

1 Abstract

Explain briefly the paper and what it does.

2 Introduction

Scientific Workflow Management Systems (SWMSs) are an essential tool for automating, managing, and executing complex scientific processes involving large volumes of data and computational tasks¹. Traditional SWMSs employ a linear sequential approach, in which tasks are performed in a pre-defined order, as defined by the workflow. While this linear method is suitable for certain applications, it might not always be the best choice: processing sequentially can prove inefficient in cases where the next step of the process should adapt to the previous one. For these use-cases a dynamic scheduler is required, of which *Managing Event Oriented Workflows*[2] (MEOW) is one.

More about DAGs' inability to adapt

MEOW employs an event-based scheduler, in which jobs are performed non-linearly, triggered based on events². By dynamically adapting the execution order based on the outcomes of previous tasks or external factors, MEOW provides a more efficient and flexible solution for processing large volumes of experimental data³.

- What work am I doing on MEOW?
- How did it go?
- **Write this last**

2.1 Problem

In its current implementation, MEOW is able to trigger jobs based on changes to monitored local files. This covers a the range of scenarios where the data processing workflow involves the creation, modification, or removal of files. By monitoring file events, MEOW's event-based scheduler can dynamically execute tasks as soon as the required conditions are met, ensuring efficient and timely processing of the data. Since the file monitor is triggered by changes to local files, MEOW is limited to local workflows.

While file events work well as a trigger on their own, there are several scenarios where a different trigger would be preferred or even required, especially when dealing with distributed systems or remote operations. To address these shortcomings and

¹citation?

²citation?

³citation?

further enhance MEOW’s capabilities, the integration of network event triggers would provide significant benefits in several key use-cases.

Introduce the BIDS graph. Draw the BIDS graph in tikz

Firstly, network event triggers would allow for manual triggering of jobs remotely, without the need for direct access to the monitored files. This is particularly useful in scenarios where human intervention or decision-making is required before proceeding with the subsequent steps in a workflow. While it is possible to manually trigger job using file events by making changes to the monitored directories, this might lead to an already running job accessing the files at the same time, which could cause problems with data integrity.

Secondly, incorporating network event triggers would facilitate seamless communication between parallel jobs, ensuring that tasks can efficiently exchange information and synchronize their progress.

Finally, extending MEOW’s event-based scheduler to support network event triggers would enable the simple and efficient exchange of data between workflows running on different machines. This feature is particularly valuable in distributed computing environments, where data processing tasks are often split across multiple systems to maximize resource utilization and minimize latency. By leveraging network event triggers, MEOW would be better equipped to manage complex workflows in these environments, ensuring seamless integration and streamlined data processing

2.2 Background

2.2.1 The structure of MEOW

The MEOW event-based scheduler has three main parts: *monitors*, *handlers*, and *the conductor*.

Draw a diagram of the structure.

Monitors monitor for triggering events. They are initialized with a number of *patterns*, which describe the triggering event. When a pattern’s triggering event occurs, the monitor signals to the conductor that the pattern has been triggered.

Handlers perform actions and jobs on behalf of the scheduler. They are initialized with a number of *recipes*, which describe the action to be taken. The handler starts a job when signal to do so by the conductor.

The conductor handles the jobs queue. It is initialized with a number of rules, which a pattern paired with a recipe. When a monitor sends it a triggered pattern, the rules are checked for that pattern. If one or more rules contain that pattern, the corresponding recipes are triggered in their handler.

2.2.2 The `meow_base` codebase

Specific (but not too granular) implementation details of `meow_base`.

The current implementation of MEOW, `meow_base`[3], ...

2.2.3 The `socket` library

The `socket` library[1], included in the Python Standard Library, serves as an interface for the Berkeley sockets API. The Berkeley sockets API, originally developed for the Unix operating system, has become the standard for network communication across multiple platforms. It allows programs to create 'sockets', which are endpoints in a network communication path, for the purpose of sending and receiving data.

- What am I using it for?
- Which parts am I using?
- Other options?
 - `ssl`
 - various API and web frameworks

3 Method

Explain the code I wrote and why I made those choices.

- Expanding on existing code, reusing boiler-plate code
- Test-driven development
- Experiments with triggering on packet
 - Removes the ability to send arbitrary data

4 Results

Does it work? How well?

4.1 Discussion

With the hindsight of the results, what could I have done better?

4.2 Future Work

What should someone do if they want to fix my mistakes, or expand on them further.

5 Conclusion

Did I succeed in what I wanted to do?

References

- [1] Python documentation. *socket - Low-level networking interface*. <https://docs.python.org/3/library/socket.html>.
- [2] David Marchant. “MEOW - Enabling Dynamic Scheduling of Scientific Analysis”. PhD thesis. University of Copenhagen, May 2021.
- [3] David Marchant. *meow_base*. https://github.com/PatchOfScotland/meow_base. 2023.