. 2003 Symposium on.* IEEE, 2003, pp. 116–119.

[3] L. Girod, T. Stathopoulos, N. Ramanathan, J. Elson, D. Estrin, E. Osterweil, and T. Schoellhammer, "A system for simulation, emulation, and deployment of heterogeneous sensor networks," in *Proceedings of the 2nd international conference on Embedded networked sensor systems.* ACM, 2004, pp. 201–213.

[4] D. Merkel, "Docker: lightweight linux containers for consistent development and deployment," *Linux Journal*, vol. 2014, no. 239, p. 2, 2014.

[5] J. Medved, R. Varga, A. Tkacik, and K. Gray, "Opendaylight: Towards a model-driven sdn controller architecture," in *2014 IEEE 15th International Symposium on.* IEEE, 2014, pp. 1–6.

[6] P. Berde, M. Gerola, J. Hart, Y. Higuchi, M. Kobayashi, T. Koide, B. Lantz, B. O'Connor, P. Radoslavov, W. Snow *et al.*, "Onos: towards an open, distributed sdn os," in *Proceedings of the third workshop on Hot topics in software defined networking.* ACM, 2014, pp. 1–6.

[7] P. Kathiravelu and L. Veiga, "An expressive simulator for dynamic network flows," in *Cloud Engineering (IC2E), 2015 IEEE International Conference on.* IEEE, 2015, pp. 311–316.

[8] ——, "An adaptive distributed simulator for cloud and mapreduce algorithms and architectures," in *Utility and Cloud Computing (UCC), 2014 IEEE/ACM 7th International Conference on.* IEEE, 2014, pp. 79–88.

[9] J.-Y. Shin, B. Wong, and E. G. Sirer, "Small-world datacenters," in *Proceedings of the 2nd ACM Symposium on Cloud Computing.* ACM, 2011, p. 2.