THE UNIVERSITY OF SYDNEY

Lowering the barrier to using computational notebooks

Computer Science Research Methods
INFO4990
Semester 1, 2023
Assignment 2

Tanaka Chitete

Supervisor:
Dr. Rahul Gopinath

The University of Sydney
School of Computer Science

October 31, 2023

Contents

Chapter 1

Literature review

Reproducibility represents the most essential aspect of any scientific discipline. Disseminating research by means of a computational notebook is meaningful only to the extent which the notebook itself can be reproduced. Taking this into account, we have identified that the process of achieving reproducibility and extensibility with Jupyter Notebooks [1] consists of replicating partial computation, resolving dependencies, and supporting collaboration.

1.1 Replicating partially-computed notebooks

Reviewed works achieve notebook replication through either a *snapshot-based* or *provenance-based* approach. Snapshot replication aims to replicate the state of a notebook at a specific point in time during the computational workflow. Provenance replication, on the other hand, seeks to replicate all artifacts pertaining to the computational workflow—inputs, processes, and outputs—and document their evolution throughout the lifetime of an experiment.

1.1.1 Snapshot replication

Wannipurage et. al. [2] utilised the magic command functionalities provided by Jupyter Notebook [3] alongside a modified IPython kernel to archive and restore partially-computed notebooks. To archive a notebook, firstly, their solution identifies all variables, functions, and imports present within the run-time environment. Secondly, it filters these entities to derive the smallest subset pertinent to the replication of the notebook. Finally, the resultant subset is serialised and archived alongside data files and library dependencies. To restore a notebook, firstly, data files are copied into their respective directories and Python libraries are reinstalled. Secondly, the IPython kernel is initialised. Finally, the serialised session entities are deserialised and injected into the kernel. Complete internal state replication is achieved in this work. Through subsequent performance analysis, it is shown that their solution introduces minimal time overhead through its mechanisms. However, coupling between their solution and the operating system of the host machine may cause issues replicating data files on different operating systems—necessitating the usage of a container runtime in cross-platform pipelines.

Juric et. al. [4] leveraged a modified JupyterHub `Spawner` class [5] to checkpoint and restore an active Jupyter Notebook session. To checkpoint a notebook, Elsa hijacks the Spawner's `stop()` method [6] to persist the complete state of the notebook, preserving all internal entities—variables, functions, and libraries. To restore a notebook, it hijacks the associated `start()` method [7] to restore it notebook from a previously saved checkpoint. Juric et. al. achieve state replication in their work. Subsequent performance and cost analyses detail that not only is Elsa performant, it also promises to significantly lower operating costs for vendors— while maintaining existing user experience levels. Furthermore, implementing Elsa as a Jupyter Notebook extension with simple play/pause buttons positioned on the GUI further promotes usability. However, Elsa's current implementation is incompatible with a variety of cloud service providers. That said, achieving compatibility has already been identified as an direction for future work.

1.1.2 Provenance replication

Pimentel et. al. [8, 9] made use of a variety of technologies and their associated capabilities to collect, analyse, and manage provenance data. To collect provenance, firstly, noWorkflow captures the structure of a notebook by constructing an abstract syntax tree (AST). Secondly, it captures library dependencies and information about the host machine. Finally, upon execution, it captures the output of the notebook's cells. To analyse provenance, the solution presents the user with a multitude of operations through a Jupyter Notebook extension. To manage provenance, noWorkflow implements local versioning—facilitating the comparison of results from differing trials, creation of backups for previously-run trials, and restoration of previous trials. The authors seemingly achieve complete provenance capture—internal entities, accessed files, and libraries. However, they fail to conduct a usability study or even quantify noWorkflow's performance by providing a performance analysis. Instead, they propose and navigate a hypothetical data science task in an effort to plead to its practicability.

1.2 Resolving dependencies

Existing research meets the end of dependency resolution through either *static* or *dynamic* dependency resolution. Static resolution aims to resolve library dependencies prior to the execution of a notebook. Dynamic resolution, on the other hand, seeks to resolve dependency issues during execution.

1.2.1 Static resolution

Wang et. al. [10] utilised a self-implemented API bank populated with packages from in the Python Package Index [11] to resolve dependencies when migrating notebooks to different computational environments. Firstly, SnifferDog builds an AST using the cells of the notebook and performs a traversal to identify library APIs. Secondly, it queries its API bank to find the set of packages which provide these APIs. Finally, SnifferDog outputs these APIs in the form of an environmental dependency file to be executed in the new environment. Dependency resolution is achieved to a remarkably high degree. Ensuing performance analysis shows that SnifferDog was able to restore the execution environments of more than 80% of the notebooks it was executed against. It achieves this while introducing minimal time overhead during execution—processing each input notebook in less than 4 seconds on average. Furthermore, the extensibility of the API bank suggests that SnifferDog's performance will improve as more libraries are mapped to their APIs.

1.2.2 Dynamic resolution

Zhu et. al. [12] utilised a machine-learning driven artificial intelligence model to iteratively fix dependency issues at runtime. Firstly, upon the raising of an exception, Relancer identifies the deprecated API and its associated fix. Secondly, it crawls GitHub repositories and API documentation to aggregate and rank candidate upgrade APIs. Finally, Relancer recreates the notebook using each candidate API, executes it, and verifies if it fixes the current exception. This work achieves dependency resolution to a relatively unsatisfactory degree. Relancer was able to successfully restore the executability of only half of the input notebooks it received. However, for those it was able to restore, it took a median of 17 seconds to restore executability. It is important to note that the complexities of resolving dependency issues often results in humans expending significant effort to work towards their resolution. With this in mind, Relancer may prove to be useful in resolving more trivial issues but substantial research and development will need to be undertaken to improve its performance with non-trivial matters.

1.3 Supporting collaboration

Prevailing literature achieves notebook collaboration through implementing either *version control* or *synchronous editing*. Version control is the process of tracking and managing changes to source code and its related artefacts. Synchronous editing, on the other hand, is an editing format where multiple people can edit the same document simultaneously.

1.3.1 Version control

Kery et. al. [13] utilised Electron [14] to implement version control on an actively-running Jupyter Notebook. For a particular version of a notebook, Verdant represents the notebook in its entirety as a tree-like data structure. The root of the tree is the notebook itself and its descendants are its Markdown and Python cells. To capture a version of a notebook, Verdant creates a new node whenever a cell is modified. To manage notebook versions, it implements a Jupyter Notebook extension which allows users to compare, contrast and restore different cell versions. Complete version control for Jupyter Notebooks is achieved in this work. Ensuing usability studies demonstrate that Verdant adequately services data scientists' fundamental need for version control in Jupyter-Notebook-based environments. However, the short duration of the studies may not accurately represent Verdant's long-term impact as professional data science work is typically conducted over an extended period of time—days, weeks, or, in some cases, months.

1.3.2 Synchronous editing

Wang et. al. [15] reported on the affect of synchronous editing on collaboration within data science teams. Firstly, they shared a formative survey with one cohort of data scientists to understand their experiences with collaboration on past projects. Secondly, they conducted an observational study involving a second cohort of data scientists working in pairs on a hypothetical data science task. This study was performed with the goal of comparing and contrasting JupyterLite and Google Colab [16]—a real-time Jupyter Notebook solution. It was found that synced notebooks can present a host of unique complications—inadvertent interference, lack of awareness, and privacy concerns. Unsurprisingly, these issues may have potentially grave implications in contexts where privacy and security are paramount—educational contexts and large-scale production pipelines, for example. Furthermore, the sheer degree of work required to successfully design and implement such functionality only further dissuades both users and developers from engaging with synchronous-editing, at all.

Chapter 2

Research outline

2.1 Contributions

I intend to make the following contributions:

1. Encapsulation of a JupyterLite environment within a self-contained HTML file

2. Construction of an encapsulated environment using script-based methods

2.2 Methods

To encapsulate a JupyterLite environment, firstly, I intend to clone the JupyterLite 0.1.0 distribution [17]. Secondly, I intend to embed all local files present inside of the repository into a singular HTML file. Finally, I intend to embed all remote files requested over the network into the same HTML file.

To construct an encapsulated JupyterLite environment, firstly, I intend to write a script that will embed the contents of an input Jupyter Notebook into an encapsulated environment. Secondly, I intend to write another script which will install dependencies contained in an input environment dependency file into the encapsulated environment.

To make my development artifacts reproducible, I will use Vagrant [18]—a virtualisation environment—and provide thorough documentation to accompany contributed lines of code.

Bibliography

(1) *Introduction to the JupyterLab and Jupyter Notebooks*, Project Jupyter, n.d.

(2) D. Wannipurage, S. Marru and M. Pierce, “A Framework to capture and reproduce the Absolute State of Jupyter Notebooks”, *Proceedings of the Practice and Experience in Advanced Research Computing conference*, 2022.

(3) *Defining custom magics*, IPython Development Team, n.d.

(4) M. Juric, S. Stetzler and C. T. Slater, “Checkpoint, Restore, and Live Migration for Science Platforms”, *Proceedings of the Astronomical Data Analysis Software and Systems conference*, 2021.

(5) *Spawners*, Project Jupyter, n.d.

(6) *Spawner.stop()*, Project Jupyter, n.d.

(7) *Spawner.start()*, Project Jupyter, n.d.

(8) J. F. Pimentel, L. Murta, V. Braganholo and J. Freire, “noWorkflow: a tool for collecting, analyzing, and managing provenance from Python scripts”, *Proceedings of the Very Large Database endowment*, 2017.

(9) J. F. Pimentel, V. Braganholo, L. Murta and J. Freire, “Collecting and Analyzing Provenance on Interactive Notebooks: when IPython meets noWorkflow”, *Proceedings of the Theory and Practice of Provenance conference*, 2015.

(10) J. Wang, L. Li and A. Zeller, “Restoring Execution Environments of Jupyter Notebooks”, *Proceedings of the International Conference on Software Engineering*, 2021.

(11) *Find, install and publish Python packages with the Python Package Index*, Python Software Foundation, n.d.

(12) C. Zhu, R. K. Saha, M. R. Prasad and S. Khurshid, “Restoring the executability of Jupyter Notebooks by automatic upgrade of deprecated APIs”, *Proceedings of the International Conference on Automated Software Engineering*, 2021.

(13) M. B. Kery and B. A. Myers, “Interactions for untangling messy history in a computational notebook”, *Proceedings of the Visual Languages and Human-Centric Computing symposium*, 2018.

(14) *Build cross-platform desktop apps with JavaScript, HTML, and CSS*, OpenJS Foundation, n.d.

(15) A. Y. Wang, A. Mittal, C. Brooks and S. Oney, "How data scientists use computational notebooks for real-time collaboration", *Proceedings of the Human-Computer Interaction conference*, 2019.

(16) *Welcome to Colab!*, Google, n.d.

(17) *JupyterLite 0.1.0*, Project Jupyter, n.d.

(18) *Development environments simplified*, HashiCorp, n.d.