CSCD530 Big Data Analytics - Project Fermispark

David Sergio, 00873694

March 16, 2025

Abstract

The goal of this project is to design a distributed data analysis framework for Gamma Ray Astronomy using big data and distributed data techniques. Specifically, design a distributed data pipeline and analysis framework using a local network Hadoop [5] and Apache Spark [8] clusters, and to demonstrate data analysis techniques such as binning adjustments and angular resolution tuning. This was accomplished using the Fermi Gamma-ray Space Large Area Telescope (LAT) [2] and its data, which is stored in FITS files on an FTP site [7]. The data was downloaded and converted to CSV format, then stored in a Hadoop HDFS cluster. The data was then processed using Spark on a local network cluster. The data was analyzed using Spark to perform analysis, such as the average energy per grid, binned energy, and binned count energy. The data was then visualized by using binned photon energy counts, processed into an array with a shape corresponding to the grid precision dimensions and energy bucket width. The array was then used to construct an image of the sky in gamma rays using Numpy and the Pillow (PIL) library in python. The constructed images demonstrate the gamma ray sky in the energy range 10-400000 MeV with various grid precision degree and energy bin widths. The code was uploaded to a public github repository [6]. The application can be run on any Hadoop/Spark cluster with the appropriate configurations, including environment variables, and directory configurations.

Introduction

Background

Gamma Ray Astronomy is the study of gamma rays from astronomical sources. Gamma-rays are the highest energy electromagnetic radiation. [9] The Fermi Gamma-ray Large Area Telescope (LAT) is a space telescope launched on June 11 2008 to perform sky surveys for gamma rays. The Fermi Large Area Telescope (LAT) is a telescope that detects gamma rays in the energy range from 10 MeV to approximately 400 GeV. The LAT scans the entire sky every day. The LAT has a large effective area and high angular resolution, and it is used to study gamma-ray sources, such as Pulsars, Supernovae, Neutron star mergers, and Supermassive black holes. [10] There are numerous studies on gamma ray sources, such as the specific studies on the Supernova Remnant Cassiopeia A. [12] and Classical Novae V1369 Centauri 2013 and V5668 Sagittarii 2015 [3] It is worth noting that these studies focus on very small sections of the sky. Single sources of gamma-ray photons are restricted to small angular sections of the sky. What this project aims to accomplish is to scale such focused analysis from small angular studies to massively parallelized analysis of the entire sky survey. One problem with such huge amounts of data in sky surveys is that it is difficult to process the data at scale. This is where big data techniques come in. By using distributed computing and big data techniques, we can process large amounts of sky survey data in parallel, aggregating data over time, and performing data analysis on a large scale without relying on massive catalogues which do not include real-time or recent data. We can also create new data catalogues in real time, and process and perform analysis on the data in real time.

The Fermi LAT has been in operation since 2008, and is still operational, although the orbit of the satellite has decayed over time. [1] There have been aggregations on the data, and catalogues, such as the 8-year catalogue of gamma-ray sources detected by the Fermi LAT. [11] However, it is very important to note that this catalogue is not real-time, and is limited to a specific time frame. This is the motivation for this project, to demonstrate that large catalogues of data can be constructed and processed in real-time using distributed computing techniques.

Data Pipeline

Data from the Fermi LAT is stored and can be queried in various ways. The data that will be used by this project is the weekly photon files hosted on the FTP site [7]. The weekly data archive is accessible via FTP. The FITS (Flexible Image Transport System) file format is a used to store multidimensional data for scientific applications. It can easily be converted to CSV format using the python library astropy [4]. The file format consists of headers and tables.

The data pipeline consists of FTP data ingestion, data processing, and data analysis.

- Data Ingestion - Download the data from the FTP site and store it in a local directory.
- Data Processing - Convert the data from FITS to CSV and store it in HDFS.
- Data Analysis - Use Apache Spark to process the data and calculate the average energy per grid, binned energy, and binned count energy.
- Data Visualization - Convert the binned photon energy counts to a CSV file, read using Pandas and processed into an array. Construct an image of the sky.

Data Ingestion

Data is ingested into the application by downloading from the FTP server. The data is downloaded from the FTP server and stored in a local directory. The data is then converted from FITS to CSV and initially stored in the following format:

Table 1: example CSV input data

TIME	RA	DEC	ENERGY
7.575345904229679E8	22.699495	-20.23353	255.21156
7.577122183870332E8	21.946495	-35.30692	194.79996
7.577968579858235E8	3.0868948	-21.585531	827.5504
7.576152464768566E8	14.83716	-20.842484	1177.6665
7.576211706464217E8	2.480437	-42.274548	195.38531
7.579903734654744E8	21.173525	-27.316578	276.86334
7.580209837771453E8	1.5604638	45.79497	121.49854
7.57785106202615E8	8.253764	-13.712734	263.71323
7.58041863906729E8	13.659444	-46.79668	411.51807
7.57973598035729E8	21.145754	-39.279194	522.3314

This data is then stored in a Hadoop HDFS cluster for further processing. The data in Table 1 is not ordered or sorted. It is worth noting that photons hit the detector from all incident angles, and the data is not sorted by RA or DEC. This is the form the data is in when it is ingested into the application.

Data Processing

To process the data, we will use Apache Spark on a local network cluster. The data will be read from HDFS and processed using Spark. The data will be processed in the following ways:

- Average Energy per Grid
- Binned Energy
- Binned Count Energy

Analysis Framework

The analysis framework consists of the following components:

- Data Analysis Algorithms - Spark Average Energy Mapper, Spark Binned Energy Mapper, Spark Binned Count Energy Mapper
- Data Visualization - Convert Counts to CSV Mapper which outputs a CSV file that can be read by Pandas and processed into an array which is then used for constructing images, visualizations, and other analytical results.
- As we scale this application to larger clusters, we can use more complex algorithms and more complex data visualizations.

Network Architecture

The network architecture consists of a local network cluster with a Hadoop HDFS cluster and a Spark cluster. The master node was configured as a Linux machine with 64 GB of RAM and 16 cores. The worker nodes were two MacBooks with 16 GB of RAM and a total of 20 cores. The Hadoop HDFS cluster was configured with a replication factor of 1. The Spark cluster was configured with 3 executors and maximum cores per executor. The challenges of this configuration was that by using machines with slightly different versions of python and Java, there were some compatibility issues. The solution was to use a virtual environment for python and to use the same version of Java on all machines. The configuration details are listed in the github repository [6] as well as instructions on setting up the environment variables and project directories.

Data Analysis

The data analysis is done using Spark MapReduce algorithms. This was done for the purpose of demonstrating that large amounts of data can be processed using distributed computing techniques, without relying on a single machine. By demonstrating this capability with a small network Hadoop and Spark cluster, we can show that this can be scaled up to larger clusters, and to more astronomical data with more complex algorithms. There were major challenges in processing data on a very small cluster, but the results showed that it is indeed possible.

Spark Average Energy Mapper

The average energy per bin is not particularly useful, but it is illustrative of how we process astronomical data binning using Spark. The following code snippet shows how we can calculate the average energy per grid using Spark.

```python
def sky_grid_mapper(row, grid_precision = 1):
    ra = row["RA"]
    dec = row["DEC"]
    energy = row["ENERGY"]

    # 1x1 degree grid
    ra_int = int(ra)
    dec_int = int(dec)

    # higher precision grid
    if grid_precision > 0:
        ra_precision = round(ra, grid_precision)
        dec_precision = round(dec, grid_precision)
    else:
        ra_precision = ra_int
        dec_precision = dec_int

    return (ra_precision, dec_precision), (energy, 1)

average_energy_per_grid_sorted = photon_counts_partitions.rdd \
        .map(lambda row: sky_grid_mapper(row, grid_precision)) \
        .reduceByKey(lambda a, b: (float(a[0]) + float(b[0]), float(a[1]) + float(b[1]))) \
        .mapValues(lambda s: float(s[0]) / float(s[1])) \
        .sortBy(lambda s: s[0][0], ascending=True) \
        .sortBy(lambda s: s[0][1], ascending=True) \
        .collect()

```

Spark Binned Energy Mapper

The binned energy mapper is used to bin the energy data into energy bins. The following code snippet shows how we can bin the energy data using Spark. Note that the bin width and max energy are parameters that can be adjusted.

```python
def sky_grid_energy_bins_mapper(row,
    bin_width_MeV = 100000,
    max_energy_MeV = 400000,
    grid_precision = 0):
    ra = row["RA"]
    dec = row["DEC"]
    energy = row["ENERGY"]

    # 1x1 degree grid
    ra_int = int(ra)
    dec_int = int(dec)

    # higher precision grid
    if grid_precision > 0:
        ra_precision = round(ra, grid_precision)
        dec_precision = round(dec, grid_precision)
    else:
        ra_precision = ra_int
        dec_precision = dec_int

    bucket_start = (energy // bin_width_MeV) * bin_width_MeV
    bucket_end = bucket_start + bin_width_MeV

    energy_bins_buckets = {(bucket_start, bucket_end): [str(energy)]}

    return (ra_precision, dec_precision), energy_bins_buckets
```

Spark Binned Count Energy Mapper

The binned count energy mapper is used to bin the energy data into energy bins and count the number of photons in each bin. This was by far the most useful of the MapReduce algorithms used. This allows us to analyze photon counts in customized ranges in both angular resolution and in energy bin width. The data extracted from this algorithm was ultimately used for the data visualizations. The following code snippet shows how we can bin the energy data and count the number of photons in each bin using Spark.

```python
def sky_grid_energy_bins_count_mapper(row,
    bin_width_MeV = 100000,
    max_energy_MeV = 400000,
    grid_precision = 0):
    ra = row["RA"]
    dec = row["DEC"]
    energy = row["ENERGY"]

    # 1x1 degree grid
    ra_int = int(ra)
    dec_int = int(dec)

    # higher precision grid
    if grid_precision > 0:
        ra_precision = round(ra, grid_precision)
        dec_precision = round(dec, grid_precision)
    else:
        ra_precision = ra_int
        dec_precision = dec_int

    bucket_start = (energy // bin_width_MeV) * bin_width_MeV
    bucket_end = bucket_start + bin_width_MeV

    energy_bins_buckets = {(bucket_start, bucket_end): ["1"]}

    return (ra_precision, dec_precision), energy_bins_buckets
```

Data Aggregations

The data aggregations were done using Spark MapReduce along with Hadoop storage. The data was aggregated using the reduceByKey function. The following code snippet shows how we can aggregate the data using Spark.

```python
def sum_lists(a, b):
    return [a[i] + b[i] for i in range(len(a))]

df = spark.createDataFrame(binned_energy_counts_per_grid_csv_week, schema)

binned_energy_counts_per_grid_csv_cum = df.union(existing_binned_counts_partitions)

binned_energy_counts_per_grid_csv_cum = binned_energy_counts_per_grid_csv_cum.rdd \
    .map(lambda x: ((x[0], x[1]), list(x[2:]))) \
    .reduceByKey(sum_lists) \
    .map(lambda x: (x[0][0], x[0][1], *x[1])) \
    .sortBy(lambda s: (s[1], s[0]), ascending=True)
```

Convert Counts to CSV Mapper

The convert counts to CSV mapper is used to convert the binned count energy data into a CSV row. The following code snippet shows how we can convert the binned count energy data into a CSV row using Spark.

```python
def convert_list_counts_to_csv_row(s,
    bin_width_MeV = 100000,
    max_energy_MeV = 400000):

row = []
for i in range(0, max_energy_MeV, bin_width_MeV):
    start = float(i)
    end = float(i + bin_width_MeV)
    key = (start, end)
    if (key in s[1]):
        list = s[1][key]
        row.append(';'.join(list))
    else:
        row.append("")

ret = (s[0])

for elem in row:
    sum = 0
    for i in elem.split(';'):
        if (i != ""):
            sum += int(i)
    ret += (sum,)
return ret
```

The result is a file in the following format (for 4 energy bins, with a grid precision of 0 and a bin width of 100000 MeV):

Table 2: Data Table for RA and DEC with Energy Bins

RA	DEC	0.0-100.0 GeV	100.0-200.0 GeV	200.0-300.0 GeV	300.0-400.0 GeV
0.0	-89.0	2.0	0.0	0.0	0.0
1.0	-89.0	4.0	0.0	0.0	0.0
2.0	-89.0	2.0	0.0	0.0	0.0
3.0	-89.0	2.0	0.0	0.0	0.0
5.0	-89.0	4.0	0.0	0.0	0.0
6.0	-89.0	8.0	0.0	0.0	0.0
7.0	-89.0	6.0	0.0	0.0	0.0
8.0	-89.0	6.0	0.0	0.0	0.0
9.0	-89.0	2.0	0.0	0.0	0.0
10.0	-89.0	2.0	0.0	0.0	0.0

Data Visualization

Data Visualization was achieved by using the binned photon energy counts CSV file, read using Pandas dataframe. The naming convention for the CSV file is such that the grid precision and bin width is in the name of the file. The CSV file contains the following columns:

- **RA** - Right Ascension
- **DEC** - Declination
- **Energy Bin 1** - The first energy bin
- **Energy Bin 2** - The second energy bin
- **Energy Bin ...** - The rest of the energy bins

When this data is read into a numpy array of size $(360 \cdot 10^p, 180 \cdot 10^p, 4)$, it can be used to construct an image. The following code snippet shows how we can convert each row to a set of array indices. These indices are X, Y pixel coordinates in the image. Something to note is that the RA and DEC are in degrees, and the image is in pixels. Another note is that this does not account for the curvature of celestial coordinates. As you go away from the celestial equator towards Declination +- 90 degrees, there will be curvature much like the higher latitudes meet at the north and south poles. This is not accounted for in this visualization, but could be a great future work project.

```python
def convert_RA_DEC_to_pixel(RA, DEC):
    RA_pixel = int(RA * 10 ** grid_precision) + 3
    DEC_pixel = int((DEC + 90) * 10 ** grid_precision) + 3

    return RA_pixel, DEC_pixel
```

When we process the data using this method, we can construct an image of the sky. The following image shows the sky in the energy range 10-400000 MeV with adjustable angular grid precision and energy bin width. This particular image was constructed with a grid precision of 1 and an energy bin width of 10000 MeV. It's worth noting that the image is mostly red. What this tells us is that the majority of the photons are in the lower energy bins. This is expected, as the Fermi LAT is designed to detect gamma rays in the energy range 10 MeV to 400 GeV. The majority of the photons detected are in the lower energy range. A higher resolution image will show that there are significant records of higher energy gamma-ray sources, visually represented by green and blue, although even in the high resolution visualizations, it is hard to tell where the green and blue sources are, because they are 1 pixel in width in most cases. There could be more work done to enhance the sparse higher energy sources so they are more easily distinguishable to the eye.

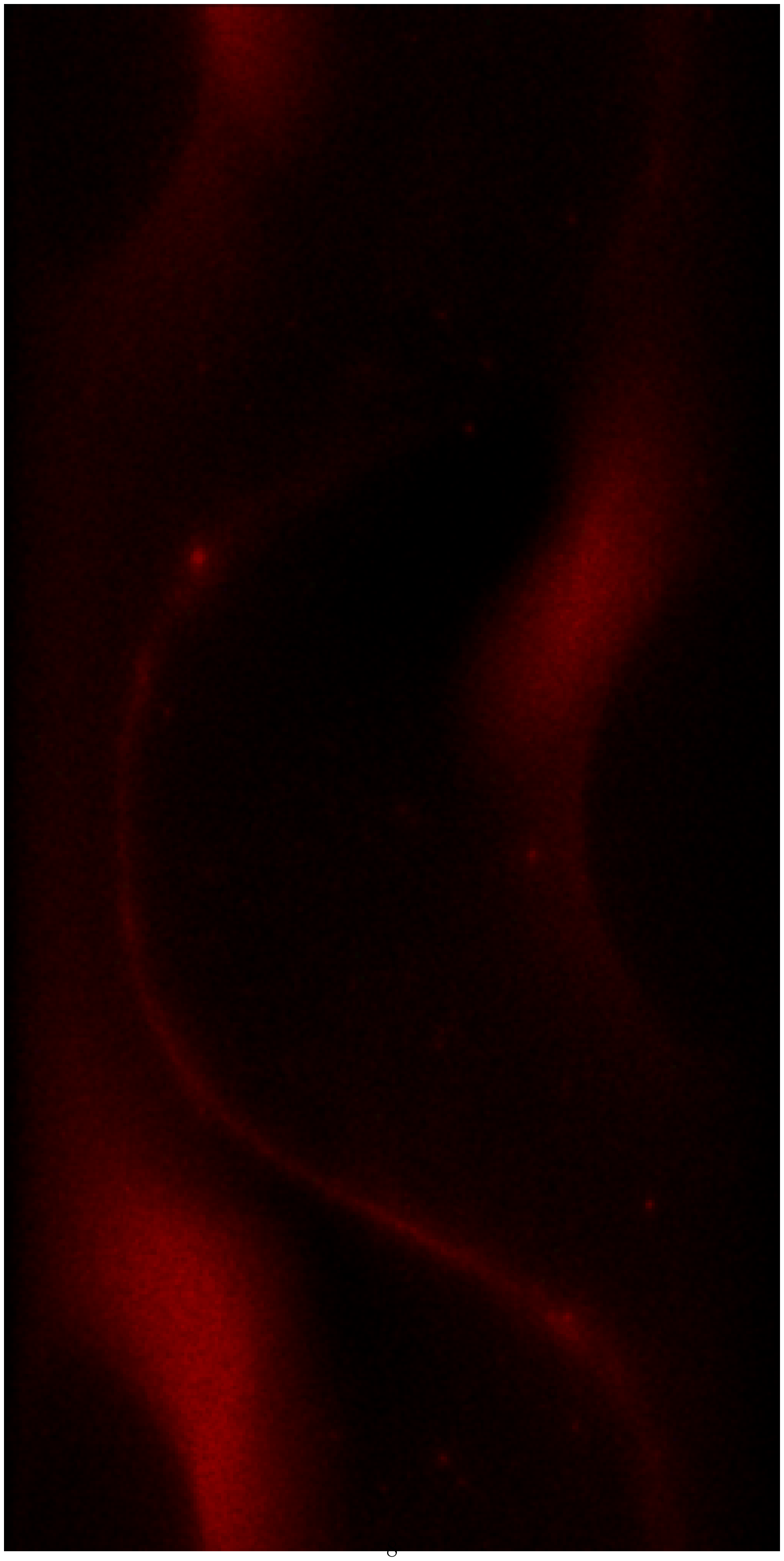

Figure 1: Constructed Image (Scaled)

Conclusion and Future Work

It was successfully demonstrated that it is possible not only to process astronomical data in one machine for one dataset, but to process astronomical data in a distributed manner and by using big data techniques, to aggregate large amounts of data over time, which can efficiently be processed with distributed algorithms and scaled up to larger clusters. Some of the challenges included the network architecture in the Hadoop/Spark cluster. The actual data analysis and data processing was relatively straightforward, but the network configuration and the computing resource management was challenging. When aggregating data in high grid degree precision, frequently the memory (RAM) usage would go as high as 60GB, and sometimes crash the master driver. Another limitation was that in the local network cluster, the worker nodes did not have the same resources, which required separate memory requirements on each worker node. While it would easily be possible to use a cloud provider (e.g. AWS EMR) to overcome these limitations, that solution would come at a computing cost, and it would be ideal to instead demonstrate that a local network of computers can achieve the same results. The data visualization was successful, but there are many more visualizations that can be achieved with future work.

There are many potential future projects involving this work, such as applying distributed computing to other sky surveys, such as the Sloan Digital Sky Survey. The data analysis can be extended to include more complex algorithms, such as machine learning algorithms. The data visualization can be extended to include more interesting visualizations.

References

[1] A. W. B. Ajello, M., Baldini, et al. Fermi large area telescope performance after 10 years of operation. *The Astrophysical Journal*, 2021. URL `https://arxiv.org/pdf/2106.12203`. Accessed: 2025-03.

[2] A. A. A. Atwood, W. B. et al. The large area telescope on the fermi gamma-ray space telescope mission. *The Astrophysical Journal*, 2009. URL `https://iopscience.iop.org/article/10.1088/0004-637X/697/2/1071`. Accessed: 2025-03.

[3] J. P. S. Cheung, C. C. et al. Fermi lat gamma-ray detections of classical novae v1369 centauri 2013 and v5668 sagittarii 2015. *The Astrophysical Journal*, 2016. URL `https://iopscience.iop.org/article/10.3847/0004-637X/826/2/142`. Accessed: 2025-03.

[4] T. A. Collaboration. The astropy project: Building an open-science project. *The Astronomical Journal*. URL `https://www.astropy.org`. Accessed: 2025-03.

[5] A. Hadoop. Apache hadoop, 2025. URL `https://hadoop.apache.org/`. Accessed: 2025-03.

[6] D. Sergio. Fermispark, 2025. URL `https://github.com/dsergio/fermispark`. Accessed: 2025-03.

[7] N. F. S. T. F. Server. Fermi ftp server, 2025. URL `https://heasarc.gsfc.nasa.gov/FTP/fermi/data/lat/weekly/photon/`. Accessed: 2025-03.

[8] A. Spark. Apache spark, 2025. URL `https://spark.apache.org/`. Accessed: 2025-03.

[9] F. S. Telescope. Fermi space telescope, 2025. URL `https://fermi.gsfc.nasa.gov/`. Accessed: 2025-03.

[10] N. F. S. Telescope. Fermi image gallery, 2025. URL `https://imagine.gsfc.nasa.gov/observatories/learning/fermi` Accessed: 2025-03.

[11] The Fermi-LAT Collaboration et al. Fermi large area telescope fourth source catalog. URL `https://fermi.gsfc.nasa.gov/ssc/data/access/lat/8yr_catalog/`. Accessed: 2025-03.

[12] F. S. Yuan, Y et al. Fermi-lat observations of the supernova remnant cassiopeia a. *The Astrophysical Journal*, 2010. URL `https://arxiv.org/pdf/1310.8287`. Accessed: 2025-03.