CSCD530 Big Data Analytics - Project Fermispark

David Sergio, 00873694

March 12, 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 HDFS and Spark clusters, and to demonstrate data analysis techniques such as binning adjustments and angular resolution tuning. This was accomplished using the Fermi Gamma-ray Space Telescope data, which is stored in FITS files on an FTP site [1]. The data was downloaded and converted to CSV format, then stored in a Hadoop HDFS cluster. The data was then processed using Apache Spark on a local network cluster. The data was analyzed using Spark to calculate the average energy per grid, binned energy, and binned count energy. The data was visualized by converting the binned photon energy counts to a CSV file, which was read using Pandas and processed into an array. The array was then used to construct an image of the sky in gamma rays. The constructed image showed the sky in the energy range 10-400000 MeV with precision 1 degree and energy bin width 1000 MeV.

Introduction

Background

Gamma Ray Astronomy is the study of gamma rays, the highest energy electromagnetic radiation. [2] The Fermi Gamma-ray Space Telescope 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 periodically. 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. [3]

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 [1]. 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. 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 injested into the application using the requests python library. The data is downloaded from the FTP site and stored in a local directory. The data is 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.

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 - 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 an image.

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.

Data Analysis

The data analysis is done using Spark MapReduce algorithms. This was done for the purpose of demostrating that large amounts of data can be processed using distributed computing techniques, without relying on a single machine. By demostrating 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.

```python
def sky_grid_energy_bins_mapper(row, bin_width_MeV = 100000, max_energy_MeV = 400000):
    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. 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):
    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
```

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 collumns:

- **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.

```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 0-400000 MeV with precision 1 and bin width 1000.

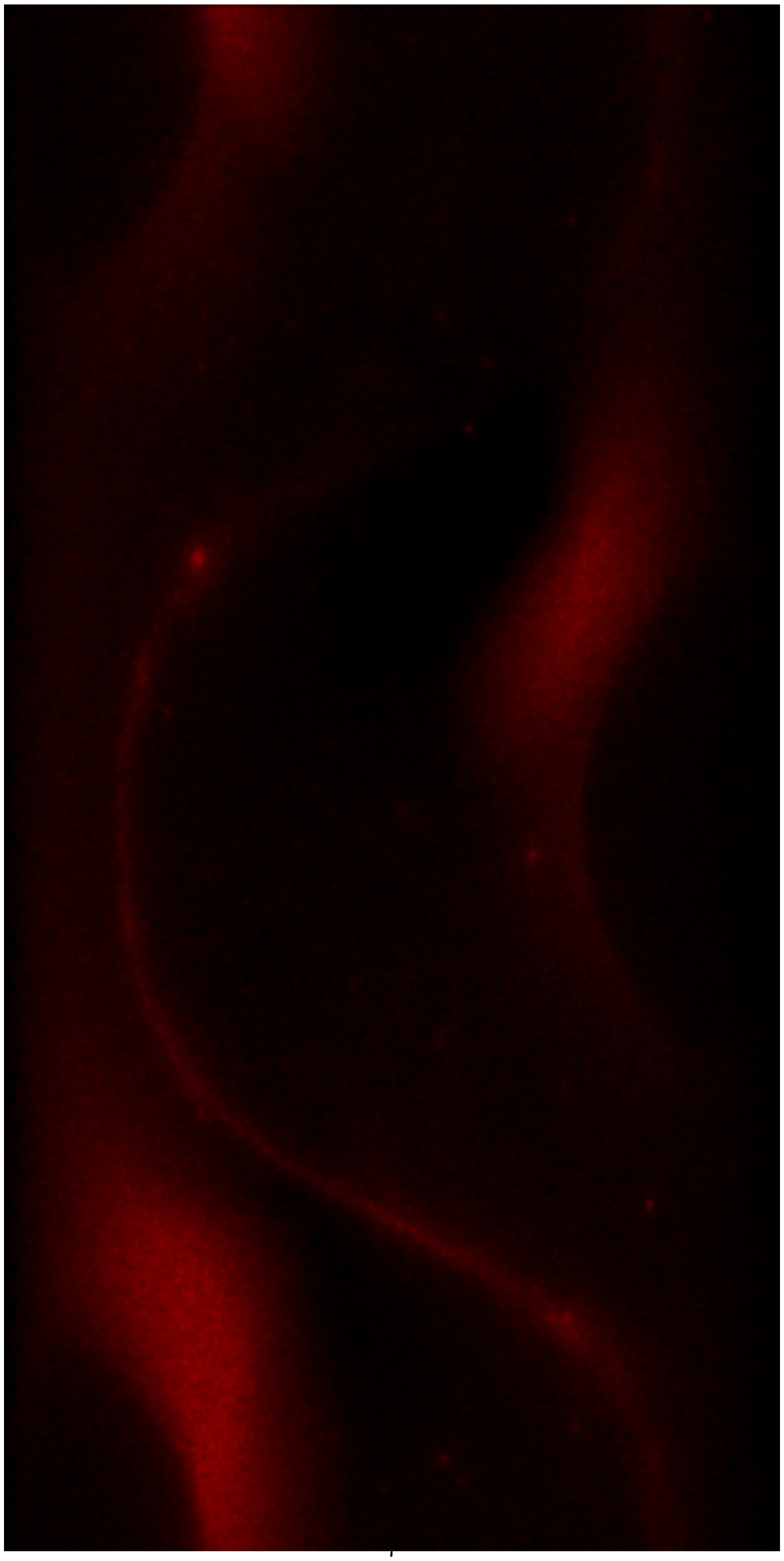

Figure 1: Constructed Image (Scaled)

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.

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] 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.

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

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