EnOSlib: A Library for Experiment-Driven Research in Distributed Systems

Matthieu Simonin

5. October 2021

Who Am I ?

Matthieu Simonin working at Inria/SED Rennes as Research Engineer

- User of Grid'5000
- Officially part of Myriads and WIDE teams
- (also bring support in experimentation questions at Rennes)
- Maintainer of EnOSlib[1]
- (and that's today's talk)

[1] Cherrueau, et al. EnosLib: A Library for Experiment-Driven Research in Distributed Computing. IEEE Transactions on Parallel and Distributed Systems, ⟨10.1109/TPDS.2021.3111159⟩. ⟨hal-03324177⟩

A Distributed system (theory-wise)

A Distributed system (theory-wise):

- What: set of entities that have to collaborate
- Goal: the system make progress (e.g. the task terminate)
- Adversary: the uncertainty of the environment (e.g async, failures)

A Distributed system (experimentation-wise)

A Distributed system (experimentation-wise):

- What: set of entities that have to collaborate
  - you'll have to deploy it
- Goal: the system make progress (e.g. the task terminate)
  - you'll have to measure the progress, understand the non-progress
- Adversary: the uncertainty of the environment (e.g async, failures)
  - you'll have to tame the uncertainty
  - if you can control it: the adv. becomes an input of the experiment
  - if you can't: evaluate the variability introduced by the env

A Distributed system (experimenter-wise)

A Distributed system (experimenter-wise):

- What: set of entities that have to collaborate
  - you'll have to deploy it
    - where ? Grid'5000, FIT, other platforms, lab cluster . . .
    - how ? bare-metal (or virtualized), network conf (IPvX, vlans . . .)
- Goal: the system make progress (e.g. the task terminate)
  - you'll have to measure the progress, understand the non-progress
    - instrument the infra. (e.g generic probes at the system level)
    - instrument the app. (e.g. insert probes)
- Adversary: the uncertainty of the environment (async, failures)
  - you'll have to tame the uncertainty (control, evaluate the variability introduced by the env)
    - environment as code : inject failures, bad network conditions, set the workload
    - apply great statistical methods (see Thursday morning)

Today we'll talk about EnOSlib

EnOSlib[2] is a library for the experimenters.

It helps to

- deploy your distributed system
- instrument your deployment (e.g. monitoring)
- control/record some environmental conditions (e.g. network)

It doesn't help you

- for computational jobs: you want to run a function and you're interesting in the result
  - massively parallel jobs
  - MPI
- and other situations (to be discussed)

[2] https://discovery.gitlabpages.inria.fr/enoslib/

Rationale

EnOSlib[2] is a library for the experimenters.

- Tries to continuously factorizing common practices / tools
  - Avoid experimenters to re-invent the wheel
  - Embed *state-of-the-practice* tools for monitoring, sniffing ...
- Targets fast iteration between an hypothesis and its (in)validation
  - Should be easy to get resources and get initial insights on what is going on
- Avoid to be locked in the initial setup
  - I want to switch from a deployment in my local machine to a testbed
  - I want to switch from a bare-metal deployment to a virtualized setup on Grid'5000
  - I want to switch from a single NIC setup to several NIC setup
  - (sci-fi) I want to switch for an all-in-one Grid'5000 setup to a Grid'5000 + Fit setup

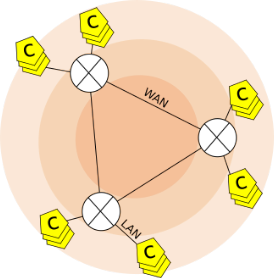
Digging into EnOSlib

Terminology

In the following let's distinguish between:

- The to-be-tested-system: The subject (e.g., software, protocol) of an experimental campaign whose behavior is studied
- The artifact: The software or set of scripts that implements the experimental protocol and allows for studying the to-be-tested system

My DBalgox system

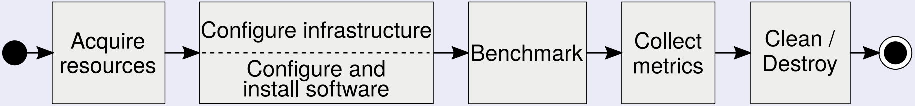


Let's assume you've build a new consensus algorithm (algoX) in a distributed database DB.

- To-be-tested system: DBalgoX (DB with algoX implemented)
- Some of the Artifacts requirements
  - Get resources from some infrastructure
  - Deploy DBalgoX + DB + other databases (for baseline comparison)
  - Run workload
  - Gather metrics

Concepts 1/2

Artifact as a simple workflow



There's a lot to mutualize:

- The science the experimenter is doing is unique
- But the experimental artifact is probably not so unique

=> EnOSlib defines 6 concepts to help the experimenter

Concepts 2/2

- Providers: A mean to get concrete resources from a testbed
- Resources: Model infrastructure resources, Host and Network.
- Roles: Add application semantics to the resources
- Modules: Deal with remote actions in a safe way
- Services: Provides *state-of-the-practice* facilities

Providers

Providers

Rationale:

- Replicate your experiment on different testbeds
  - corollary: develop locally then move on a production testbed
- Do the heavy-lifting of getting the concrete resources (hosts and networks)
  - Sits on top of the testbeds API/SDKs (if any . . .)
  - Brings the resources in good shape (e.g. VLANs configured . . .)

Providers

In EnOSlib you describe the resources in an abstract way:

Local setup

```
  conf = (
    Configuration()
    .add_machine(
       roles=["database"],
       flavour="medium",
       number=3
    )
    .add_machine(
       roles=["client"],
       flavour="tiny",
       number=1
    )
    .add_network(
       roles=["db-net"],
    )
    .finalize()
)
provider = Vagrant(conf)
```

Grid’5000

```
conf = (
    Configuration
    .add_network_conf(prod_network)
    .add_machine(
        roles=["database"],
        cluster="paravance",
        nodes=3,
        primary_network=network
    )
    .add_machine(
        roles=["client"],
        cluster="paravance",
        nodes=10,
        primary_network=network,
    )
    .finalize()
)
provider = G5k(conf)
```

Providers

It's the lib responsability to provide reliable and scalable providers

Supported providers

- Local: VirtualBox, KVM
- Grid5000
  - Bare-metal (automatic multi NIC configurations, multisite support, . . .)
  - Virtual Machines
  - LXC (distem)
- Chameleon Cloud
- FIT/IOTLab
- Custom (target any machine)

Providers: Focus on Grid'5000/FIT

Grid'5000 provider:

- `deploy/non-deploy` jobs
- `vlan` reservation + initial setup of your nodes
- `subnet` reservation (and use)
- `multisite` support (without oargridsub)
- `reservation` support
- Some dedicated experimental facilities
  - VMs / Distem
  - Docker

The lib has been extended to support FIT in an (almost) unified way

- nodes reservation
- common abstraction of IPv6 on both platforms
- more to come ...

Resources & Roles

Resources & Roles

```
resources = provider.init()
```

- idempotent: facilitate interactive programming
- wraps the hosts and network inventories

  ```
  hosts, networks = resources
  database_nodes = hosts["database"]
  database_network = networks["db-net"]
  ```

  - dict-like interface for accessing the host/network by role name
- resources are testbed agnostic
  - Corollary: hosts from different providers can be mixed (Bare Metal, VMs, containers . . .)
  - Corollary: networks from different providers can be mixed (vlan/non vlan, IPv4/IPv6)

Modules

Modules: remote actions on the hosts

```
# clone a repo
run(hosts["database"],
    "git clone mydbcode_url")

# add user
run(hosts["database"],
    "useradd -m foo")

# create a directory
run(hosts["database"],
    "mkdir /var/lib/mydb")
```

- This isn't idempotent code:
- This can be fixed manually but limits readability/maintainability

```
on(hosts["database"]) as p:

    p.git(repo="mydbcode_url",
          update="yes")

    p.user(name="foo",
           state="present")

    p.file(name="/var/lib/mydb",
           state="directory")
```

- idempotent / implicit parallelism
- This relies on Ansible Modules (6000+)

Services

Services

Rationale

- Bootstraps a software stack commonly used when experimenting
- Based on modules
- Hides the low-level details of its deployment

```
m = Monitoring(collector=hosts["util"],
               agent=hosts["database"],
               ui=hosts["util"],
               network=networks["mon-net"])
m.deploy()
# later
m.backup()
# later
m.destroy()
```

Services

It's the lib responsability to provide tested and relevant services

Packaged service

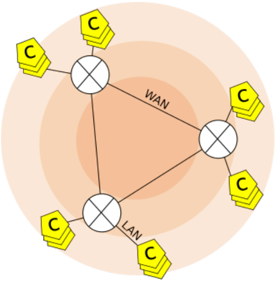
- Docker: clients configuration (registry configuration (proxy cache)) (swarm deployment)
- Netem: Network emulation (Netem) (HTB based with filtering)
- Monitoring: monitor your experiments (Dstat) (Influx based) (Prometheus based)
- Skydive: distributed Wireshark on steroïds
- TCPDump: dump network packets going through some remote interfaces
- Dask: Deploy a Dask Cluster
- Locust: Load ingestion system
- K3s: Minimalist Kubernetes

Services: Syntactic sugar

```
[...]
with en.Dstat(roles["database"]) as d:
    en.run_command("stress --cpu 4 --timeout 10",
                   roles=roles["db"])
    backup_dir = d.backup_dir
# from here, the metrics are available in csv files in the local machine
# (one per remote hosts)
```

```
[...]
with en.TCPDump(roles["database"],
               networks=networks["mynetwork"],
               options="icmp")
as t:
    # ... do stuff that do stuff on the network ...
    backup_dir = t.backup_dir
# from here, the pcap files are available in the local machine
# (one per remote hosts)
```

Experimental code so far



```
def deploy(provider, conf, netem="delay 10ms rate 1gbps")
  hosts, networks = provider(conf).init()

  # network emulation
  n = Netem(netem,
            hosts=hosts["database"],
            networks=networks["db-net"])
  n.deploy()

  with Monitoring(collector=hosts["util"],
                  agent=hosts["database"],
                  ui=hosts["util"],
                  network=networks["mon-net"])
    # Deploy the tbts and the clients
    # ...
    # -> specific code <-
```

Tasks

Conclusion

- Currently its hard to bootstrap new experiment with complex distributed systems
  - Validating an hypothesis requires to build an experimental artifact
  - Could be difficult to do if built from scratch
  - EnOSlib provides 5 concepts to support / mutualize experimentation practice
    - ★ providers / resources / roles / modules / services
- More practicaly:
  - Check if EnOSlib can help you :)
  - (Doc) `https://discovery.gitlabpages.inria.fr/enoslib/`
  - (Chat) `https://framateam.org/enoslib`