A WORKFLOW-BASED NOTEBOOK INTERFACE FOR EXPERIMENTS IN CLOUDLAB AND POWDER

by

Axe Tang

A Senior Thesis Submitted to the Faculty of
The University of Utah
In Partial Fulfillment of the Requirements for the

Degree of Bachelor of Science

In

Computer Science

Approved:

Eric Eide Thesis Faculty Supervisor	Mary Hall Director, Kahlert School of Computing
Jim de St. Germain Director of Undergraduate Studies	

April 2024

ABSTRACT

Notebooks, because of their interactive nature and portability, have been widely used in both academia and industry to build experimental processes, encode workflows for data analysis, and present results. Researchers have even used notebooks for conducting experiments on network testbeds. However, there are some limitations to this way of using notebooks. Although notebooks can encode workflows, the encoding is not flexible or capable of orchestrating complex experiments. Our hypothesis is that separating notebook workflow and experiment workflow allows users to better coordinate experiment steps, explore different procedures of an experiment, and benefit from the portability and reliability of workflow systems. To test our hypothesis, we built a novel workflow-based notebook interface for conducting experiments in CloudLab and POWDER. This interface, which incorporates a DSL for workflow description, an execution manager, and distributed execution engines, allows fine-grained control over experimental procedures, complex orchestration, and fault-recovery ability from a notebook. We demonstrate and evaluate our prototype by converting a selection of POWDER experiments to notebook-based experiments, and we present lessons learned.

CONTENTS

LIST OF FIGURES

ACKNOWLEDGMENTS

I wish to thank Prof. Eric Eide for his advising, Kirk Webb and Dustin Maas for recommending POWDER profiles, and many other Flux Research Group members.

CHAPTER 1

INTRODUCTION

Computational notebooks, also known as scientific notebooks, are powerful tools that allow data scientists, researchers, and programmers to combine code, data, documentation, and visualization into a single document. Because of their interactive nature and portability, notebooks have been widely used in both academia and industry to describe workflows [2, 19].

Workflows are declarative ways of specifying the high-level logic of an application. They are composed of multiple individual logical tasks that are part of an ultimate goal [20], which are useful for describing network experiments. For example, a notebook may have one component that runs some experimental procedures, one component that collects the network performance data, one component that does data processing, and a final component that visualizes the data. Usually, the execution model of a notebook *encodes* a workflow.

Some testbeds, such as Chameleon [1] and Grid'5000 [4], support notebook-based interfaces. Notebook-based interfaces allow users to describe and run experiments on testbeds directly from a notebook.

However, there are limitations and pain points to using scientific notebooks as workflows. These include convoluted setup processes, poor reliability, and painful exploration and analysis processes [6]. Reproducing the results of a notebook can be challenging [19] because cells can be skipped or executed out of order, and the dependencies of an experiment may not be installed [18]. As a result, when a user tries to reproduce an experiment that is encoded as a notebook, they might run into errors or obtain different results. Some notebook systems, such as Livebook [17], partially address this issue by only allowing top-down order execution. This restriction sacrifices the flexibility of experimenting with different combinations of execution orders.

To address the issues mentioned above, we propose a research prototype for conducting experiments on CloudLab [9] and POWDER [5] that combines a notebook interface with an explicit and featureful workflow system. Compared to the traditional notebook experiment setup, where the notebook *is* the workflow of an experiment, the prototype decouples the experiment's workflow from the notebook. A user can define modular and reusable workflows that are managed by an execution engine that is distinct from the execution of the notebook. For example, a user can define "breakpoints" within a workflow and step over individual tasks, which are processes executed natively on CloudLab nodes. The results of workflow tasks are captured and relayed back to the notebook for visualization and analysis via the execution engine. In our research prototype, a notebook *defines and drives* a workflow for an experiment. We believe this improves the flexibility of running experiments and testing different configurations of notebooks, the ability to build on top of existing notebooks while retaining reproducibility, and the reliability of the system when an unexpected crash happens.

We hypothesize that *separating workflows and notebooks enable users to explore different procedures of an experiment while benefitting from the portability and reliability of workflow systems*. We expect the prototype to have the same capability as the traditional methods of running experiments on CloudLab and POWDER. To test the capabilities of the prototype, we converted three existing POWDER experiments to use the prototype. We evaluate its ability to support experiments and we present lessons learned.

CHAPTER 2

BACKGROUND

This section briefly introduces some key concepts that will appear in later sections of the proposal.

2.1 Elixir and Livebook

Elixir [10] is a functional, concurrent, and fault-tolerant high-level programming language based on Erlang [11]. It is used to build distributed and fault-tolerant applications. Some applications that use Elixir as their building blocks are Discord, Duffel and V7. They choose to use Elixir because of its high scalability, great orchestration ability, and reliability [16].

Livebook [17] is a system for creating interactive and collaborative notebooks built on top of the Elixir programming language and Phoenix framework. A sample notebook created with Livebook is shown in Figure 2.1. Some of the features that Livebook supports include remote code execution, built-in secret management, and an interface for creating new kinds of notebook cells, which are called "smart cells." Cells in Livebook can only be executed from top to bottom and Livebook tracks the state of the notebook, which makes the execution of the notebook predictable. This makes it a good candidate for our research.

2.2 Testbeds

In the field of computer science, testbeds are used to conduct experiments involving actual hardware and software. They provide a controllable yet realistic environment to test the behaviors of components without having them deployed into a production environment. Some famous testbed examples are CloudLab [9], Chameleon [1], DETER [3], Grid'5000 [4], and POWDER [5]. On CloudLab and POWDER, users create profiles to describe what resources they need for a specific experiment. The testbeds automatically provision and allocate resources according to users' profiles. Once all nodes are ready for

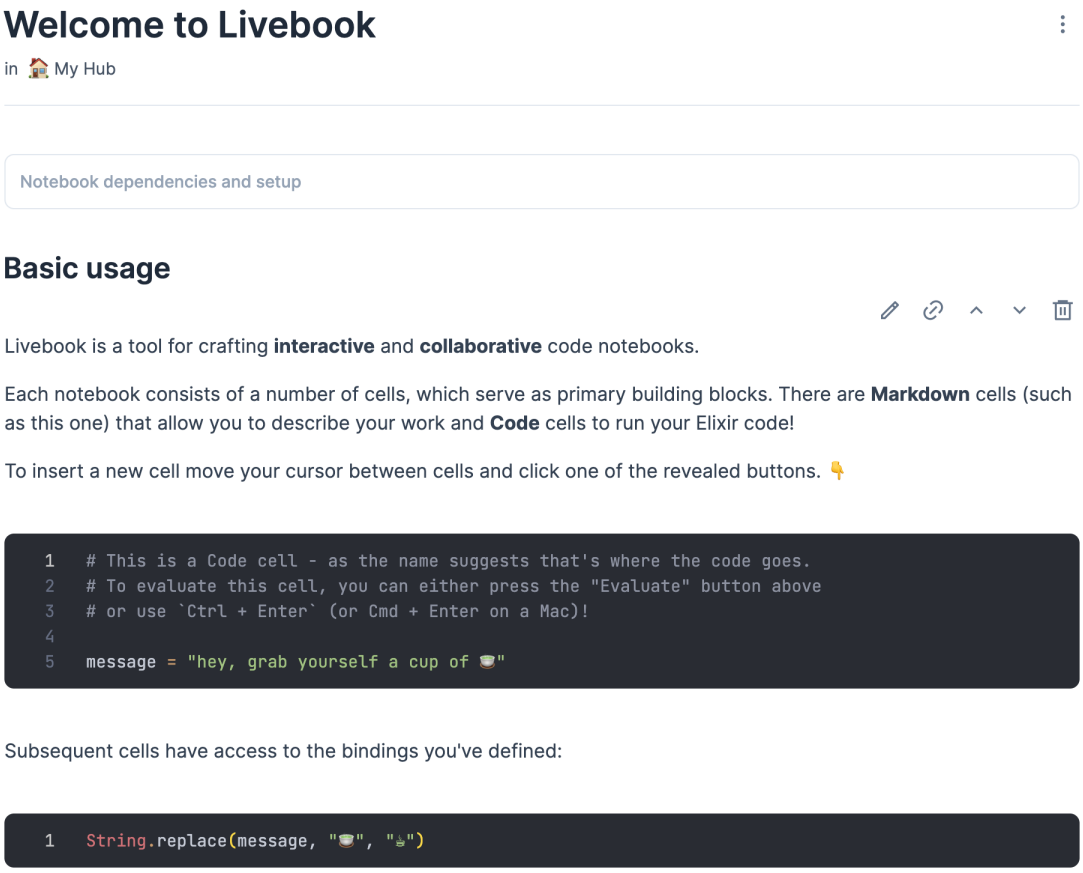

Figure 2.1. Livebook Interface

the experiment, a user can start experimenting on the nodes. The user must coordinate the execution steps of their experiment across multiple nodes, e.g., start a server before starting the client.

2.3 Workflows

Typically, a workflow is a "well-defined, and possibly repeatable, pattern or systematic organization of activities designed to achieve a certain transformation of data" [20]. In other words, workflows are driven largely by data. As the different steps of a workflow are executed, data are transformed into other data that can be used by later steps and processes. A user will get a data product at the end of workflow execution.

The dataflow model of workflows is fundamentally different from the model that our prototype uses, however. In the prototype, the driver is not data but procedures and orchestration among different elements of the experiment. Even though some data may be generated throughout the experiment, obtaining some data as the final product is seldomly the goal, since the most interesting part of any network and system experiment is the interaction among various components.

CHAPTER 3

RELATED WORK

Workflow management systems and orchestration tools enable researchers to run large-scale and complicated tasks. However, these orchestration tools and workflow management systems are not quite applicable to CloudLab and POWDER experiments. They are also not similar to our prototype. In this section, we discuss how our system is different from previous works.

3.1 Computational Notebooks

The idea of integrating computational notebooks into testbeds has been previously explored. Chameleon, a cloud testbed, supports the integration of Jupyter notebooks [1]. In the Chameleon cloud, experimenters can use notebooks as an interface to develop experimental environments, execute their experiments, and analyze their experiment results. Similarly, Grid'5000, a large-scale reconfigurable testbed in France, also supports using Jupyter notebooks for conducting various experimental processes on that testbed [4]. Both Chameleon and Grid'5000 use notebooks as experiment drivers: in other words, the notebooks *are* the experimental workflows. Even though this could improve the reproducibility of experiments to some degree and provide a better user experience for experimenters, it does not fully take advantage of the interactive environment provided by notebooks nor does it provide the reliability and portability that are necessary for experiments.

Glinda [8] is a research prototype that combines live programming, graphical user interfaces (GUIs), and a domain-specific language (DSL). It provides an easy-to-use, expressive, and reactive environment for data scientists. Users can express data-science-related tasks that are described by an expandable set of pre-defined "recipes" using the DSL in Glinda.

Among all the prior workflow systems and orchestration systems, Glinda is the most relevant to our prototype. Both use a YAML-syntax DSL to describe atomic steps and aim

to solve some frustrations with computational notebooks. With Glinda focusing more on the experiences of data scientists, our prototype emphasizes enhancing the orchestration and reliability of notebook-controlled network experiments. The DSL in Glinda has a higher level of abstraction because the ultimate goal is to obtain or visualize data, which concerns little about *what* each step does. Unlike Glinda, which abstracts away low-level details of each task, our system gives users fine-grain control over each experimental step, as network experiments often mandate *how* different components interact.

3.2 Workflow Management Systems

Numerous workflow management systems can turn workflows into tasks that execute on cross-institutional computation grids, including Pegasus [7] and Condor-G [12]. Besides executing workflows, they are responsible for monitoring and managing tasks across different administrative domains and nodes. Workflow management systems must consider task failures, as stated in many workflow management systems papers [7, 12, 14]. Workflow management systems, such as Pegasus and Condor-G, usually have a different form representing workflows for the execution compared to the original user input, which allows workflow engines to optimize input workflows.

Compared to Pegasus, the workflow system in our prototype does not have a separation between the workflow description and the execution environment description, which is largely due to the nature of experiments running on CloudLab and POWDER. First, users have to specify what kind of hardware they want to use to run their experiments on CloudLab and POWDER. Second, CloudLab and POWDER experiments generally have descriptions that *are* the experiments. In other words, *how* things happen in a CloudLab or POWDER experiment is critical, such as the order of commands executed on nodes. Hence, approaches similar to Pegasus's compile-time or run-time optimization of workflow descriptions are detrimental to the experiments.

Compared to Condor-G, which is about harnessing resources in domains as if they all belong to one domain [12], our research prototype focuses on managing resources within a single domain, even though both address the initialization, monitoring, and managing execution of tasks on different nodes.

3.3 Orchestration Systems

Orchestration systems automatically configure, coordinate, and manage processes or software based on user input. They are somewhat similar to workflow management systems, but are essentially different because they do not transform data nor are they driven by data. Instead, they are "configuration-driven" [13]. Unlike workflow management systems, which are rarely concerned about the life cycles of tasks, orchestration systems need to carefully take care of the life cycle of each component [13, 14]. For example, SmartFrog [13] is a configuration management framework that allows its users to create, deploy, and manage large-scale distributed systems. It allows such systems to be constructed and managed automatically from the configurations.

In more recent years, there have been attempts to orchestrate complicated experiments on testbeds. For instance, MAGI is an orchestration tool that "translates an experiment specification into execution on an emulation-based testbed with high-level directives for message passing, remote process execution, and failure tracking, for conducting large and complex experiments" [14]. While orchestration systems excel at coordinating activities across many devices, orchestration systems based on static configurations are not flexible or interactive. To change a step in an experiment, a user must create a new configuration and reload the configuration. Our prototype, however, is designed to support this kind of change directly.

CHAPTER 4

METHODOLOGY

To fully take advantage of the interactivity that notebooks can bring to testbeds while maintaining the flexibility of orchestrating complex experiments, we built a research prototype that combines workflows and notebooks in a novel way. This allows the prototype to combine modular, reusable experiment descriptions, the exploration of different experimental steps, and the ability to visualize data via powerful tools that are integrated with the notebook.

4.1 Architecture

The prototype is composed of three components: a parser, an execution manager, and execution engines deployed on CloudLab nodes. The parser and the execution manager are located on the user's machine, as they will be loaded for in-notebook use. Figure 4.1 shows the high-level architecture of the prototype.

The parser converts user-defined experiment workflows into graphs for the execution manager.

The execution manager orchestrates the activities within an experiment and keeps the state of that experiment. It sends the to-be-executed commands to remote execution engines on corresponding nodes according to the order defined by the workflow. It is also responsible for managing "breakpoints," pausing/unpausing the workflow at user-specified steps. Another important role of the execution manager is failure recovery in case some or all execution engines fail. It can re-run some parts or all parts of an experiment based on the user-defined fault recovery scope.

Execution engines, which are located on CloudLab nodes, are responsible for running users' experimental commands and returning command results to the manager. The execution manager uses RPC calls to pass the to-be-executed commands to execution engines.

4.2 The Workflow Language

A user of our prototype defines the workflow of a CloudLab or POWDER experiment by using a custom workflow language. The specification of an experiment's workflow is contained in a cell of a Livebook notebook, as illustrated in Figure 4.4. The workflow language allows a user to use pre-defined interaction patterns and define atomic workflow steps, which are abstracted as "cells." (Note that the cells in our workflow language are different from cells in a notebook.) A cell can be used to execute commands on experimental nodes, set environment variables, and enable complex control flows.

Our workflow language also defines "patterns." A pre-defined pattern combines cells in a particular way while providing anchors of recovery. To use a pre-defined pattern, the user must specify the pattern they desire to use by identifying it with the `is` keyword and providing entry points using the `requires` keyword. The body of a `requires` construct associates the roles defined by a pattern with the names of cells that implement those roles.

A list of cells can be defined using the `cells` keyword. The value of the `cells` keyword should be a map of cells, with the cell name as the key and the specifications of a cell as value.

A user can define various functionalities of a workflow cell, including its target hosts, commands, fault recovery action, output option, and if the cell blocks until its commands return a result. The syntax of our workflow language is based on YAML. A user can use words like `commands` and `targets` as keys and lists of to-be-executed commands and nodes as values for the YAML string.

A user can also define relations between cells. To do that, a user needs to specify the `setup` and `teardown` fields of a cell. As the names of these fields suggest, the cell specified in `setup` will be executed before the `commands` and the cell specified in `teardown` will be executed after the `commands` of this cell. Both `setup` and `teardown` can be recursive. That is, a cell's `setup` and `teardown` can have its own `setup` and `teardown`, which results in a tree of dependencies.

A user can use a cell to set the environment for a set of nodes. This can be done by specifying the target nodes and specifying the environment to be used. An environment is declared as a dictionary of environment variables as keys and their contents as values,

as shown in Figure 4.2. Different environments can be constructed for different nodes, as long as their names start with `env`.

A user can specify a cell's advanced control features as well. A user can specify a cell's on-error recovery anchor, which is the cell that it jumps to whenever an error occurs. This can be defined by setting the `on_error` field of a cell. A user can also compare two values using a cell and define actions when the result is true or false. These kinds of special cells are called "conditional cells." First, a user needs to specify the `is_conditional` field of that cell as true. The user can then define the `test_clause` field, which must contain the left and right values or variables to compare and any of the following strings: `==`, `<=`, `>=`, `<`, `>`. When the cell is executed, the conditional is tested. If it is true, the execution engine will proceed to the cell as specified in `thn_cell`. If the result is evaluated as not true, the content of `els_cell` will be executed.

Lastly, a user can define and use variables. To use a variable in a cell's `commands` section or a conditional cell's `test_clause` section, a user must wrap the variable name with `$()`. To bind a value to a variable, a user can set the `binds_to` field of a cell to the variable name, which will bind the results of a cell to the corresponding variable.

4.3 Intermediate Representation and Workflow Execution

After a user defines a list of cells, the interrelation between each cell, the pattern that the workflow is using, and the entry point(s) of the pattern, the workflow manager can convert the YAML workflow specification into a graph. The workflow manager starts with the user-defined entry point of a pre-defined pattern, where it will inspect the interrelation between this cell and other cells. This inspection is done through an in-order traversal of dependencies. After all dependencies are obtained, cells are converted to vertices and interrelations (`setup` and `teardown`) are converted to edges. One thing to note here is that it is possible for a cell's `setup` or `teardown` to be a conditional cell. To maintain the correct dependency between cells, different branches of a conditional cell are treated as independent trees. The workflow manager will individually traverse through the dependency tree of two branches and update the graph. During the parsing of the YAML workflow specification, the workflow manager also adds fault-recovery anchors based on the pattern that the workflow is using. By default, all cells that belong to one entry point of

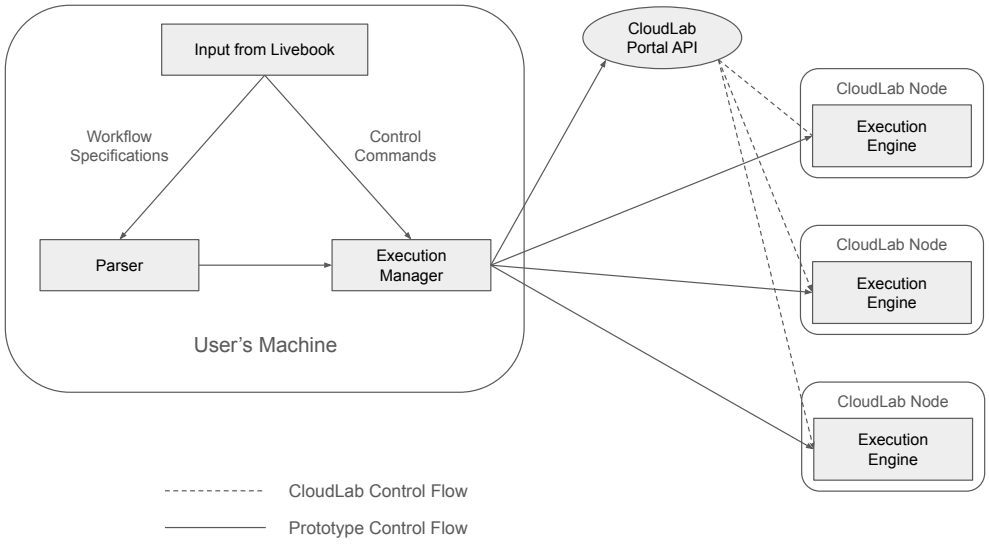

Figure 4.1. Architecture of the research prototype

```
env:
  OPENAIR_HOME: /opt/openairinterface5g
  OPENAIR_DIR: /opt/openairinterface5g
  LTE_AWGN_RESULTS_DIR: /opt/openairinterface5g/openair1/
  SIMULATION/LTE_PHY/BLER_SIMULATIONS/AWGN/AWGN_results
  NR_AWGN_RESULTS_DIR: /opt/openairinterface5g/openair1/
  SIMULATION/NR_PHY/BLER_SIMULATIONS/AWGN/AWGN_results
  NR_MIMO2x2_AWGN_RESULTS_DIR: /opt/openairinterface5g/openair1/
  SIMULATION/NR_PHY/BLER_SIMULATIONS/AWGN/AWGN_MIMO2x2_results
  OPENAIR1_DIR: /opt/openairinterface5g/openair1
  OPENAIR2_DIR: /opt/openairinterface5g/openair2
  OPENAIR3_DIR: /opt/openairinterface5g/openair3
  OPENAIR_TARGETS: /opt/openairinterface5g/targets
  IIOD_REMOTE: 192.168.1.2
  PATH: /usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/
  bin:/usr/games:/usr/local/games:/opt/openairinterface5g/targets/bin
```

Figure 4.2. Sample code for defining an environment for a set of nodes.

a pattern will have their fault-recovery anchor set to that entry point. However, a different fault-recovery anchor can be set by changing the `on_error` field of a cell. Once all cells and interrelations are parsed, the workflow manager will append a special `terminate` vertex at the end of the workflow.

At run time, the workflow manager starts traversing the graph with the first entry point defined by the workflow. The workflow manager maintains a "current vertex" and it traverses the graph by following edges from its current vertex. When the workflow manager traverses to a new vertex, it looks up the cell associated with that vertex and executes the contents of that cell. The workflow manager first plugs in the values of variables in `commands` or `test_clause` and then instructs the remote execution engine(s) to run those commands or evaluate the conditionals. The results of the execution are captured and returned to the workflow manager. If the cell's `commands` or a conditional cell's `test_clause` successfully execute without any errors, the workflow manager traverses to the next cell through the `next` edge. If an error occurs during the execution of a cell, the workflow manager recovers the fault by traversing to the anchor through the `on_error` edge. After a certain number of retries, if the fault persists, the execution manager will break the loop and throw. The graph traversal process is repeated until the `terminate` vertex is reached.

4.4 An Example Experiment

In this section, we use an iPerf3 experiment on CloudLab to demonstrate how the prototype works. iPerf3 is a tool for measuring network performance. Our experiment measures the link performance between two CloudLab nodes by setting up an iPerf3 client-server pair. First, in the notebook, a user tells CloudLab what resources they need for this experiment, as shown in Figure 4.3. This can be done by calling the execution manager's resource allocation methods, as shown in the code snippet below. After the user has defined all the resources they need, they can start the experiment and wait until the experiment is ready.

The user defines a workflow for their experiment using YAML, as shown in Figure 4.4. The YAML definition of the workflow is passed to the parser for the execution engine.

The parser converts the workflow specification string into a graph, as shown in Figure

4.5. The graph is used by the execution manager to determine the order of execution, dependencies, and the scope of fault recovery. Once the execution manager has the graph and initializes its status, it is ready to orchestrate the workflow on CloudLab nodes.

```
pid1 = Experimentwizard.Master.get_experiment_interface("manager1")
# Add credentials, profile name, experiment name, and projects
experiment_name = "PortalAPITesting"
Experimentwizard.Manager.experiment(pid1, experiment_name)
|> Experimentwizard.Manager.credentials(experiment_name, %{
  certfile: "/Users/atang/Documents/notebook-with-powder/portalapi/cloudlab-decrypted.pem",
  cacertfile: "/Users/atang/Documents/notebook-with-powder/portalapi/emulab-ca.pem"
})
|> Experimentwizard.Manager.profile(experiment_name, "PowderSandbox,Repo-Based_Cluster")
|> Experimentwizard.Manager.project(experiment_name, "PowderSandbox")
# Start the experiment, or try to connect to the existing experiment.
Experimentwizard.Manager.connect_or_start(pid1, experiment_name)
```

Figure 4.3. Sample code for resource allocation using the prototype. The user must have access to the project and the profile they wish to use.

```
yaml = """
is: client-server
requires:
  client: client
  server: server
cells:
    client:
        targets:
            - executer@node1.portalapitesting.powdersandbox.emulab.net
        commands:
            - iperf3 -c 192.168.1.2 -t 10
        output: true

    server:
        setup: server_setup
        teardown: server_teardown

    server_setup:
        targets:
            - executer@node2.portalapitesting.powdersandbox.emulab.net
        commands:
            - iperf3 -s
        synchronous: false

    server_teardown:
        targets:
            - executer@node2.portalapitesting.powdersandbox.emulab.net
        commands:
            - pkill iperf3
"""
yaml_result = Experimentwizard.RuntimeManager.parse_yaml_string(yaml)
```

Figure 4.4. Sample code for experiment workflow definition in the research prototype. This specification uses a "client-server" interaction pattern, which organizes client and server cells in a specific execution order. For example, in the client-server pattern, the server starts before the client.

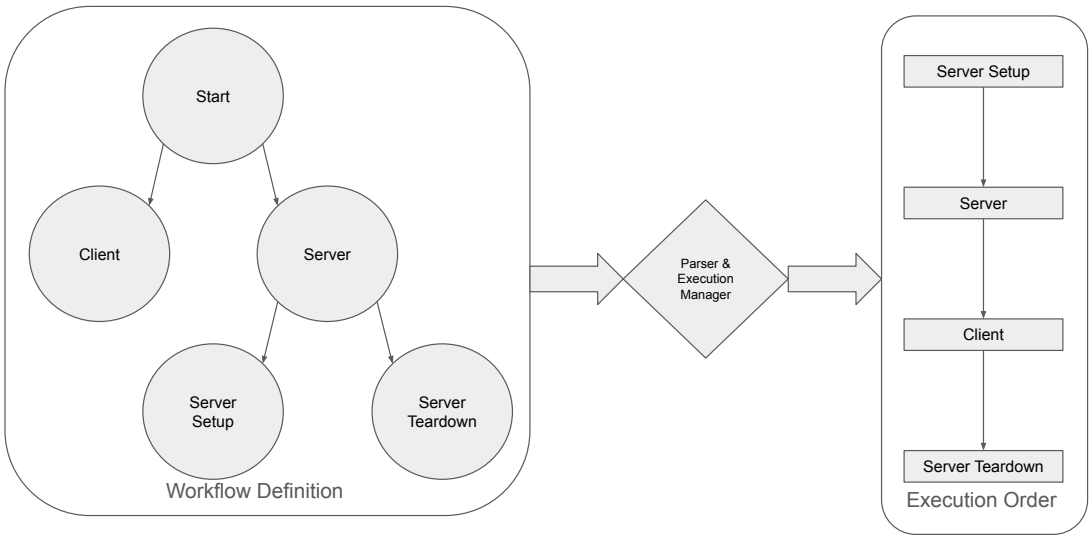

Figure 4.5. The YAML workflow specification is passed to the parser and execution manager to generate the necessary information for workflow execution.

CHAPTER 5

EVALUATION

We hypothesize that separating notebook workflows from experiment workflows allows users to better coordinate experiment steps, explore different procedures of an experiment, and benefit from the portability and reliability of workflow systems. To evaluate our hypothesis, we converted three POWDER-based experiments to research prototype-based experiments. We asked members of the Flux Research Group to recommend POWDER and CloudLab profiles that could potentially be converted to use the research prototype. They suggested two profiles with good potential. Another profile was selected from the hands-on lab sections of the fall 2023 advanced computer networks class. We selected these experiments because they contain detailed, step-by-step instructions, which are hard to find among profiles on POWDER. They also have commands that need to be executed by users, which are suitable for the prototype to automate.

5.1 Converting Profiles

The general process of converting a profile starts with modifying the profile to install bootstraps for our prototype's execution engines on CloudLab or POWDER nodes. The next step is rewriting the instructions for the profile — which are intended to be read and carried out by a user — into an experiment workflow specification using the YAML workflow language. Usually, this step is straightforward because the profiles have already divided commands into different sections.

5.1.1 "Teaching 5G with POWDER and OpenAirInterface5G" Profile

The "Teaching 5G with POWDER and OpenAirInterface5G" profile instantiates an end-to-end 5G network with simulated RF transmission on a single POWDER node using Kubernetes and OpenAirInterface (OAI), an open-source 5G stack. The user can change the noise level in the simulated link to observe its effects. The profile starts with inspect-

ing the OAI components included in the node. Subsequently, the user needs to execute commands for deploying the 5G core network and for building and deploying the radio access network components. After the OAI 5G network is set up, one can start the user equipment (UE) and generate network traffic. Finally, the user can adjust the noise level in the simulated RF link to observe changes in link metrics. The original profile uses a web-based VNC session for soft oscilloscopes to monitor link metrics. However, since our research prototype does not support VNC sessions, we modified the original profile to collect iPerf3 results to visualize link-metric changes in Livebook.

When converting this profile, we realized that the behavior of the `erlexec` library for command execution on nodes is not what we expected. It runs each command in isolation from each other, which means that if a command sets an environment variable, the change will not be propagated to the following commands. We extended our workflow language to allow users to define lists of environmental variables for different nodes. After addressing the issue, we were able to successfully convert the POWDER profile to use the prototype.

5.1.2 "WiMatch-OTA" Profile

The "WiMatch-OTA" profile allocates and deploys the WiMatch system [21] across multiple POWDER nodes. It is a wireless resource matchmaking system that contains an RF measurement system, called Shout, and software for serving user requests. The instructions begin with commands to start the orchestrator on one node. After the orchestrator starts, the user needs to run scripts for over-the-air RF measuring on multiple nodes. Once both the orchestrator and measuring clients are ready, the measurement can be started and takes a few minutes. Finally, the user can analyze measurement results and generate graphs by running a command. Similar to the "Teaching 5G with POWDER and OpenAirInterface5G" profile, this profile also makes use of VNC sessions in order to display generated graphs in separate windows.

5.1.3 "NexRAN-O-RAN" Profile

The "NexRAN-O-RAN" profile deploys a top-to-bottom Open RAN near-real-time RAN intelligent controller (RIC) stack, based on the NexRAN [15] implementation. This stack allows "closed-loop control of a RAN slicing realization in an O-RAN ecosystem" [15]. The

user can change the downlink performance of a UE by assigning it to a different network slice. To use this profile, the user first needs to create a 4G network using srsRAN, which includes starting up the core network, the eNodeB, and the UE. The user can then start the xApp, which is responsible for network slicing. The network slices can be managed by a northbound API, and UEs can be bound to different slices using the same API. Unlike the previous profile, which uses VNC to visualize results, this profile simply uses live iPerf3 sessions as results.

Despite this profile being more complicated than the "Teaching 5G with POWDER and OpenAirInterface5G" profile, we still smoothly converted it to use the research prototype. A challenge in converting this profile was that, because some parts of the 4G network — e.g, the UE — usually takes 20 to 40 seconds to finish setting up, it is not trivial to determine when to execute the next command. With the fault recovery feature of the prototype, we can define a step in the workflow that examines the current UE status and specifies the fault recovery action to itself, as described in Figure 5.1. This ensures the UE is ready to go for upcoming commands.

5.2 Lessons Learned from Converting Profiles

We now discuss some insights we gained when evaluating the prototype. The prototype is capable of encoding CloudLab/POWDER experiments. We successfully converted all of the candidate POWDER profiles to notebook-based profiles. We found separating experimental workflows and notebook workflows provides additional advantages compared to existing methods of orchestrating experiments. It allows better automation, simplifies the process of building and starting components, returns data to the notebook, and provides the ability to orchestrate complex experiments. Fault-recovery ability is crucial in achieving these advantages.

During our evaluation, we noticed that many POWDER profiles assume the use of virtual desktops/SSH sessions. This requires multiple terminals to be open for different tasks, which is cumbersome and error-prone. For instance, in the "Teaching 5G with POWDER and OpenAirInterface5G" profile, a user needs to use one SSH session to start the core, one session to start the gNodeB, another session to start the UE and two other sessions for running iPerf3 client and server. We also noticed that all of the selected profiles

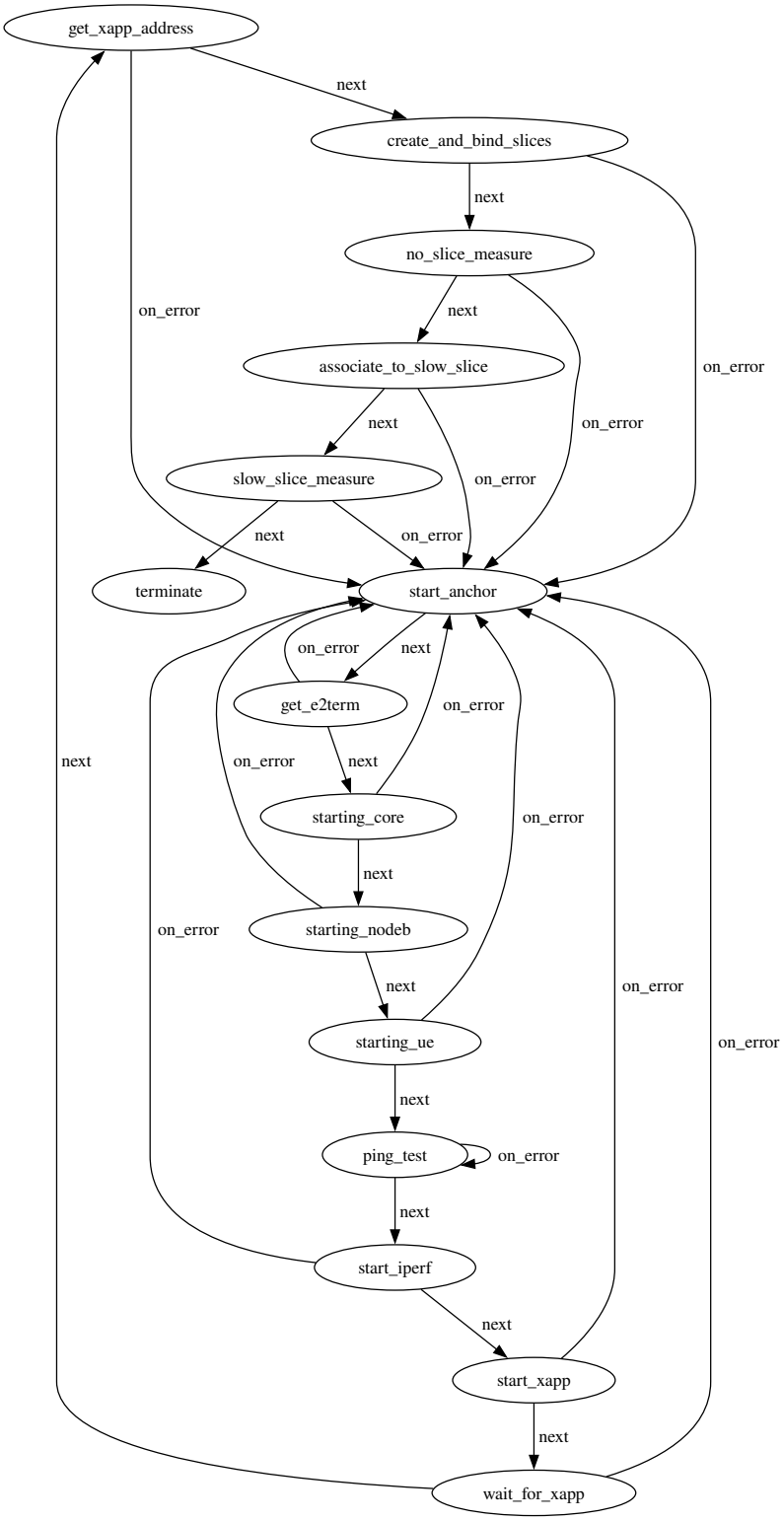

Figure 5.1. The intermediate representation generated by the parser for the "NexRAN-O-RAN" profile. The `ping_test` cell has its `on_error` action pointing to itself for fault recovery.

visualize results in the virtual desktop or SSH sessions, which means no data files are produced for further investigation. Converting these profiles required us to rethink how outputs are produced and collected.

While converting the three profiles, we noticed our workflow language became repetitive. Every time a new workflow cell is created, its `targets` must be specified. We used Elixir's string interpolation feature to reduce repetition in the workflow descriptions.

Our DSL uses YAML, which imposes some constraints on how a command must be formatted. Some special characters in a command must be escaped and it limits commands we can use.

Even though we implemented support for interaction patterns, none of the tested profiles used it, which is also the case for conditional cells and some other advanced control flow features. We expect that these features would be more useful in experiment workflows that are designed from the start to be run in our prototype notebook-based interface.

CHAPTER 6

FUTURE WORK

6.1 A Better Workflow Language

Even though the prototype's YAML-syntax DSL is adequate for describing complex experimental workflows, we believe that it would be beneficial to upgrade our DSL to use a fully custom syntax. This could reduce repetition in the workflow description, remove restrictions on command formatting, and fully utilize the modularity of the workflow description. Expanding our DSL to support allocating resources could also be helpful. Our current prototype uses CloudLab's and POWDER's profiles for resource allocation, which requires the users to write separate codes to describe what kind of hardware resources they want. Allowing users to define their hardware resources in the DSL would provide a seamless environment.

6.2 Streaming Interface

The current design of the prototype returns the output of a cell to the notebook only after the cell has finished executing. Even though such behavior allows a user to obtain a complete data artifact, it limits how results can be displayed. Adding a streaming interface that continuously returns outputs of commands to the notebook may allow better visualization of data, especially combined with Livebook's powerful data visualization tools. For instance, a user may see the iPerf3 throughput drop drastically after a workflow step of changing the network slice.

CHAPTER 7

CONCLUSIONS

Although notebooks encode workflows, the encoding is not flexible or capable of orchestrating complex experiments. Notebook workflows are linear, which hinders their ability to encode control logic. We hypothesized that separating notebook workflows from experiment workflows would allow users to better coordinate experiment steps, explore different procedures of an experiment, and benefit from the portability and reliability of workflow systems. We built a workflow-based notebook interface for conducting experiments on CloudLab and POWDER. To test our hypothesis, we converted three POWDER experiments to notebook-based experiments using our prototype. We found our notebook interface is capable of encoding CloudLab/POWDER experiments and that separating notebook workflows from experiment workflows allows better automation, improves reliability, and gives the ability to orchestrate complex experiments.

REFERENCES

[1] Jason Anderson and Kate Keahey. A case for integrating experimental containers with notebooks. In *2019 IEEE International Conference on Cloud Computing Technology and Science (CloudCom)*, pages 151–158, 2019. doi: 10.1109/CloudCom.2019.00032.

[2] Marijan Beg, Juliette Taka, Thomas Kluyver, Alexander Konovalov, Min Ragan-Kelley, Nicolas M. Thiéry, and Hans Fangohr. Using Jupyter for reproducible scientific workflows. *Computing in Science & Engineering*, 23(2):36–46, 2021. doi: 10.1109/MCSE.2021.3052101.

[3] T. Benzel, R. Braden, D. Kim, C. Neuman, A. Joseph, K. Sklower, R. Ostrenga, and S. Schwab. Experience with DETER: a testbed for security research. In *2nd International Conference on Testbeds and Research Infrastructures for the Development of Networks and Communities, 2006. TRIDENTCOM 2006.*, pages 10 pp. – 388, 2006. doi: 10.1109/TRIDNT.2006.1649172.

[4] Luke Bertot and Lucas Nussbaum. Leveraging notebooks on testbeds: the Grid'5000 case. In *IEEE INFOCOM 2021 - IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, pages 1–6, 2021. doi: 10.1109/INFOCOMWKSHPS51825.2021.9484501.

[5] Joe Breen, Andrew Buffmire, Jonathon Duerig, Kevin Dutt, Eric Eide, Mike Hibler, David Johnson, Sneha Kumar Kasera, Earl Lewis, Dustin Maas, Alex Orange, Neal Patwari, Daniel Reading, Robert Ricci, David Schurig, Leigh B. Stoller, Jacobus Van der Merwe, Kirk Webb, and Gary Wong. POWDER: Platform for open wireless data-driven experimental research. In *Proceedings of the 14th International Workshop on Wireless Network Testbeds, Experimental Evaluation & Characterization*, WiNTECH '20, page 17–24, New York, NY, USA, 2020. Association for Computing Machinery. ISBN 9781450380829. doi: 10.1145/3411276.3412204.

[6] Souti Chattopadhyay, Ishita Prasad, Austin Z. Henley, Anita Sarma, and Titus Barik. What's wrong with computational notebooks? Pain points, needs, and design opportunities. In *Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems*, CHI '20, page 1–12, New York, NY, USA, 2020. Association for Computing Machinery. ISBN 9781450367080. doi: 10.1145/3313831.3376729.

[7] Ewa Deelman, Karan Vahi, Gideon Juve, Mats Rynge, Scott Callaghan, Philip J. Maechling, Rajiv Mayani, Weiwei Chen, Rafael Ferreira da Silva, Miron Livny, and Kent Wenger. Pegasus, a workflow management system for science automation. *Future Generation Computer Systems*, 46:17–35, 2015. ISSN 0167-739X. doi: https://doi.org/10.1016/j.future.2014.10.008.

[8] Robert A DeLine. Glinda: Supporting data science with live programming, GUIs and a domain-specific language. In *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems*, CHI '21, New York, NY, USA, 2021. Association for Computing

Machinery. ISBN 9781450380966. doi: 10.1145/3411764.3445267. URL https://doi.org/10.1145/3411764.3445267.

[9] Dmitry Duplyakin, Robert Ricci, Aleksander Maricq, Gary Wong, Jonathon Duerig, Eric Eide, Leigh Stoller, Mike Hibler, David Johnson, Kirk Webb, Aditya Akella, Kuangching Wang, Glenn Ricart, Larry Landweber, Chip Elliott, Michael Zink, Emmanuel Cecchet, Snigdhaswin Kar, and Prabodh Mishra. The design and operation of CloudLab. In *2019 USENIX Annual Technical Conference (USENIX ATC 19)*, pages 1–14, Renton, WA, July 2019. USENIX Association. ISBN 978-1-939133-03-8. URL https://www.usenix.org/conference/atc19/presentation/duplyakin.

[10] Elixir Core Team. Elixir, 2024. URL https://elixir-lang.org/.

[11] Erlang Core Team. Erlang, 2024. URL https://www.erlang.org/.

[12] J. Frey, T. Tannenbaum, M. Livny, I. Foster, and S. Tuecke. Condor-G: a computation management agent for multi-institutional grids. In *Proceedings 10th IEEE International Symposium on High Performance Distributed Computing*, pages 55–63, 2001. doi: 10.1109/HPDC.2001.945176.

[13] Patrick Goldsack, Julio Guijarro, Steve Loughran, Alistair Coles, Andrew Farrell, Antonio Lain, Paul Murray, and Peter Toft. The SmartFrog configuration management framework. *SIGOPS Oper. Syst. Rev.*, 43(1):16–25, Jan 2009. ISSN 0163-5980. doi: 10.1145/1496909.1496915.

[14] Alefiya Hussain, Prateek Jaipuria, Geoff Lawler, Stephen Schwab, and Terry Benzel. Toward orchestration of complex networking experiments. In *13th USENIX Workshop on Cyber Security Experimentation and Test (CSET 20)*. USENIX Association, August 2020. URL https://www.usenix.org/conference/cset20/presentation/hussain.

[15] David Johnson, Dustin Maas, and Jacobus Van Der Merwe. NexRAN: Closed-loop RAN slicing in Powder -a top-to-bottom open-source Open-RAN use case. In *Proceedings of the 15th ACM Workshop on Wireless Network Testbeds, Experimental Evaluation & CHaracterization*, WiNTECH '21, page 17–23, New York, NY, USA, 2021. Association for Computing Machinery. ISBN 9781450387033. doi: 10.1145/3477086.3480842. URL https://doi.org/10.1145/3477086.3480842.

[16] José Valim. Real time communication at scale with elixir at discord, 2020. URL https://elixir-lang.org/blog/2020/10/08/real-time-communication-at-scale-with-elixir-at-discord/.

[17] Livebook Core Team. Livebook, 2024. URL https://livebook.dev/.

[18] João Felipe Pimentel, Leonardo Murta, Vanessa Braganholo, and Juliana Freire. A large-scale study about quality and reproducibility of Jupyter notebooks. In *2019 IEEE/ACM 16th International Conference on Mining Software Repositories (MSR)*, pages 507–517, 2019. doi: 10.1109/MSR.2019.00077.

[19] Adam Rule, Aurélien Tabard, and James D. Hollan. Exploration and explanation in computational notebooks. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems*, CHI '18, page 1–12, New York, NY, USA, 2018. Association for Computing Machinery. ISBN 9781450356206. doi: 10.1145/3173574.3173606.

[20] K. Thramboulidis, B. C. Lai, J. Cao, and Domenico Talia. Workflow systems for science: Concepts and tools. *ISRN Software Engineering*, 2013:404525, 2013. doi: 10.1155/2013/404525. URL `https://doi.org/10.1155/2013/404525`.

[21] Kirk Webb, Sneha Kumar Kasera, Neal Patwari, and Jacobus Van der Merwe. Wimatch: Wireless resource matchmaking. In *IEEE INFOCOM 2021 - IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, pages 1–6, 2021. doi: 10.1109/INFOCOMWKSHPS51825.2021.9484538.