Kollaps/Thunderstorm: Reproducible Evaluation of Distributed Systems
Tutorial Paper

Miguel Matos(✉)

U. Lisboa & INESC-ID, Lisbon, Portugal
miguel.marques.matos@tecnico.ulisboa.pt
https://www.gsd.inesc-id.pt/∼mm/

Abstract. Reproducing experimental results is nowadays seen as one of the greatest impairments for the progress of science in general and distributed systems in particular. This stems from the increasing complexity of the systems under study and the inherent complexity of capturing and controlling all variables that can potentially affect experimental results. We argue that this can only be addressed with a systematic approach to all the stages and aspects of the evaluation process, such as the environment in which the experiment is run, the configuration and software versions used, and the network characteristics among others. In this tutorial paper, we focus on the networking aspect, and discuss our ongoing research efforts and tools to contribute to a more systematic and reproducible evaluation of large scale distributed systems.

1 Introduction

Evaluating distributed systems is hard. The underlying network topology, in particular, can have a drastic impact on key performance metrics, such as throughput and latency but also on correctness depending, for instance, on the asynchrony assumptions made by the system designer. With the increasingly popular deployment of geographically distributed applications operating at a global scale [5], assessing the impact of geo-distribution, and hence network topology, is fundamental to build and tune systems that perform correctly and meet the desired Service Level Objectives. Unfortunately, there is still an important gap between the easiness of deploying a distributed system and its evaluation.

On the one hand, the deployment of geographically distributed systems was made simpler thanks to the increasing popularity of container technology (*e.g.*, Docker [13], Linux LXC [8]). Big IT players introduced such technologies in their commercial offering (*e.g.*, Amazon Elastic Container Service [1], Microsoft Azure Kubernetes Service [2] or Google Cloud Kubernetes Engine [7]), and they are an attractive mechanism to deploy large-scale applications.

On the other hand network properties such jitter, packet loss, failures of middle-boxes (*i.e.*, switches, routers) are by definition difficult, if not impossible, to predict from the standpoint of a system developer, who has no control over the underlying network infrastructure. Moreover, such conditions are the

Published by Springer Nature Switzerland AG 2020
A. Remke and V. Schiavoni (Eds.): DAIS 2020, LNCS 12135, pp. 121–128, 2020.
https://doi.org/10.1007/978-3-030-50323-9_8

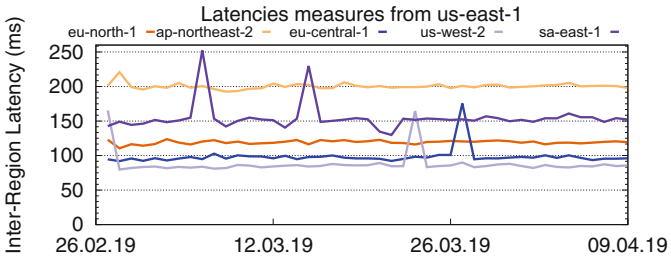

Fig. 1. Latency variability between five different AWS regions across the world over 45 days. Latencies vary on average between 90 ms and 250 ms, while spikes occur across all regions.

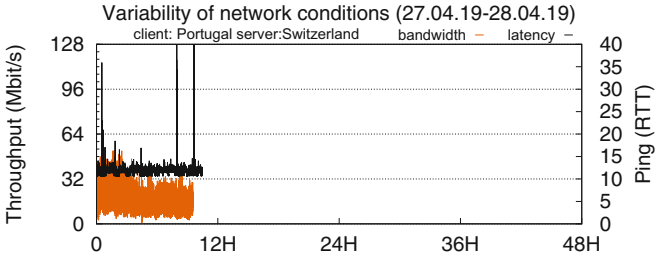

Fig. 2. Dynamic network conditions between two university campuses in Europe, in Portugal and Switzerland respectively.

norm rather than the exception, in particular when considering large-scale wide-area networks that might cross several distinct administrative domains. As a motivating example, consider Fig. 1 which shows the average latency between six AWS [1] regions over 45 days measured by https://www.cloudping.info. We observe that even in the infrastructure of a major cloud provider, there are significant and unpredictable variations in latency.

Variability is not limited to latency. We demonstrates this with a measurement experiment for two different cases. The first set of measures are taken between two stable endpoints inside university networks, respectively in Portugal and Switzerland, shown in Fig. 2. The second case measures the network conditions between a remote AWS instance in the `ap-northeast-1a` zone (in Tokyo) and a server node in Switzerland (Fig. 3). As we can observe, there are important variations both for bandwidth and latency. Such variability can have a dramatic effect not only on a system's performance but also on reliability as shown by recent post-mortem analysis of major cloud providers [4]. The challenge, therefore, is how to equip engineers and researchers with the tools that allow to systematically understand and evaluate how this variability affects system's performance and behavior.

In our ongoing work, we are conducting research and developing tools to precisely enable these experiments. In this tutorial paper, we briefly introduce

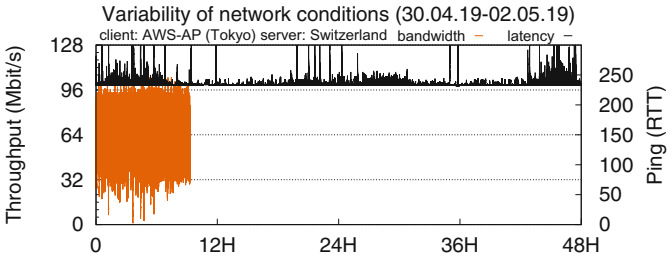

Fig. 3. Dynamic network conditions between a university node in Switzerland and a node in AWS `ap-northeast-1a` (Tokyo).

KOLLAPS [10], a decentralized and scalable topology emulator (Sect. 2), and THUNDERSTORM [12], a compact domain specific language to describe dynamic experiments on top of KOLLAPS (Sect. 3). Then we present some experiments enabled by KOLLAPS and THUNDERSTORM in Sect. 4 and conclude in Sect. 5.

2 Kollaps

In this section, we briefly describe the architecture and workflow KOLLAPS, depicted in Fig. 4.

First, the user must describe the topology in the THUNDERSTORM Description Language (TDL), discussed in the next section. This includes the network topology, network dynamics, if any, and the Docker images of the distributed application being evaluated (Fig. 4, *define* step). These images can come from either private repositories or public ones such as Docker Hub [3].

With the experiment defined, the user invokes the *deployment generator*, a tool shipped with KOLLAPS that transforms the TDL into a Kubernetes Manifest file, or a Docker compose file. This file is ready to be deployed using Kubernetes or Docker Swarm, but the user can manually fine-tune it if needed.

The user can then use this file to deploy the experiment in any Kubernetes cluster (Fig. 4, *deploy* step). This deploys not only the target application under evaluation but also an Emulation Manager component per physical machine (Fig. 4, *execute* step). The Emulation Manager is a key component of KOLLAPS responsible for maintaining and enforcing the emulation model in a distributed fashion. More details on the design and implementation of KOLLAPS can be found in [10].

3 Thunderstorm

In this section, we describe the THUNDERSTORM Description Language (TDL). The TDL abstracts the low level details of KOLLAPS and allows to succinctly express dynamic experiments. An example of the TDL, illustrating the main features of the language can be found in Listing 1.1.

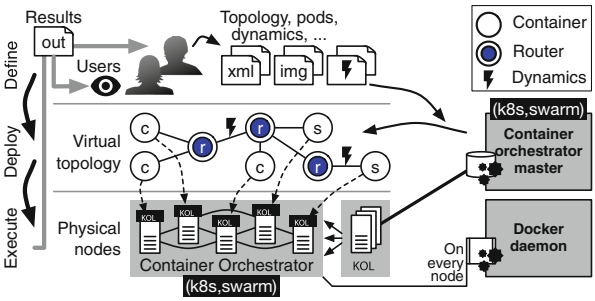

Fig. 4. KOLLAPS architecture and workflow. We assume the existence of an existing cluster and a working Docker Swarm or Kubernetes environment.

The language describes the services (lines 3–6), the static topology (lines 10–14) and the dynamics in the topology (lines 16–26). The `service`, `bridge` and `link` elements can get an arbitrary number of `tag`s. In the example, the `api` and `db` services belong to the backend, while the `server` belongs to the frontend. These three are grouped into the same application together, expressed by the `app` tag. The `client` is not part of the application. We use tags to group services and links together based on real-world criteria. For example, one of the most common causes for network "failures" is the distributed roll-out of software upgrades [14], *e.g.* for routers. Tags help to capture groups of devices sharing network status-relevant characteristics, *e.g.*, driver versions that could be updated at the same time. Tags could also be used to map services to data centers (*i.e.*, what if one's connection suddenly changes?) or to logical parts of a distributed system (frontend, backend). Bridges (line 8) must have unique names. Links must specify the source, destination and the properties (*e.g.*, latency, bandwidth, jitter, etc.). The `symmetric` keyword allow to easily create bidirectional links with the same specified properties.

The dynamic events can be expressed in a concise yet rich manner. In our example, we first start all application services (3 replicas for the `api` and the `db`, and 5 replicas for the `server`, lines 16–19), and after 30 s the clients (line 19). After 30 min, we inject several faults into the topology. The `churn` keyword crashes either an absolute number of services, or a certain share of all instances of that service. The `replace` keyword then specifies the probability of such a service to immediately re-join the cluster. At line 20, we specify that the `server` replicas will be subject to churn over a 3 h period. In particular, 40% of the servers will crash uniformly at random over this period, and of those, 50% will be replaced immediately. Although the language allows to define events with a degree of randomness, such as the churn event above, it is possible to systematically reproduce the same order of events by setting a fixed random seed. We can also specify the dynamic behavior for a specific container. In the example, one server instance leaves the system at four hours and twenty, and joins 5 min later (lines 21–22).

```
1 bootstrapper thunderstorm:2.0
2
3 service server img=nginx:latest tags=frontend;app
4 service api img=api:latest tags=backend;app
5 service client img=client:1.0 command=['80']
6 service db img=postgres:latest tags=backend;app
7
8 bridges s1 s2
9
10 link server--s1 latency=9.1 up=1Gb down=800Mb
11 link api--s1 latency=5.1 up=1Gb symmetric
12 link s1--s2 latency=0.11 up=1Gb symmetric
13 link client--s1 latency=23.4 up=50Mb down=1Gb
14 link db--s2 latency=8.0 up=1Gb symmetric
15
16 at 0s api join 3
17 at 0s db join 3
18 at 0s server join 5
19 at 30s client join
20 from 30m to 3h30m server churn 40% replace 50%
21 at 4h20m server--s1 leave
22 at 4h25m server--s1 join
23 from 10h2m to 10h6m api--s1 flap 0.93s
24 from 12h to 24h tags=be leave 60%
25 from 15h to 15h20s server disconnect 1
26 at 18h20m api--s1 set latency=10.2 jitter=1.2
```

Listing 1.1. Example of experiment descriptor using the THUNDERSTORM description language. Link rates are given in 'per second'.

The language supports *link flapping*, where a single link connects and disconnects in quick succession [14]. In the experiment, the link between service `api` and bridge `s1` flaps every 0.93 s during a period of 4 min (line 23). The `leave` action, used to define which entities should leave the emulation, takes as a parameter an absolute number or a share of all selected instances. At line 24, 60% of all nodes with the *backend* tag, chosen uniformly at random, will leave the experiment. Internally, when the language is translated into the lower level format used by the KOLLAPS engine, we keep track of all nodes that have joined, left, connected, or disconnected. Thus, if a percentage rather than an absolute number is provided, that is always relative to the amount of legal targets in the cluster *at that moment*.

The output of the parser is a XML file, ready to be consumed by the deployment generator and starting the experiment workflow discussed in the previous section. Further details about the design and implementation of THUNDERSTORM can be found in [12].

4 Experiments

In this section, we illustrate the capabilities of Kollaps and Thunderstorm. The goals are two-fold: show that the emulation is accurate, and also that it allows to easily evaluate a system under network dynamics.

The evaluation cluster is composed of 4 Dell PowerEdge R330 servers where each machine has an Intel Xeon E3-1270 v6 CPU and 64 GB of RAM. All nodes run Ubuntu Linux 18.04.2 LTS, kernel v4.15.0-47-generic. The tests conducted on Amazon EC2 use `r4.16xlarge` instances, the closest type in terms of hardware-specs to the machines in our cluster.

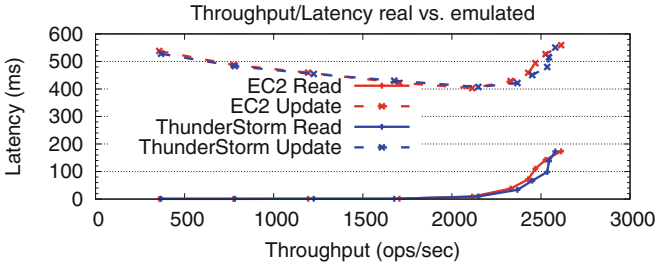

Fig. 5. Throughput/latency of a geo-replicated Cassandra deployment on Amazon EC2 and Kollaps

We start by comparing the results of benchmarking a geo-replicated Apache Cassandra [6,11] deployment on Amazon EC2 and on Kollaps. The deployment consists of 4 replicas in Frankfurt, 4 replicas in Sydney and 4 YCSB [9] clients in Frankfurt. Cassandra is set up to active replication with a replication factor of 2. In order to model the network topology in Kollaps, we collected the average latency and jitter between all the Amazon EC2 instances used, prior to executing the experiment. Figure 5 shows the throughput-latency curve obtained from the benchmark on both the real deployment on Amazon and on Kollaps. The curves for both reads and updates are a close match, showing only slight differences after the turning point where response latencies climb fast, as Cassandra replicas are under high stress. This experiment demonstrates how such issues can be identified, debugged and eliminated with Kollaps before expensive real-life deployments.

We now highlight the unique support for dynamic topologies through the use of the TDL. This allows to easily evaluate the behaviour of complex systems in a variety of scenarios. In this experiment, the intercontinental link from `EU` to `AP` used for the Cassandra experiment in Fig. 5 suddenly changes its latency to half (at 240 s), and later on (at 480 s) the original latency is restored. In Fig. 6 we report the update latency observed by YCSB. Note that read operations do not use the intercontinental link and hence are not affected (not shown). This shows that the network dynamics imposed by Kollaps have a direct impact in

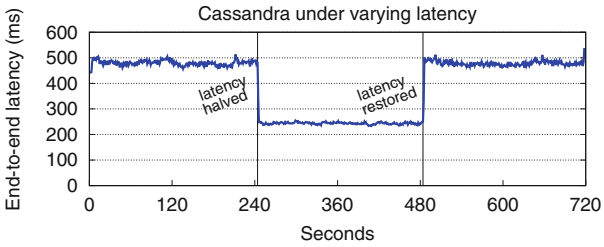

Fig. 6. Latency variations measured by YCSB during a transitory period: one of the replicas is moved to a far away region.

client-facing metrics. Engineers and researchers can therefore use KOLLAPS and THUNDERSTORM to conduct controlled and reproducible experiments to assess the behavior of real system under a wide range of network dynamics and devise the best strategies to adopt when such events happen in production.

5 Discussion

In this tutorial paper we illustrated the main features of KOLLAPS and THUNDERSTORM, in particular the accuracy of the emulation with respect to a real system deployed in a real environment, and also the dynamic experiments that THUNDERSTORM enables. Both tools are available as open source at https://github.com/miguelammatos/Kollaps.

We believe the ability to systematically reproduce experiments in a controlled environment, and the ability to subject a system to a wide range of dynamic scenarios provided by KOLLAPS and THUNDERSTORM are a step towards building more robust and dependable distributed systems.

Acknowledgments. The work presented in this tutorial paper is the joint effort of researchers at the University of Lisbon, Portugal, and researchers at the Univerité de Neuchâtel, Switzerland, namely: Paulo Gouveia, João Neves, Carlos Segarra, Luca Lietchi, Shady Issa, Valerio Schiavoni and Miguel Matos. This work was partially supported by national funds through FCT, Fundação para a Ciência e a Tecnologia, under project UIDB/50021/2020 and project Lisboa-01-0145- FEDER- 031456 (Angainor).

References

1. Amazon elastic container service. https://aws.amazon.com/ecs/
2. Azure kubernetes service. https://azure.microsoft.com/en-us/services/kubernetes-service/
3. Docker hub. https://hub.docker.com/
4. Google cloud post-mortem analysis. https://status.cloud.google.com/incident/cloud-networking/18012?m=1
5. Containers: real adoption and use cases in 2017. Technical report, Forrester, March 2017

6. Apache Cassandra (2019). https://cassandra.apache.org/. Accessed 12 Mar 2020
7. Google cloud kubernetes engine (2019). https://cloud.google.com/kubernetes-engine/. Accessed 12 Mar 2020
8. Linux LXC (2019). https://linuxcontainers.org/. Accessed 12 Mar 2020
9. Cooper, B.F., Silberstein, A., Tam, E., Ramakrishnan, R., Sears, R.: Benchmarking cloud serving systems with YCSB. In: Proceedings of the 1st ACM Symposium on Cloud Computing SoCC 2010, pp. 143–154. ACM, New York (2010). https://doi.org/10.1145/1807128.1807152
10. Gouveia, P., et al.: Kollaps: decentralized and dynamic topology emulation. In: Proceedings of the Fifteenth European Conference on Computer Systems EuroSys 2020. Association for Computing Machinery, New York (2020). https://doi.org/10.1145/3342195.3387540
11. Lakshman, A., Malik, P.: Cassandra: a decentralized structured storage system. ACM SIGOPS Oper. Syst. Rev. **44**(2), 35–40 (2010). https://doi.org/10.1145/1773912.1773922
12. Liechti, L., Gouveia, P., Neves, J., Kropf, P., Matos, M., Schiavoni, V.: THUNDERSTORM: a tool to evaluate dynamic network topologies on distributed systems. In: 2019 IEEE 38th International Symposium on Reliable Distributed Systems SRDS2019 (2019)
13. Merkel, D.: Docker: lightweight Linux containers for consistent development and deployment (2014). https://doi.org/10.1097/01.NND.0000320699.47006.a3. https://bit.ly/2IuhKBv
14. Potharaju, R., Jain, N.: When the network crumbles: an empirical study of cloud network failures and their impact on services. In: Proceedings of the 4th Annual Symposium on Cloud Computing SOCC 2013, pp. 15:1–15:17. ACM, New York (2013). https://doi.org/10.1145/2523616.2523638