D3.3.4 LSST to 4MOST communication bridge

WP3.3 Spectroscopic classification of transients

Project Acronym	LUSC-B
Project Title	UK Involvement in the Legacy Survey of Space and Time
Document Number	LUSC-B-43

Submission date	27/06/2023
Version	1.0
Status	Final
Author(s) inc. institutional affiliation	Christopher Frohmaier (Southampton) Mark Sullivan (Southampton)
Reviewer(s)	Stephen Smartt (Oxford) Matt Nicholl (QUB)

Dissemination level	
Public	This report may be distributed to any interested parties.

Version History

Version	Date	Comments, Changes, Status	Authors, Contributors, Reviewers
0.1	05/MAY/2023	First draft for review	Chris Frohmaier
0.2	27/JUN/2023	Updated following reviwer comments	Chris Frohmaier
1.0	21/JUL/2023	Exec Group Approved	

Table of Contents

List of Figures

List of Tables

1 Executive Summary

The Legacy Survey of Space and Time (LSST) and the 4m multi-object spectroscopic telescope (4MOST) will commence operations in 2024. The Time-Domain Extragalactic Survey (TiDES) on 4MOST will follow-up LSST discovered transients to obtain spectroscopic measurements for tens-of-thousands of supernovae, galaxies, active-galactic nuclei (AGN), and strongly-lensed systems. For the transients and their hosts, this data will allow us to map the astrophysical diversity of cosmic explosions and measure the equation of state parameter for dark energy to unprecedented precision. TiDES forms the basis of WP3.3: Spectroscopic classification of transients.

In D3.3.3 we produced a software product that interacted with the Lasair broker and ran a selection function on the retrieved light curves. At the time of submission of D3.3.3[1], the Zwicky Transient Facility (ZTF) had been undergoing maintenance for a number of weeks leaving us unable to test the operation of the code on a real-time data stream. Instead, we used an archieve of transients and simulated a dataflow. In this deliverable, D3.3.4, we implement the real-time Kafka stream capabilities from Lasair and present our communication bridge to submit LSST discovered transients into the 4MOST system.

Our communication bridge is the link between *any* transient broker and 4MOST - although for our purposes, we have developed it around the UK's Lasair broker. Our software periodically checks our TiDES Kafka stream on Lasair for new transients. The software ingests the transient light curve and applies the D3.3.3 software products. The objects that meet this selection criteria are stored in a PostgreSQL database that stages the transients before they are submitted to 4MOST. We then use the 4MOST API (currently in proprietary development stages within 4MOST) to submit transients to the 4MOST observing queue. The 4MOST API returns the status of the submitted transient and we update our system to keep in sync with the 4MOST database. The software pipeline is orchestrated using Prefect[1] and deployed on the Somerville system. Given that LSST is still a number of years away, the development and testing has used the ZTF data but with minimal tweaks we can adapt our software to work with LSST alerts.

[1] https://www.prefect.io/

2 Introduction

The software presented in this deliverable will triage transients issued through a broker stream and divert the desired events to 4MOST for spectroscopic follow-up. Work Package 3.3 focuses on the spectroscopic classification of transient events from LSST using the 4MOST/TiDES facility. The majority of extra-galactic transients discovered by LSST will be supernovae (SNe), with Type Ia SNe playing the critical role in our cosmology analysis. 4MOST/TiDES will survey the entire southern sky, collecting spectra of transients and their hosts at an industrial scale, unrivalled in volume by an contemporary facility. TiDES will not, however, dictate the overall strategy or any individual pointing of 4MOST during its operations, nor will TiDES know precisely where 4MOST will observe on any given night. This means TiDES will need to be ready with a target list to accommodate 4MOST observations where ever and whenever they occur. It is, therefore, paramount that TiDES have the software capabilities to rapidly identify targets from the LSST real-time stream and pass these objects on for spectroscopic follow-up.

LSST is still under-construction, but the Zwicky Transient Facility (ZTF) and the Lasair broker act as our development services with the intent to scale-up operations for LSST. In this document we present our software that links the ZTF(LSST) data streams with the TiDES selection functions, and the passes objects on to the 4MOST observing queue. Our software has been deployed and runs on the Somerville HPC system. This software is not intended to run locally on a users machine (although it will if desired). Finally, the software uses the 4MOST Transient API, this is a RESTful API with a python wrapper that is still being refined with the 4MOST collaboration and subject to changes. It is not included in this deliverable, but the packages are imported into our code. Within the TiDES area of Somerville a user can find all the necessary python libraries to execute the software. All the code presented in this document are available from the following GitHub repository: https://github.com/lsst-uk/tidesInterface-WP3.3/tree/main/tidesCommunicate

2.1 Glossary of Acronyms

4MOST	4m **M**ulti-**O**bject **S**pectroscopic **T**elescope
DESC	**D**ark **E**nergy**S**cience **C**ollaboration
LSST	**L**egacy **S**urvey of **S**pace and **T**ime
OB	**O**bserving **B**lock
TiDES	**Ti**me-**D**omain **E**xtragalactic **S**urvey
ZTF	**Z**wicky **T**ransient **F**acility

3 Requirements

This code is written predominantly in the `Python` language (v3.10.9 for development), the database management scripts are written in `SQL` and managed in `Postgres` v15.1. Standard, prepackaged Python libraries are used, but the following libraries must be installed:

```
    lasair==0.0.5
    prefect==2.8.3
    prefect-dask==0.2.3
    json5
    PyYAML==6.0
    pandas==1.5.2
    SQLAlchemy==1.4.46
    numpy
```

Furthermore, the 4MOST Transient API is used, but this is currently private to authorised 4MOST users. This appears as `import submit_transients as st` in the main body code.

3.1 Prefect Orchestration

3.1.1 flowSetting.yaml

There are several settings in the pipeline that can be customised via a YAML file. An example of the contents of the parameter file is shown below. The first three settings are all related to the communication with Lasair[2]. The `topic` setting holds the Lasair filter you wish to pull your targets from, the `groupID` specifics your 'bookmark' in the Kafka stream – using a consistent number means your next call to Lasair picks up from where you left off. The `lasairToken` parameter is your API key to access Lasair services. The `selectFunctionPath` and `selectFunction` keys represent the selection functions described in D3.3.3[1]. Finally, the pipeline requires a PostgreSQL database installed to stage the transients and to sync with the 4MOST OB database, the access credentials are provided in the final three parameters.

```
devConfig:
    topic: lasair_19tidesSelect
    groupID: 123456
    lasairToken: Cb6442e76373g2M7bwjd738ae1c946d54b8f7993
    selectFunctionPath: ./path/to/tidesSelectionFunctions.yml
    selectFunction: tidesSNZTFSelectDEMO1
    tidesDBUser: username
    tidesDBpass: tidesDBpassword
    tidesDBdatabase: tides
```

4 The Pipeline

The pipeline in this deliverable is shown in the flowchart schematic in Figure 1. The code is hosted on the Somerville compute resource on a TiDES virtual machine (VM). At this stage of development, the VM is relatively modest - 4 CPU cores, 16GB RAM, and access to 15TB of storage shared between all TiDES projects. Our development environment currently listens to the ZTF data alerts as brokered by Lasair, the resources we've allocated the VM is sufficient to process this stream. As we will see in Section 4.2.1, the batch processing of transient light curves is performed by the Dask Client, allowing for resource to be scaled to meet the requirements of an LSST data stream. We would like to note that in D3.3.3 our selection function code ran on a single core and processed transients in series. We stated that with a small amount of additional

[2] https://lasair.readthedocs.io/en/main/index.html

effort we would be able to re-write a section of the code to be parallelisable, in this deliverable we have achieved that goal.

In the following Sections we go through the key stages of the pipeline's execution. The pipeline is orchestrated using the Prefect system of `@flow` and `@task` decorators in the Python code. A Flow is a container for workflow logic, can handle data input, execute work, and output results, they operate just like a function in Python. Tasks are small discrete units of work – again acting just like Python functions – that are able to receive metadata about upstream dependencies before they are executed, allowing for tasks to trigger from other task dependencies when needed. Strictly speaking, this workflow logic can be executed using just `@flow` logic, although in this pipeline we use both.

4.1 `getlatestBatch()`

This function interacts with the Lasair Kafka stream for the topic described in Section 3.1.1. It receives a new transient for each call to the stream, if the consumer doesn't get a response for 5 seconds it assumes all recent transients have been received, or if the queue is empty then our pipeline is up to date. In the event of the latter, the pipeline terminates with a successful exit status. Data from Lasair are received in the JSON format, and once completed these packets are converted into a Pandas DataFrame and parsed through the rest of the pipeline.

4.2 Light curve analysis

The Lasair API can, at most, retrieve 50 light curve from a single API call. Furthermore, there is limit on the number of API calls per day, we therefore make more efficient use of our API allocation by splitting our transients up into arrays of 50 transients per API call in the function `splitIntoChunks()`, this then feeds into the `chunkyAssign()` flow decorated function associated with a Dask Client. At this point, we have light curves for all our transients held in local memory.

4.2.1 `chunkyAssign()`

This function interacts with the a DASK client. All the transients light curves are submitted into the task pool. The Dask client will use all allocated resources to process the light curves in parallel using the `lightCurveSatisfy()` task. This step performs the D3.3.3[1] products to compare any transient light curve to a user defined selection function. Each transient is then assigned a `True` or `False` flag based on whether or not is satisfied the selection criteria.

4.3 Transient Databases

At this point in the pipeline, we move from Python-centric code to the execution of SQL scripts to manage the data. From Section 4.2.1, all transients that receive a `True` flag are automatically sent to a temporary PostgreSQL table called `tides_stage`. As the name suggests, this is the staging area for these transients in preparation to be sent to 4MOST. A sample of the `tides_stage` table is shown in Figure 2, where it can be seen that the ZTF alert packet and Lasair value-added data are included as columns in this table. As this is only a temporary table, it exists for the duration of the session and is automatically wiped when the database connection closed at the end of the pipeline.

All data that has ever satisfied a selection function is stored in the `tides_master` table, where each object receives a unique Primary Key identifier and timestamps denoting the object's creation and the date the object was last manipulated. There is also a column for the object's 4MOST Primary Key, which we will discuss in Section 4.4.

The `upsertToMaster()` function is executed on the `tides_stage` and `tides_master` tables. In SQL, the `UPSERT` statement will either `UPDATE` a table if a key match is made or `INSERT` a new row if there is no match. Updates include information in the latest observations such as magnitude, observation dates, or Sherlock classifications. For our context, we UPSERT data from `tides_stage` into `tides_master`.

Our communication bridge to 4MOST is not just for sending new targets. We also need to do some general housekeeping to remove objects that either haven't been observed by LSST in a user defined number of days, or that have faded below the magnitude limit of 4MOST. This is performed via the `deactivateUnobservedTransients()` task and simply changes the `active` flag in the `tides_master` table.

The objects identified by our scripts as new transients, or transients just updated, or transients needing to be deactivated are all pulled from their relevant tables and held in a Pandas Dataframe we refer to as `4MOST Stage` in Figure 1.

4.4 Communicating with 4MOST

We now have all the information necessary to interact with the 4MOST database to include our transients in the observing plans. To achieve this, we use the 4MOST Transient RESTful API which provides a full CRUD service (Create, Retrieve, Update, Delete). The service is still under development by 4MOST, although the way in which we interact with the API will remain fixed, the exact columns in the database are still subject to change too. There is a beta version for a python wrapper that we use in this deliverable, but it remains private to 4MOST membership. Finally, an API key is needed and only made available to the 4MOST members who request it.

Two python functions run on our `4MOST Stage` table. The first, `createNewTransientin4MOST()`, sends the new transients to the 4MOST PostgreSQL database and for each new entry the primary key for the new event is returned. The second function, `updateExisitingTransient()`, updates 4MOST with the latest information on the transient's current status and, if necessary, deactivates the event's flag in their database. Deactivated events remain in the 4MOST database, but will not be selected by the 4MOST target scheduler for observation.

The pipeline then performs its final task by updating the `tides_master` table with all the new primary keys from 4MOST so that both the 4MOST and TiDES databases are sync-ed with the same information.

5 Deployment

As mentioned throughout, the pipeline is orchestrated by Prefect. Prefect Flows can be scheduled to run based on time intervals (similar to Cron jobs), hooks (i.e. triggered by external events), or at the users discretion either through just running the python script or a web interface. The Prefect Agent is a continuously running process in the background on the TiDES VM that looks for work via one of the methods mentioned above. A screenshot of our agent running on Somerville and an example of our scheduler are shown in Figure 3.

When the code is triggered, a real-time flowchart is visualised in the Prefect web portal to document the pipeline's execution progress. A screenshot of a recently completed flow can be found in Figure 4, although a more useful animation of how the flow progresses and how it can be interacted with is shown in the GitHub repo[3]. This portal can either be hosted locally or deployed into the cloud. For development purposes the cloud version was used due to the ease of

[3] https://github.com/lsst-uk/tidesInterface-WP3.3/tree/main/tidesCommunicate

setting this up. However, there are several drawbacks to a free-tier cloud account, including only 7 days of execution history recorded and only 1 user able to manage the deployment. During Phase C, we will switch to a Somerville-hosted Prefect Agent and Deployment so that anyone in TiDES can monitor the pipeline and it's history recorded for the full 5-years of operations.

Our pipeline has been running on Somervile for a couple of weeks and performing as expected. The only problem we have encountered is when the connection drops to the Prefect Agent it must be manually restarted. This happens about every 24 hours and we currently believe it is a Somerville issue assuming our virtual machine is inactive as no users are logged in. It should be noted that this doesn't happen when the agent was hosted on a desktop during development and left for a longer period – of course, this is not a solution for the LSST-era. We are currently investigating solutions.
Update 27 Jun 2023: This bug has been fixed and was traced to an `httpcore` library. This can be resolved via the environment variable `PREFECT_API_ENABLE_HTTP2=False` or by upgrading to the latest `httpcore`.

However, the issue of the Somerville system's stability still remains a risk as highlighted in the reviewer comments. Our deployment ran for a couple of weeks without issue, however a maintenance upgrade of Somerville killed our Prefect Agent. It was a number of days before I noticed and had to manually reboot. This would not be good news in the LSST era and the data backlog would build up very quickly. This should be added to our Risk Management.

6 References

References

[1] LSST/TiDES Metrics Software, Project Deliverable D3.3.3

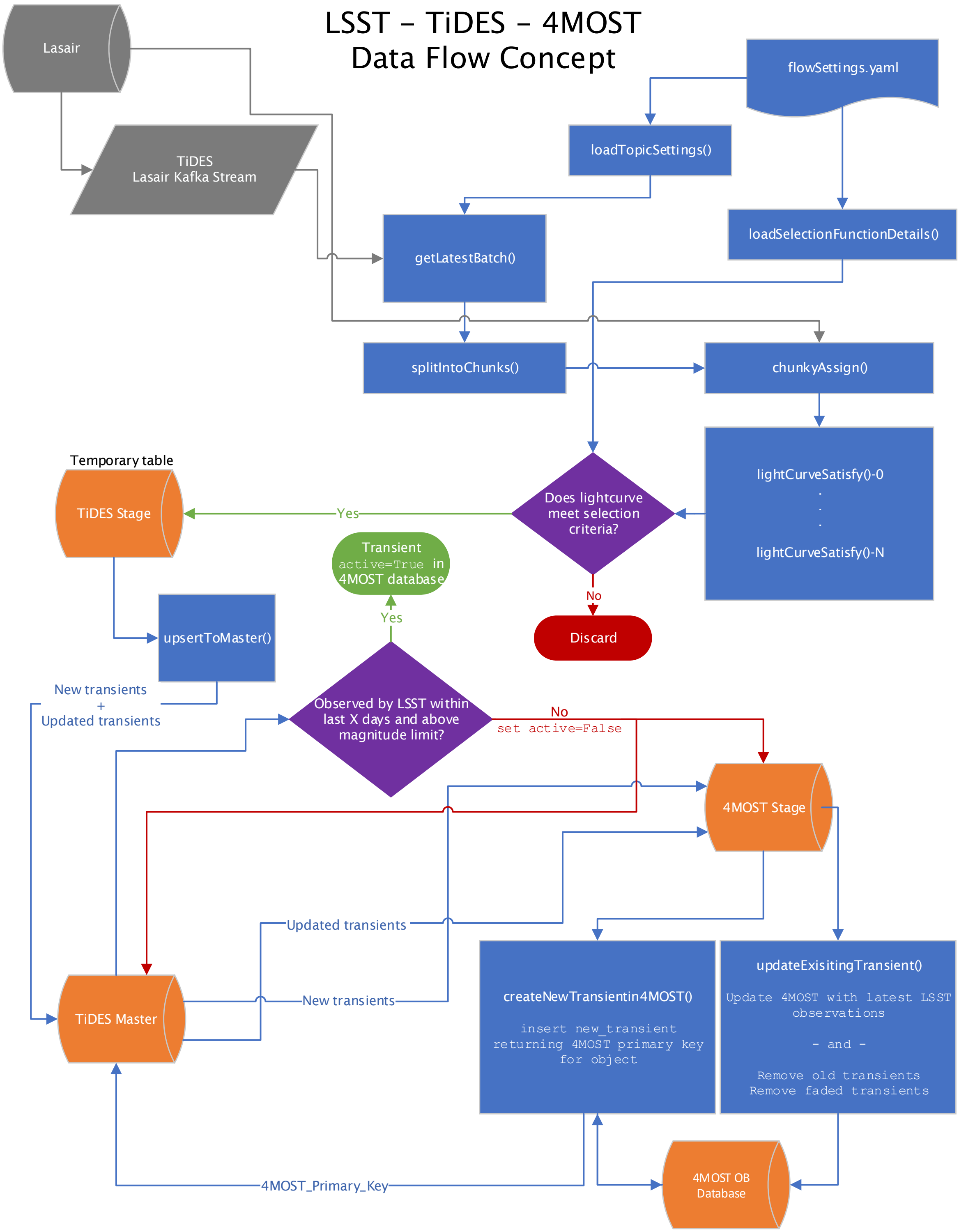
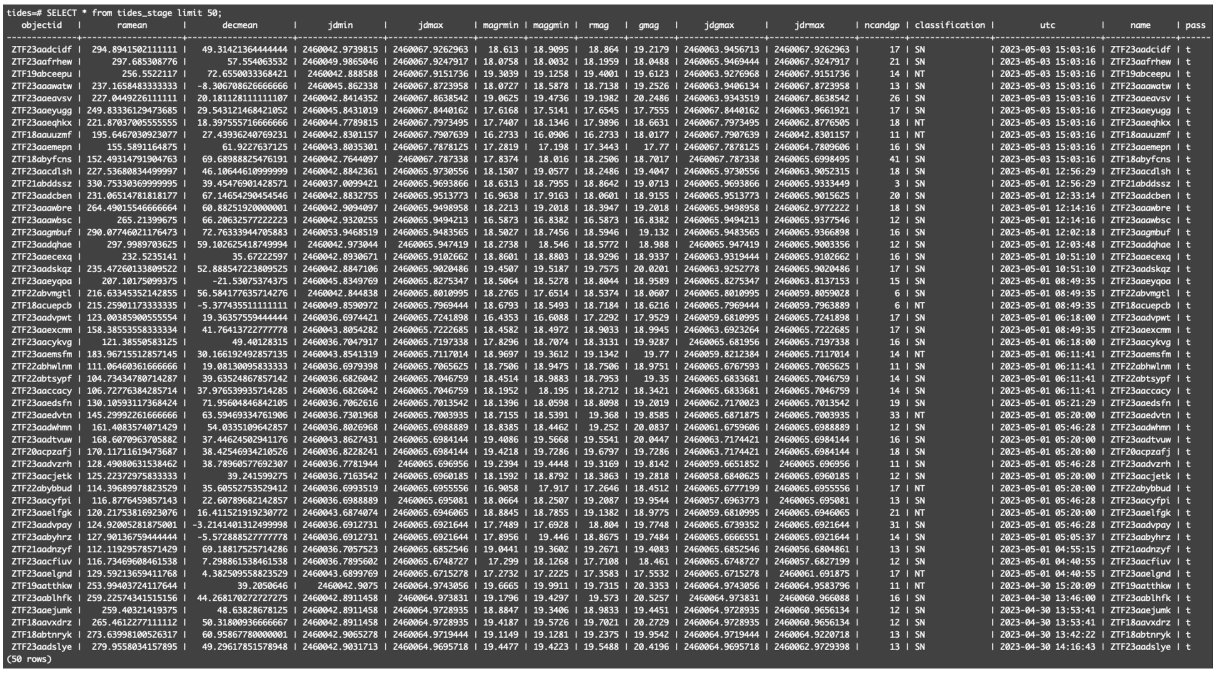


Figure 1: The flowchart describing the logic of D3.3.4. External Lasair products are shown in grey, SQL databases are shown in orange, python functions and SQL scripts are in blue, with selection decisions in purple.

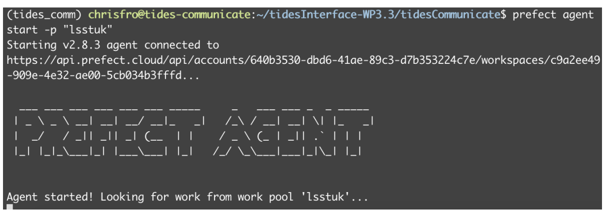


tides=# SELECT * from tides_stage limit 50;

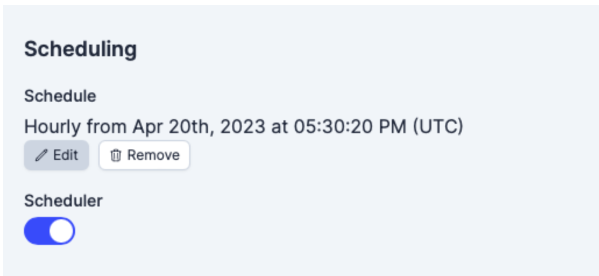
objectid	ramean	decmean	jdmin	jdmax	magrmin	maggmin	rmag	gmag	jdgmax	jdrmax	ncandgp	classification	utc	name	pass
ZTF23aadcidf	294.8941502111111	49.31421364444444	2460042.9739815	2460067.9262963	18.613	18.9095	18.864	19.2179	2460063.9456713	2460067.9262963	17	SN	2023-05-03 15:03:16	ZTF23aadcidf	t
ZTF23aafrhew	297.685308776	57.554063532	2460049.9865046	2460067.9247917	18.0758	18.0032	18.1959	18.0488	2460065.9469444	2460067.9247917	21	SN	2023-05-03 15:03:16	ZTF23aafrhew	t
ZTF19abceepu	256.5522117	72.6550033368421	2460042.888588	2460067.9151736	19.3039	19.1258	19.4001	19.6123	2460063.9276968	2460067.9151736	14	NT	2023-05-03 15:03:16	ZTF19abceepu	t
ZTF23aaawatw	237.1658483333333	-8.306708626666666	2460045.862338	2460067.8723958	18.0727	18.5878	18.7138	19.2526	2460063.9406134	2460067.8723958	13	SN	2023-05-03 15:03:16	ZTF23aaawatw	t
ZTF23aaeavsv	227.0449226111111	20.181128111111107	2460042.8414352	2460067.8638542	19.0625	19.4736	19.1982	20.2486	2460063.9343519	2460067.8638542	26	SN	2023-05-03 15:03:16	ZTF23aaeavsv	t
ZTF23aaeyugg	249.83336129473685	29.543121468421052	2460045.8431019	2460067.8440162	17.6168	17.5141	17.6545	17.7555	2460067.8440162	2460063.9661921	17	SN	2023-05-03 15:03:16	ZTF23aaeyugg	t
ZTF23aaeqhkx	221.87037005555555	18.397555716666666	2460044.7789815	2460067.7973495	17.7407	18.1346	17.9896	18.6631	2460067.7973495	2460062.8776505	18	NT	2023-05-03 15:03:16	ZTF23aaeqhkx	t
ZTF18aauuzmf	195.6467030923077	27.43936240769231	2460042.8301157	2460067.7907639	16.2733	16.0906	16.2733	18.0177	2460067.7907639	2460042.8301157	11	NT	2023-05-03 15:03:16	ZTF18aauuzmf	t
ZTF23aaemepn	155.5891164875	61.9227637125	2460043.8035301	2460067.7878125	17.2819	17.198	17.3443	17.77	2460067.7878125	2460064.7809606	16	SN	2023-05-03 15:03:16	ZTF23aaemepn	t
ZTF18abyfcns	152.49314791904763	69.68988825476191	2460042.7644097	2460067.787338	17.8374	18.016	18.2506	18.7017	2460067.787338	2460065.6998495	41	SN	2023-05-03 15:03:16	ZTF18abyfcns	t
ZTF23aacdlsh	227.53680834499997	46.10644610999999	2460042.8842361	2460065.9730556	18.1507	19.0577	18.2486	19.4047	2460065.9730556	2460063.9052315	18	SN	2023-05-01 12:56:29	ZTF23aacdlsh	t
ZTF21abddssz	330.75330369999995	39.45476901428571	2460037.0099421	2460065.9693866	18.6313	18.7955	18.8642	19.0713	2460065.9693866	2460065.9333449	3	SN	2023-05-01 12:56:29	ZTF21abddssz	t
ZTF23aadcben	231.06514781818177	67.14654290454546	2460042.8832755	2460065.9513773	16.9638	17.9163	18.0601	18.9155	2460065.9513773	2460065.9015625	20	SN	2023-05-01 12:33:14	ZTF23aadcben	t
ZTF23aaawbre	264.49015546666664	60.88251920000001	2460042.9094097	2460065.9498958	18.2213	19.2018	18.3947	19.2018	2460065.9498958	2460062.9772222	18	SN	2023-05-01 12:14:16	ZTF23aaawbre	t
ZTF23aaawbsc	265.21399675	66.20632577222223	2460042.9320255	2460065.9494213	16.5873	16.8382	16.5873	16.8382	2460065.9494213	2460065.9377546	12	SN	2023-05-01 12:14:16	ZTF23aaawbsc	t
ZTF23aagmbuf	290.07746021176473	72.76333944705883	2460053.9468519	2460065.9483565	18.5027	18.7456	18.5946	19.132	2460065.9483565	2460065.9366898	16	SN	2023-05-01 12:02:18	ZTF23aagmbuf	t
ZTF23aadqhae	297.9989703625	59.102625418749994	2460042.973044	2460065.947419	18.2738	18.546	18.5772	18.988	2460065.947419	2460065.9003356	12	SN	2023-05-01 12:03:48	ZTF23aadqhae	t
ZTF23aaecexq	232.5235141	35.67222597	2460042.8930671	2460065.9102662	18.8601	18.8803	18.9296	18.9337	2460063.9319444	2460065.9102662	16	SN	2023-05-01 10:51:10	ZTF23aaecexq	t
ZTF23aadskqz	235.47260133809522	52.888547223809525	2460042.8847106	2460065.9020486	19.4507	19.5187	19.7575	20.0201	2460063.9252778	2460065.9020486	17	SN	2023-05-01 10:51:10	ZTF23aadskqz	t
ZTF23aaeyqoa	207.10175099375	-21.53075374375	2460045.8349769	2460065.8275347	18.5064	18.5278	18.8044	18.9589	2460065.8275347	2460063.8137153	15	SN	2023-05-01 08:49:35	ZTF23aaeyqoa	t
ZTF22abvmgtl	216.63345352142855	56.584177635714276	2460042.844838	2460065.8010995	18.2765	17.6514	18.5374	18.0607	2460065.8010995	2460059.8059028	6	SN	2023-05-01 08:49:35	ZTF22abvmgtl	t
ZTF18acuepcb	215.25901173333335	-5.377435511111111	2460049.8590972	2460065.7969444	18.6793	18.5493	18.7184	18.6216	2460065.7969444	2460059.7963889	6	NT	2023-05-01 08:49:35	ZTF18acuepcb	t
ZTF23aadvpwt	123.00385900555554	19.36357559444444	2460036.6974421	2460065.7241898	16.4353	16.6088	17.2292	17.9529	2460059.6810995	2460065.7241898	17	SN	2023-05-01 06:18:00	ZTF23aadvpwt	t
ZTF23aaexcmm	158.38553558333334	41.76413722777778	2460043.8054282	2460065.7222685	18.4582	18.4972	18.9033	18.9945	2460063.6923264	2460065.7222685	17	SN	2023-05-01 08:49:35	ZTF23aaexcmm	t
ZTF23aacykvg	121.38550583125	49.40128315	2460036.7047917	2460065.7197338	17.8296	18.7074	18.3131	19.9287	2460065.681956	2460065.7197338	16	SN	2023-05-01 06:18:00	ZTF23aacykvg	t
ZTF23aaemsfm	183.96715512857145	30.166192492857135	2460043.8541319	2460065.7117014	18.9697	19.3612	19.1342	19.77	2460059.8212384	2460065.7117014	14	NT	2023-05-01 06:11:41	ZTF23aaemsfm	t
ZTF22abhwlnm	111.06460361666666	19.08130095833333	2460036.6979398	2460065.7065625	18.7506	18.9475	18.7506	18.9751	2460065.6767593	2460065.7065625	11	SN	2023-05-01 06:11:41	ZTF22abhwlnm	t
ZTF22abtsypf	104.73434780714287	39.63524867857142	2460036.6826042	2460065.7046759	18.4514	18.9883	18.7953	19.35	2460065.6833681	2460065.7046759	14	SN	2023-05-01 06:11:41	ZTF22abtsypf	t
ZTF23aaccacy	106.72776384285714	37.976539935714285	2460036.6826042	2460065.7046759	18.1952	18.195	18.2712	18.3421	2460065.6833681	2460065.7046759	14	SN	2023-05-01 06:11:41	ZTF23aaccacy	t
ZTF23aaedsfn	130.10593117368424	71.95604846842105	2460036.7062616	2460065.7013542	18.1396	18.0598	18.8098	19.2019	2460062.7170023	2460065.7013542	19	SN	2023-05-01 05:21:29	ZTF23aaedsfn	t
ZTF23aaedvtn	145.29992261666666	63.59469334761906	2460036.7301968	2460065.7003935	18.7155	18.5391	19.368	19.8585	2460065.6871875	2460065.7003935	33	NT	2023-05-01 05:20:00	ZTF23aaedvtn	t
ZTF23aadwhmn	161.4083574071429	54.0335109642857	2460036.8026968	2460065.6988889	18.8385	18.4462	19.252	20.0837	2460061.6759606	2460065.6988889	12	SN	2023-05-01 05:46:28	ZTF23aadwhmn	t
ZTF23aadtvuw	168.6070963705882	37.44624502941176	2460043.8627431	2460065.6984144	19.4086	19.5668	19.5541	20.0447	2460063.7174421	2460065.6984144	16	SN	2023-05-01 05:20:00	ZTF23aadtvuw	t
ZTF20acpzafj	170.11711619473687	38.42546934210526	2460036.8228241	2460065.6984144	19.4218	19.7286	19.6797	19.7286	2460063.7174421	2460065.6984144	18	SN	2023-05-01 05:20:00	ZTF20acpzafj	t
ZTF23aadvzrh	128.49080631538462	38.78960577692307	2460036.7781944	2460065.696956	19.2394	19.4448	19.3169	19.8142	2460059.6651852	2460065.696956	11	SN	2023-05-01 05:46:28	ZTF23aadvzrh	t
ZTF23aacjetk	125.22372975833333	39.241599275	2460036.7163542	2460065.6960185	18.1592	18.8792	18.3863	19.2818	2460058.6840625	2460065.6960185	12	SN	2023-05-01 05:20:00	ZTF23aacjetk	t
ZTF22abybbud	114.39689978823529	35.60552753529412	2460036.6993519	2460065.6955556	16.9058	17.917	17.2646	18.4512	2460065.6777199	2460065.6955556	17	NT	2023-05-01 05:20:00	ZTF22abybbud	t
ZTF23aacyfpi	116.8776459857143	22.60789682142857	2460036.6988889	2460065.695081	18.0664	18.2507	19.2087	19.9544	2460057.6963773	2460065.695081	13	SN	2023-05-01 05:46:28	ZTF23aacyfpi	t
ZTF23aaelfgk	120.21753816923076	16.411521919230772	2460043.6874074	2460065.6946065	18.8845	18.7855	19.1382	18.9775	2460059.6810995	2460065.6946065	21	NT	2023-05-01 05:20:00	ZTF23aaelfgk	t
ZTF23aadvpay	124.92005281875001	-3.2141401312499998	2460036.6912731	2460065.6921644	17.7489	17.6928	18.804	19.7748	2460065.6739352	2460065.6921644	31	SN	2023-05-01 05:46:28	ZTF23aadvpay	t
ZTF23aabyhrz	127.90136759444444	-5.572888527777778	2460036.6912731	2460065.6921644	17.8956	19.446	18.8675	19.7484	2460065.6666551	2460065.6921644	14	SN	2023-05-01 05:05:37	ZTF23aabyhrz	t
ZTF21aadnzyf	112.11929578571429	69.18817525714286	2460036.7057523	2460065.6852546	19.0441	19.3602	19.2671	19.4083	2460065.6852546	2460056.6804861	13	SN	2023-05-01 04:55:15	ZTF21aadnzyf	t
ZTF23aacfiuv	116.73469608461538	7.298861538461538	2460036.7895602	2460065.6748727	17.299	18.1268	17.7108	18.461	2460065.6748727	2460057.6827199	12	SN	2023-05-01 04:40:55	ZTF23aacfiuv	t
ZTF23aaelgnd	129.59213659411768	4.382509558823529	2460043.6899769	2460065.6715278	17.2732	17.2225	17.3583	17.5532	2460065.6715278	2460061.691875	17	NT	2023-05-01 04:40:55	ZTF23aaelgnd	t
ZTF19aatthkw	253.99403724117644	39.2050646	2460042.9075	2460064.9743056	19.6665	19.9911	19.7315	20.3353	2460064.9743056	2460064.9583796	11	NT	2023-04-30 15:20:09	ZTF19aatthkw	t
ZTF23aablhfk	259.22574341515156	44.268170272727275	2460042.8911458	2460064.973831	19.1796	19.4297	19.573	20.5257	2460064.973831	2460060.966088	16	SN	2023-04-30 13:46:00	ZTF23aablhfk	t
ZTF23aaejumk	259.40321419375	48.63828678125	2460042.8911458	2460064.9728935	18.8847	19.3406	18.9833	19.4451	2460064.9728935	2460060.9656134	12	SN	2023-04-30 13:53:41	ZTF23aaejumk	t
ZTF18aavxdrz	265.4612277111112	50.31800936666667	2460042.8911458	2460064.9728935	19.4187	19.5726	19.7021	20.2729	2460064.9728935	2460060.9656134	12	SN	2023-04-30 13:53:41	ZTF18aavxdrz	t
ZTF18abtnryk	273.63998100526317	60.95867780000001	2460042.9065278	2460064.9719444	19.1149	19.1281	19.2375	19.9542	2460064.9719444	2460064.9220718	13	SN	2023-04-30 13:42:22	ZTF18abtnryk	t
ZTF23aadslye	279.9558034157895	49.29617851578948	2460042.9031713	2460064.9695718	19.4477	19.4223	19.5488	20.4196	2460064.9695718	2460062.9729398	13	SN	2023-04-30 14:16:43	ZTF23aadslye	t

(50 rows)

Figure 2: A sample of the `tides_stage` table



(a) The Prefect Agent is looking for work!



(b) The scheduler via the online interface shows that our Flow will be triggered every hour.

Figure 3: Prefect Agent screenshots.

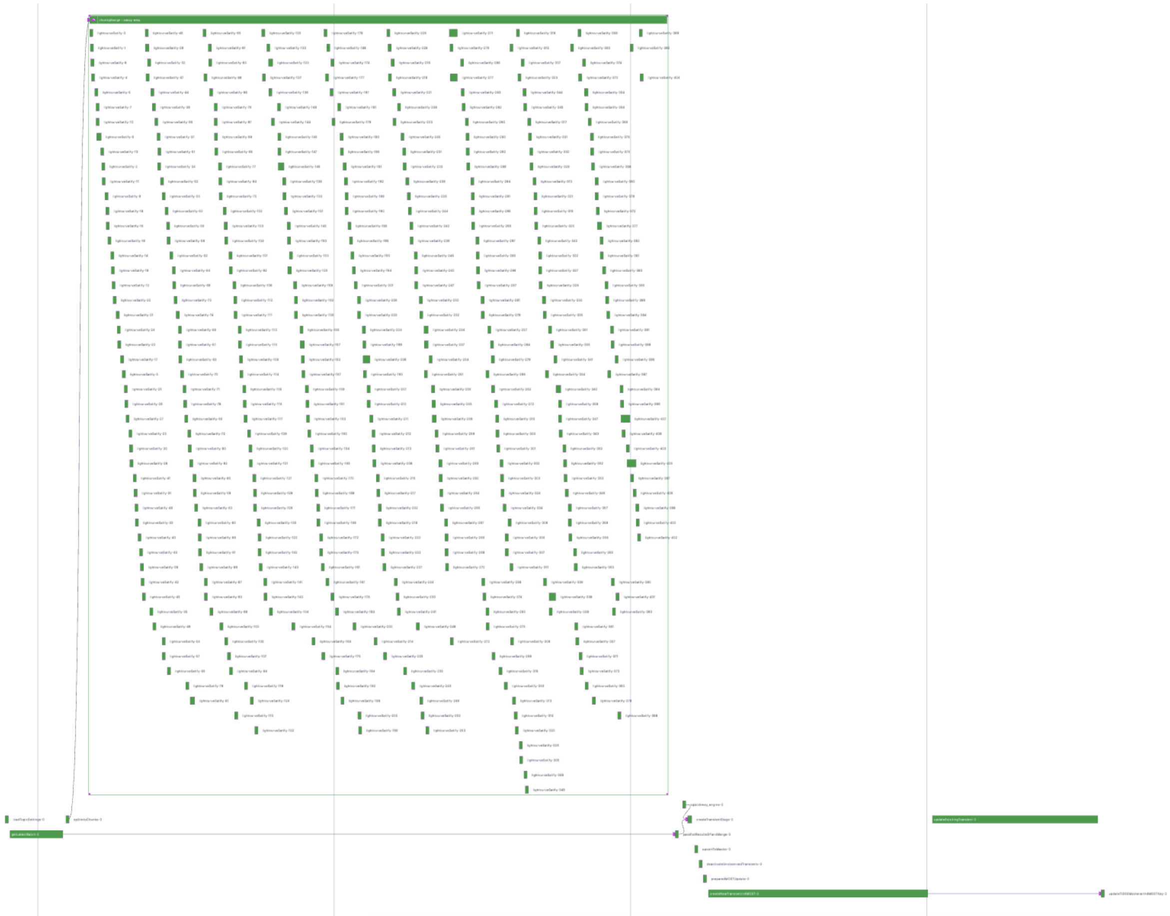


Figure 4: The TiDES flow on the Prefect web client. Each green block represents a function that was executed during this pipeline, with the horizontal length equal to the execution time. The waterfall of many small tasks under an umbrella flow shows the Dask parallel processing of transient light curves. Dependencies are shown by arrows connecting tasks/flows, although not all dependencies are show if executed outside the Prefect environment (e.g. PostgreSQL table operations).