Script of Scripts
Polyglot Notebook and Workflow System

Gao Wang
Department of Human Genetics
The University of Chicago
Chicago, IL
gaow@uchicago.edu

Man Cheng Leong
Children’s Environmental Health Initiative (CEHI)
Rice University
Houston, TX
henryleong@rice.edu

Bo Peng∗
Department of Bioinformatics and Computational Biology
The University of Texas, MD Anderson Cancer Center
Houston, TX
bpeng@mdanderson.org

ABSTRACT
Computationally intensive disciplines such as computational biology often use tools implemented in different languages and analyze data on high-performance computing systems. Although scientific workflow systems can powerfully execute large-scale data processing, they are not suitable for ad hoc data analysis. Interactive tools such as Jupyter Notebook can be used for data exploration, but it remains difficult to work with multiple scripting languages and to streamline analysis for batch data processing. To bridge the gap between interactive and workflow systems we developed Script of Scripts (SoS), which consists of a polyglot notebook that supports multiple languages in a single notebook and a workflow engine that provides an intuitive syntax for multi-style workflows and a unified interface for executing tasks on a variety of computing platforms. By allowing the use of SoS workflow engine in a polyglot notebook environment, SoS provides a unified environment for both interactive data analysis and batch data processing.

KEYWORDS
scientific workflow system, interactive data analysis software

Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for third-party components of this work must be honored. For all other uses, contact the owner/author(s).

ićket


1 INTRODUCTION
Computationally intensive disciplines such as computational biology often require exploitation of a variety of tools implemented in different programming languages and analyses of large data sets using high-performance computing systems. Although scientific workflow systems can powerfully organize and execute large-scale data-analysis processes, creating and maintaining such workflows usually comes with nontrivial learning curves and engineering overhead [6]. More fundamentally, most workflow systems are not suitable for exploratory data analysis, a type of analysis that relies heavily on interactive and ad hoc and typically also of polyglot nature (involving the use of multiple scripting languages). Interactive data analysis tools such as Jupyter Notebook supports a large number of kernels and magics for executing scripts in other languages, but it remains difficult to use multiple scripting languages and to use workflow systems to streamline the analysis of large data sets in interactive analysis environments. There is clearly a need from the community to develop a workflow system suitable for both batch data processing and ad hoc analysis, and tailored for domain experts in data science with rich experience in analysis methodology but less in software engineering [1, 9].

To address these issues, we developed Script of Scripts (SoS), a Python3-based workflow engine with a Jupyter frontend that allows the use of multiple kernels in one notebook. As a workflow engine, SoS provides an intuitive syntax for creating workflows in process-based, outcome-oriented (makefile style), and mixed styles, as well as a unified interface for executing and managing tasks on a variety of computing platforms with automatic synchronization of files among isolated file systems. As a polyglot notebook [7], SoS allows the use of multiple kernels in a single Jupyter notebook. It provides a unified platform on which workflows composed in mixed languages can be easily composed with documentation narratives, executed cross-platform, and shared and reproduced.

Researchers will benefit from the SoS workflow system and Jupyter kernel the flexibility to use their preferred tools for tasks without having to worry about data flow, and can perform light interactive analysis while executing heavy remote tasks simultaneous in the same notebook in a neat and organized fashion.

2 POLYGLOT NOTEBOOK
2.1 Multi-kernel data communication
SoS Notebook consists of a new kernel and a number of front-end extensions of the Jupyter Notebook platform [4]. The SoS kernel acts as a proxy to the SoS workflow engine and a hub between Jupyter and more than 60 existing Jupyter kernels. In addition to magics such as %expand and %capture to compose scripts and capture output from all Jupyter kernels, SoS allows exchange of variables between kernels of supported languages. Other useful features of the SoS kernel of SoS includes a side-panel that allows scratch execution of statements, preview of files and expressions, and line-by-line execution of statements in cells. This unique combination allows users to analyze data using multiple scripting languages.
in one notebook, and, if needed, convert scripts to workflows to analyze large amount of data on remote systems.

A more powerful data exchange method is provided to exchange variables among kernels for supported languages (Bash, JavaScript, Julia, MATLAB, Octave, Python2 and 3, R, SAS and TypeScript). Because of large differences in datatypes among scripting languages, SoS transfers variables through the creation of independent homonymous variables of the most similar datatypes in the destination language.

2.2 Dynamic reports and workflow system IDE

SoS Notebook facilitates the generation of reports from multi-language data analysis by providing magics to capture and render results from multiple kernels, and tags, shortcuts, and templates to generate reports in HTML format. For example, the %sos_report template allows generation of HTML reports that display only selected material. A reader can therefore focus immediately on the core messages of a report and display more details if needed. Output from the %preview magic can be included to report large datasets as dynamic tables and plots.

In addition to being a self-contained data analysis environment, SoS Notebook also serves as an execution and management console for the SoS workflow system, as will be discussed next.

3 WORKFLOW SYSTEM

3.1 Multi-style workflow syntax

The SoS workflow system empowers daily research applications ranging from neatly consolidating fragments of scripts into a single executable source file to executing sophisticated workflows that harness the power of multiple remote computing environments in an environment that streamlines the entire process of script development, interactive data analysis, batch data processing, and reporting and sharing of results. In contrast with other workflow tools, SoS enhances existing scripts with workflow functionalities with little to no modification of the scripts themselves. The simplest of SoS workflow is merely verbatim inclusion of existing scripts. Scripts can further be numerically ordered, or be assigned simple headers, to reflect simple workflow-like execution logic.

To build more sophisticated workflows, SoS extends from Python 3.6 and allows for the use of arbitrary Python statements and modules inside an SoS workflow. Like most other workflow systems it specifies workflow step input, output, dependent targets (if applicable); together they determine how a workflow should be created from the sections. Different from other workflow systems, SoS has a multi-style design to provide more than one way to define a workflow. The most intuitive SoS workflow style is the process-oriented style as used with some other pipeline tools [2, 8]. In its simplest form, this style sequentially executes a series of numerically ordered steps. Although such a workflow by default has a directed acyclic graph (DAG) with a single linear execution path, users can introduce dependencies (edges of DAGs) by defining input, output, and dependent targets in each step. Another workflow style that SoS supports is the outcome-oriented or Makefile style, which relies on implicit wild-card idioms introduced with the Make utility, and adopted by some pipeline tools [3, 5]. Harnessing the power of wild-card pattern matching, the outcome-oriented style implicitly determines dependencies, and automatically builds and executes DAGs. The third style is the mixed style, where the trunk of the workflow can be process-oriented, with dependencies generated by outcome-oriented steps (a step depends on what another step provides), or outcome-oriented, with dependencies generated by sub-workflows consisting of multiple process-oriented steps (a step depends on another workflow).

3.2 Cross-platform execution

SoS uses a YAML-based configuration format to specify properties of all remote computing environments (hosts), and submit tasks in the same workflow to one or multiple isolated hosts, such as high-performance computing clusters running various task queue systems (PBS / MOAB / LFS / Slurm), standalone workstations owned in a laboratory, and virtual machine instances hosted by cloud services. When a task is executed by different remote hosts, SoS translates paths of the task into paths on remote hosts and optionally synchronizes input as output files as tasks are executed. SoS provides built-in support for docker containers, which isolate applications and their supporting libraries and tools so that the entire tool chain and related resources are encapsulated as a bundle and can be readily executed on different platforms. By simply specifying the docker image to use as a parameter for the script to be executed, SoS downloads the specified docker image, mounts appropriate directories, and executes the script inside the docker container. This makes SoS workflow becomes less dependent on the runtime environment and therefore enhances reproducibility.

ACKNOWLEDGMENTS

We appreciate feedback and support from members of the Department of Bioinformatics and Computational Biology at The University of Texas MD Anderson Cancer Center, especially Chris Wakefield and James Melott from the programming team. We thank Dr. Matthew Stephens at The University of Chicago for helpful discussions and support. Some tests of SoS were partially performed using high-performance computing clusters at MD Anderson and The University of Chicago Research Computing Center.

REFERENCES

