ISBN: 978-3-95450-193-9 doi:10.18429/JACoW-ICALEPCS2017-THPHA043

LIGHTFLOW - A LIGHTWEIGHT, DISTRIBUTED WORKFLOW SYSTEM

A. Moll, R. Clarken, P. Martin, S. Mudie, Australian Synchrotron - ANSTO, Melbourne, Australia

Abstract

The Australian Synchrotron, located in Clayton, Melbourne, is one of Australia's most important pieces of research infrastructure. After more than 10 years of operation, the beamlines at the Australian Synchrotron are well established and the demand for automation of research tasks is growing. Such tasks routinely involve the reduction of TB-scale data, online (realtime) analysis of the recorded data to guide experiments, and fully automated data management workflows.

In order to meet these demands, a generic, distributed workflow system was developed. It is based on well-established Python libraries and tools. The individual tasks of a workflow are arranged in a directed acyclic graph and one or more directed acyclic graphs form a workflow. Workers consume the tasks, allowing the processing of a workflow to scale horizontally. Data can flow between tasks and a variety of specialised tasks is available.

Lightflow has been released as open source on the Australian Synchrotron GitHub page [1].

INTRODUCTION

With the advent of sample changing robots and automated analysis tools, the beamlines at the Australian Synchrotron require the automation of data processing and analysis tasks. Looking into the individual automated workflows for the beamlines it is found that the implementation of the workflows share a very similar set of requirements. This allows the deployment of a common workflow system across all beamlines. An example for an automated workflow is shown in Fig. 1. A detector captures the data produced in an experiment and writes the data to files on a high speed storage. Upon the arrival of new files a pipeline is started which reads, processes, and analyses the new files. Often the result of processing the new files has to be merged with previous runs of the pipeline, recorded in a central storage. At the end, the result of the pipeline is displayed to the user.

A system that supports such a workflow has to offer the capability to start one or more pipelines based on external events, such as the appearance of new files or the change of EPICS Process Variables; model complex pipelines and allow their execution on a distributed computing system; and offer a central storage for keeping intermediate results. Lightflow, the workflow system presented here, fulfills those requirements.

ARCHITECTURE

Lightflow models a workflow as a set of individual tasks arranged as a directed acyclic graph (DAG). This specification encodes the direction that data flows as well as dependencies between tasks. Each workflow consists of one or more

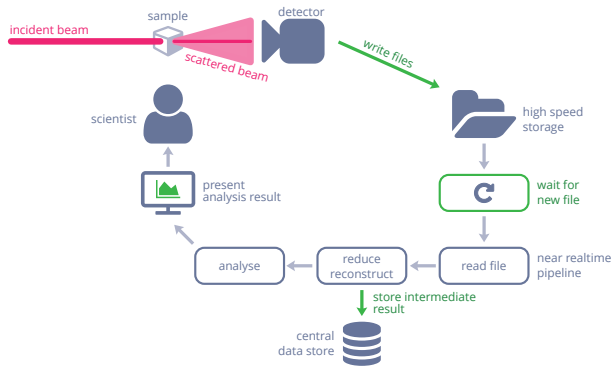

Figure 1: Near realtime pipeline at a beamline. Data captured by the detector is processed in near realtime in order to provide users with quick feedback.

DAGs. While the arrangement of tasks within a DAG cannot be changed at runtime, other DAGs can be triggered from within a task, therefore enabling a workflow to be adapted to varying inputs or changing conditions during runtime.

Lightflow employs a worker-based queuing system, in which workers consume individual tasks. This allows the processing of workflows to be distributed. Such a scheme has multiple benefits: It is easy to scale horizontally; tasks that can be executed in parallel are executed on available workers at the same time; tasks that require specialised hardware or software environments can be routed to dedicated workers; and it simplifies the integration into existing container based cloud environments.

In order to avoid single points of failure, such as a central daemon often found in other workflow tools, the queuing system is also used to manage and monitor workflows and DAGs. When a new workflow is started, it is placed in a special queue and is eventually consumed by a worker. A workflow is executed by sending its DAGs to their respective queues. Each DAG will then start and monitor the execution of its tasks. The diagram in Fig. 2 depicts the worker-based architecture of Lightflow.

IMPLEMENTATION

Lightflow is written in Python 3 and supports Python 3.5 and higher. It uses the Celery [2] library for queuing tasks and the NetworkX [3] module for managing the directed acyclic graphs. As redis [4] is a common database found at many beamlines at the Australian Synchrotron, it is the default backend for Celery in Lightflow. However, any other Celery backend can be used as well. In addition to redis, Lightflow uses MongoDB [5] in order to store data that is persistent during a workflow run. Examples include the

ISBN: 978-3-95450-193-9 doi:10.18429/JACoW-ICALEPCS2017-THPHA043

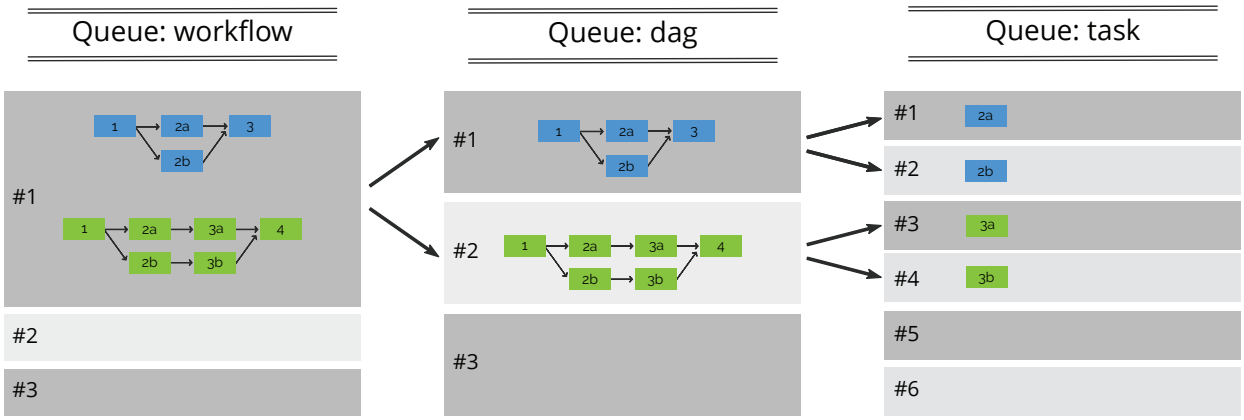

Figure 2: Worker based architecture of Lightflow. Workers are used to manage the three main components of a workflow: workflows, DAGs and tasks. In a typical setup for a computing cluster there are multiple queues for tasks.

aggregation of values, calculation of running averages, or the storage of flags.

Tasks can receive data from upstream tasks and send data to downstream tasks. Any data that can be serialised can be shared between tasks. Typical examples for data flowing from task to task are file paths, pandas [6] DataFrames or numpy [7] arrays. The exchange of data across a distributed system is accomplished by using cloudpickle [8] in order to serialise and deserialise the data. Lightflow provides a fully featured command line interface for starting, stopping and monitoring workflows and workers. The command line interface is based on the click [9] Python module. An API is also available for easy integration of Lightflow with existing tools and software.

In order to keep Lightflow lightweight, the core library focuses on the essential functionality of a distributed workflow system and only implements two tasks, a generic Python task and a bash task for calling arbitrary bash commands. Specialised tasks and functionality is implemented in extensions. Currently there are three extensions to Lightflow available: The filesystem extension offers specialised tasks for watching directories for file changes and tasks covering basic file operations [10]; the EPICS [11] extension offers tasks that hook into EPICS [12], a control system used at the Australian Synchrotron for operating the hardware devices of the accelerator and the beamlines; and the REST extension provides a RESTful interface for starting, stopping and monitoring workflows via HTTP calls [13].

WORKFLOW DEFINITION

Lightflow uses Python and an efficient API to define a workflow. Users don't have to learn a domain specific language and can use their preferred Python libraries. The workflow definition can be tested locally and scales without changes to run on a computing cluster. Lightflow ships with 15 examples, highlighting all features of the workflow system. The following source code shows a simple workflow definition consisting of three tasks, incrementing a number:

```
from lightflow.models import Dag
from lightflow.tasks import PythonTask

# the callback function for all tasks
def inc_number(data, store, signal, context):
    print('Task {task_name} being run in '
          'DAG {dag_name} for workflow '
          '{workflow_name} ({workflow_id}) '
          'on {worker_hostname}'.
          format(**context.to_dict()))

    if 'value' not in data:
        data['value'] = 0

    data['value'] = data['value'] + 1
    print('This is task ↵
                  #{}'.format(data['value']))

# create the main DAG
d = Dag('main_dag')

# create the 3 tasks that increment a number
task_1 = PythonTask(name='task_1',
                    callback=inc_number)

task_2 = PythonTask(name='task_2',
                    callback=inc_number)

task_3 = PythonTask(name='task_3',
                    callback=inc_number)


# set up the graph of the DAG
# as a linear sequence of tasks
d.define({
    task_1: task_2,
    task_2: task_3
})
```

ISBN: 978-3-95450-193-9 doi:10.18429/JACoW-ICALEPCS2017-THPHA043

LIGHTFLOW AT THE AUSTRALIAN SYNCHROTRON

Lightflow at the MX Beamline

The two Crystallography beamlines (MX1, MX2) at the Australian Synchrotron have employed a custom made data management workflow for a number of years. Both the raw and reconstructed data of an experiment is compressed into squashfs files, verified and stored in the central storage system of the Australian Synchrotron. Recently this workflow has been upgraded to use Lightflow in order to take advantage of a distributed system to compress multiple experiments at the same time. The updated setup consists of a management virtual machine that hosts the workflow and DAG queues as well as acting as a REST endpoint for starting the squashfs workflow. Three physical servers act as squashfs nodes. The workflow is triggered by a HTTP REST call from the experiment change management system at the Crystallography beamlines.

Lightflow at the SAXS/WAXS Beamline

Several data processing pipelines are implemented using Lightflow for the SAXS/WAXS beamline. An example is the phaseID pipeline. This pipeline identifies diffraction peak positions within SAXS profiles and infers the most likely Space Group. This pipeline enables researchers to rapidly determine phase diagrams for self-assembled lyotropic liquid crystal systems. These systems are important for drug delivery and controlled release.

CONCLUSION

Lightflow is a lightweight and distributed workflow system written in Python and has been released as open source software on GitHub [1]. It is currently used at several beamlines at the Australian Synchrotron for managing data or implementing data processing pipelines. The next steps are to extend the use of Lightflow at the Australian Synchrotron to the experiment change management at beamlines, complex data management workflows and auto processing workflows at the Crystallography beamlines.

REFERENCES

[1] Lightflow, https://github.com/AustralianSynchrotron/lightflow

[2] Celery, http://www.celeryproject.org

[3] NetworkX, https://networkx.github.io

[4] Redis, https://redis.io

[5] MongoDB, https://www.mongodb.com

[6] Pandas, http://pandas.pydata.org

[7] Numpy, http://www.numpy.org

[8] Cloudpickle, https://github.com/cloudpipe/cloudpickle

[9] Click, http://click.pocoo.org

[10] Lightflow Filesystem, https://github.com/AustralianSynchrotron/lightflow-filesystem

[11] EPICS, http://www.aps.anl.gov/epics

[12] Lightflow EPICS, https://github.com/AustralianSynchrotron/lightflow-epics

[13] Lightflow Rest, https://github.com/AustralianSynchrotron/lightflow-rest