Developing A Photometric Alert Stream for ZTF's Lower Variance Transients

SURF Student: Cason Shepard

Mentors: Dr. Matthew Graham, Dr. Niharika Sravan

Awarded the Harold and Mary Zirin Fellowship

Abstract

The Zwicky Transient Facility (ZTF) is a new optical time-domain survey that uses the Palomar 48-inch Schmidt telescope. ZTF's main goal has been to detect and map changes in the brightness of transients to learn about their physical properties. The development of an alternate photometric alert stream algorithm will be critical in the function of LIGO O4, as one of the main targets of monitoring during this new LIGO run will be Active Galactic Nuclei (AGN) in the error volume of a LIGO source. This algorithm utilizes the SQL query pipeline of the Infrared Processing and Analysis Center (IPAC) to detect which sources from the specified catalog of sources have been observed on a given night. Photometric data of corresponding identified sources are then pulled from IPAC using ztfquery and astropy packages. This algorithm uses a cluster-based query system that helps increase database query efficiency and reduce runtime.

Background

The Zwicky Transient Facility (ZTF) is based upon the Palomar 48-inch Schmidt telescope and has been an integral part of the development of time domain astronomy over the past few years [2]. Since its inception, ZTF's main goal has been to detect and map changes in the brightness of transients (objects in our visible universe that vary in brightness). In doing so, we can study these objects' variability to learn about their physical properties. Every two nights, ZTF surveys the entire northern sky, using a past sky map as a reference key to automatically detect changes in brightness over the five sigma threshold [2]. When a transient from the night sky is compared to the reference and this threshold of change is detected, an alert is sent in real-time to the alert stream, which records all noteworthy changes for that night. In total, each night, ZTF can produce up to one million alerts [2].

This system, however, is not ideal when we want to look at objects changing over longer timescales. Just around a year ago, ZTF detected the first plausible electromagnetic (EM) counterpart to a binary black hole merger (BBH) [1]. This detection seemed to support what would be expected of a merger of this scale within the accretion disk of an active galactic nucleus (AGN) [1]. These EM counterparts may not, however, display variance at a level needed to be automatically detected by ZTF. Instead, it may be weeks, months, or even years before these events reach five sigma variability. This presents a current limitation to ZTF, as it is not currently able to automatically monitor AGN. With the start of the LIGO O4 run approaching, the development of an alternate photometric alert stream will be critical in the function of LIGO O4, as one of the main targets of monitoring during this new LIGO run will be AGN in the error volume of a LIGO source.

The Algorithm

This project was tackled using the 'ztfquery' and 'astropy' packages. They each provide a set of useful tools that can be used to efficiently query data from IRSA (Infrared Science Archive) [3]. The algorithm takes in a CSV file of information on all the transients we wish to monitor. Each row in the file corresponds to an object's field, objID (object ID), RA (right ascension), and DEC (declination) values. Using this information, we will be able to perform a cluster-query system based on field values and then calculate the closest match of the data using the RA and DEC values for each object. Once this file has been added to the proper location, the

algorithm will then sort and create separate CSV files containing all the transient information for objects which share the same field value.

```
header= ["field", "objid", "ra", "dec"]

# turns csv  of transient data to array of lists called data_2d
with open("Data/data_sort") as file:
    data_2d = []
    reader = csv.reader(file, delimiter = ',')
    next(reader) # this will skip the header
    for row in reader:
        data_2d.append(row)

data_2d = data_2d[0:len(data_2d) - 1]

# generates csv file for each field in the data file
# also creates a list of the fields (field_list) for use with ztfquery later
field_list = []
for row in data_2d:
    field_list.append(int(float(row[0])))
    temp = data_2d[data_2d.index(row)-1];
    with open('Data/grid_'+str(int(float(row[0]))), 'a') as field_file:
        writer = csv.writer(field_file)
        if int(float(temp[0])) != int(float(row[0])):
            writer.writerow(header)
        writer.writerow(row)

# sorts the list of fields for progress tracking later
field_list = [*set(field_list)]
field_list.sort
```

Figure 1 shows the initial process that will take a file of transient information (in this case data_sort), and create separate CSV files containing information for transients of similar field values. For example, 'csv_123' will contain the information on all the transients located in field 123, and will be stored in the 'Data' directory. A list of the fields (field_list) is also created, which will track which fields have been completed as the algorithm runs.

Once the transients we wish to look at have been sorted into their corresponding field files, we begin the process of querying these fields and recording the data observed by ZTF.

```
# initializes instance of ztfquery
zquery = query.ZTFQuery()

# gets the julian date from 24 hours previous
day = date.today() - timedelta(days=1)
day_str = day.strftime("%Y-%m-%d")
jdate = time.Time(day_str).jd

# initializes start time and counter variables
# for progress tracking/runtime information
start = time2.time()
field_update_count = 0
object_update_count = 0


for field in field_list:

    # prints 'Completed query of field __' message after each iteration
    if field_list.index(field) != 0:
        print(f'Completed query of field {field_list[field_list.index(field)-1]},'
              f'this is field {field_list.index(field)}/{len(field_list)}')
```

Figure 2 is the first of four figures depicting the data retrieval algorithm. In this section, the date is collected and the runtime checkpoints are set. Then our loop through the field array begins, and a progress message is displayed for each completed iteration.

Upon querying the current field (denoted by the current iteration), load_metadata returns a data frame of many entries. Each of these entries corresponds to a unique psfcat.fits file. To find the correct entry containing the object we wish to observe, a SkyCoord catalog of the metadata is created using the RA and DEC values. Later on, this catalog is compared to each transient in the current field query to distinguish which psfcat.fits file contains the desired transient observation.

```
    zquery.load_metadata(kind="sci", sql_query=f'obsjd >= {jdate} and field = {field}')

    if len(zquery.metatable['ra']) != 0:
        # only increases the field_update_count if the metadata query is nonempty
        field_update_count = field_update_count + 1

        #creates a SkyCoord catalog of ra and dec values for the queried metatable
        float_ra = [float(i) for i in list(zquery.metatable['ra'])]
        float_dec = [float(f) for f in list(zquery.metatable['dec'])]
        metadata_catalog = SkyCoord(float_ra*u.degree, float_dec*u.degree)

        zquery.get_data_path(suffix="psfcat.fits")
        urls, locations = zquery.download_data(suffix="psfcat.fits", source="local", nprocess=8,
                                               download_dir='ztf_out/', show_progress=True, nodl=True,)
```

Figure 3 is the second of four figures depicting the data retrieval algorithm. Here, we see the metadata query using load_metadata. Then, if the metatable is non-empty, a SkyCoord catalog is created and the data is prepared for download later, using download_data.

```
        # creates field_data array which can be compared to the
        # metatable catalog to find closest match
        with open('Data/field_'+str(field)) as file:
            field_data = []
            reader = csv.reader(file, delimiter=',')
            next(reader) # this will skip the header
            for row in reader:
                field_data.append(row)

            # updated to count the number of transients observed
            object_update_count = object_update_count + len(field_data)

            # create csv files for data output
            with open('data_out/csv_'+str(field), 'w') as file:
                writer = csv.writer(file)
                header = ['objid', 'ra', 'dec', 'flux', 'sigflux', 'mag', 'sigmag']
                writer.writerow(header)
```

Figure 4 is the third of four figures depicting the data retrieval algorithm. Here, the sorted grid files are taken and turned into an array for access later. Furthermore, the CSV output files are created. For example, the observed data for transients in field 123 will be stored in file ‘csv_123’ in the ‘data_out’ directory.

Now, for each transient (object) in our list we wish to observe, the RA and DEC values will be stored in a SkyCoord instance. This SkyCoord is then compared to the SkyCoord of the metadata created earlier. The comparison results in an index value pointing to the specific psfcat.fits file containing the observation data on this specific transient. Once this file is downloaded, a new SkyCoord instance is created using the RA and DEC values stored inside. Once again, comparing this catalog with our original SkyCoord point results in the specific observation for our transient to be recorded. As shown in the Figure 4 header, we will record the object ID (objid), RA, and DEC values in the output file. We will also record flux, sigflux, mag, and sigmag in this output file. By running these SkyCoord comparisons for each transient in the given field, we can compile data for all transients observed in the past 24 hours. Figure 5 shows this process.

```
            for obj in field_data:
                if len(obj) > 0:
                    # creates reference SkyCoord object from the given transient info
                    original = SkyCoord(ra=float(obj[2])*u.degree, dec=float(obj[3])*u.degree)
                    index_meta_match, angle, quant = match_coordinates_sky(original, metadata_catalog, nthneighbor=1)

                    # load/download file for the closest match to the reference within the metadata catalog
                    temp_url = buildurl.filename_to_scienceurl(locations[index_meta_match])
                    io.download_single_url(temp_url,locations[index_meta_match], show_progress = False)

                    with fits.open(locations[index_meta_match]) as hdu:
                        data = hdu[1].data

                        # creates another SkyCoord catalog from the single downloaded file
                        float_ra = [float(i) for i in data['ra']]
                        float_dec = [float(f) for f in data['dec']]
                        catalog = SkyCoord(float_ra*u.degree, float_dec*u.degree)

                        # calculates the closest match between the reference and the new catalog
                        index_match, angle, quant = match_coordinates_sky(original, catalog, nthneighbor=1)
                        data_match = data[index_match]

                        # writes coordinate matched data to csv ouput file
                        newline = [obj[1], data_match['ra'], data_match['dec'], data_match['flux'],
                                data_match['sigflux'], data_match['mag'], data_match['sigmag']]
                        writer.writerow(newline)
```

Figure 5 is the final of four figures depicting the data retrieval algorithm. Notice how we compare the original transient SkyCoord to the metadata SkyCoord to download the correct file and then compare it once again to the file's SkyCoord catalog to find the correct measurement.

After testing this algorithm over several nights, an average runtime of 11336 seconds (3h:9m) was recorded. Furthermore, of the 1149 fields spanned by the transient survey file, an average of 296 fields are updated each night. This roughly translates to five thousand observations per night.

Sources

[1] Graham, Matthew. J., et al. “Candidate Electromagnetic Counterpart to the Binary Black Hole Merger Gravitational-Wave Event S190521G.” Physical Review Letters, vol. 124, no. 25, 25 June 2020, https://doi.org/10.1103/physrevlett.124.251102. \

[2] Graham, Matthew J., et al. “The Zwicky Transient Facility: Science Objectives.” Publications of the Astronomical Society of the Pacific, vol. 131, no. 1001, 2019, p. 078001., https://doi.org/10.1088/1538-3873/ab006c

[3] Mickael Rigault. (2018). ztfquery, a python tool to access ZTF data (doi). Zenodo. https://doi.org/10.5281/zenodo.1345222