Monitoring Parsl Workflows

Connor Pigg*, Anna Woodard°, Yadu Babuji°, Kyle Chard°, and Daniel S. Katz*

*University of Illinois at Urbana-Champaign, °University of Chicagp

ABSTRACT

As a workflow software, Parsl provides users the ability to define complex workflows in simple Python to be executed in parallel on any computer system. One useful feature that Parsl lacked was convenient workflow monitoring. This project set out to add workflow monitoring to Parsl as to enhance its feature set. Simple and comprehensive monitoring of a workflows state and resource usage provides users value by allowing auditing, debugging, and confirmation of workflow execution. The work discusses the components of workflows that are monitored by Parsl, the strategy for capturing these components, and the tools used to accumulate and display these components. Primarily, this project used python libraries to collect task status and resource usage and log these tasks to an Elasticsearch database - a flexible and searchable indexing database. A Kibana dashboard - a tool for visualizing data stored in an Elasticsearch database - was then created to visualize the collected logs in a realtime and interactive user interface (UI). In the end, the newer versions of Parsl will allow users the option to monitor the status and resource usage of their workflows via an Elasticsearch database and Kibana dashboard.

INTRODUCTION

Modern scientific practices increasingly incorporate computational elements into their processes. For computational science, these elements almost entirely comprise any given process. The consistent increase in the significance and abundance of computational elements drives the development of computational tools.

One such tool for the modern scientist is workflow software. This software allows the definition and execution of a series of computational elements.

Benefits of workflow software:

- **Automating their task execution**
- **Documenting steps taken in their process**
- ***Plus additional features!***

METHOD

The method for monitoring implemented by this work is to capture information, collect information, and then visualize the information. For capturing workflow status, information about the task state is aggregated and then logged to an **Elasticsearch** database at state transitions. To monitor a task's resource usage, a process is spawned when a task is started which periodically checks the resource usage of its parent process - the task's python process. This information is also logged to the Elasticsearch database. Once this information is centralized in the database, it is visualized by a connected **Kibana** instance. These visualizations are created using Kibana's tools to aggregate and filter the data before presenting it to the user in a dashboard. To allow users to set up individual instances, Parsl provides saved Kibana objects and a script that will create the necessary Elasticsearch templates.

DASHBOARD

Parsl features:

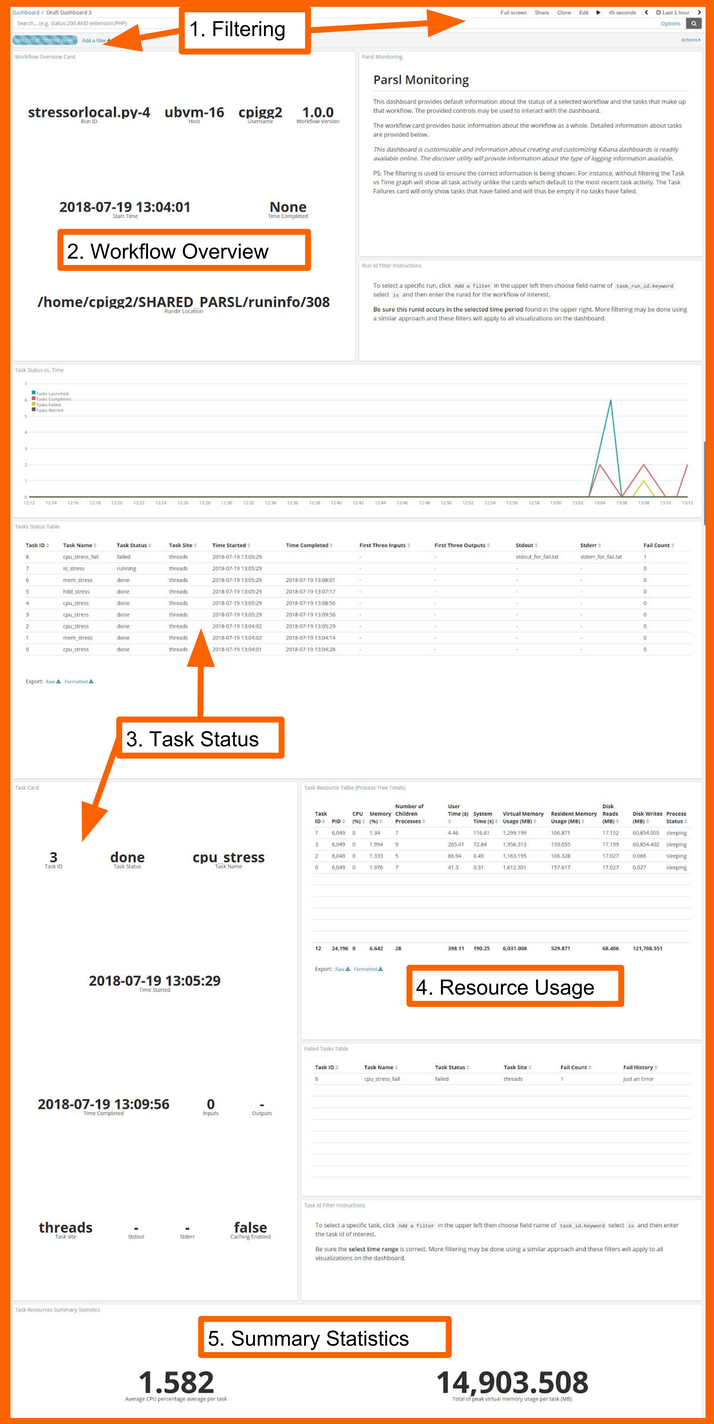
- **Implicit parallelization of task execution**
- **Intermediate result caching**
- **Resource agnostic**
- **Simple workflow definition**
- ***Workflow Monitoring***

Significance:

- **Debugging erroneous workflows**
- **Auditing the execution and resource usage of tasks**
- **Confirm the status of running workflows**

Dashboard Features:

1. Workflow, time, and task filtering
2. Workflow overview
3. Task status
4. Resource usage
5. Summary statistics
6. Live monitoring



EXAMPLE

```
import parsl
from parsl import *

threads_config = parsl.config.Config(
                executors=[parsl.executors.threads.ThreadPoolExecutor( label='threads',
            max_threads=4)], db_logger_config={'enable_es_logging': True, 'enable_remote_monitoring':
            True})

dfk = DataFlowKernel(config=threads_config)

@App('bash', dfk)
def cpu_stress_fail(workers=1, timeout=10, inputs=[], stdout='stdout_for_fail.txt',
stderr='stderr_for_fail.txt'):
    raise AssertionError("Just an Error")
    return "stress --cpu {workers} --timeout {timeout}"

@App('bash', dfk)
def cpu_stress(workers=1, timeout=10, inputs=[], outputs=['out.txt']):
    return "stress --cpu {} --timeout {} > {}".format(timeout, workers, outputs[0])

@App('bash', dfk)
def mem_stress(workers=1, mem_size=1, timeout=10, inputs=[]):
    return "stress --vm {workers} --vm-bytes {mem_size}G --vm-keep --timeout {timeout}"

@App('bash', dfk)
def io_stress(workers=1, timeout=10, inputs=[]):
    return "stress --io {workers} --timeout {timeout}"

@App('bash', dfk)
def hdd_stress(workers=1, hdd_size=1, timeout=10, inputs=[]):
    return "stress -d {workers} --hdd-bytes {hdd_size}G --timeout {timeout}".format(workers=workers,
hdd_size=hdd_size, timeout=timeout)

if __name__ == "__main__":
    MAXTIMEOUT=10
    from random import random
    a1,b1,c1 = [cpu_stress(workers=1, timeout=random() * MAXTIMEOUT, outputs=['out.txt']),
                mem_stress(workers=1, timeout=random() * MAXTIMEOUT, mem_size=1),
                cpu_stress(workers=1, timeout=random() * MAXTIMEOUT)]
    a1.result()
    b1.result()
    c1.result()

    dfk.cleanup()
```

CONCLUSIONS

Monitoring has been added to Parsl. This monitoring collects and presents actionable informations to users. The information is generated from arbitrary and distributed sources. It is then collected to a central space before being visualized. This visual dashboard is made available to users and is independent of both user and data source allowing great flexibility. This effective and simple implementation relied heavily on the chosen tools.

REFERENCE

Yadu Babuji, Alison Brizius, Kyle Chard, Ian Foster, Daniel S. Katz, Michael Wilde, and Justin Wozniak. Introducing Parsl: A Python Parallel Scripting Library, August 2017.