1. Problem Definition:

Implement Image Set Compression algorithm[1] using PySpark to enable distributed and parallel processing using HPC.

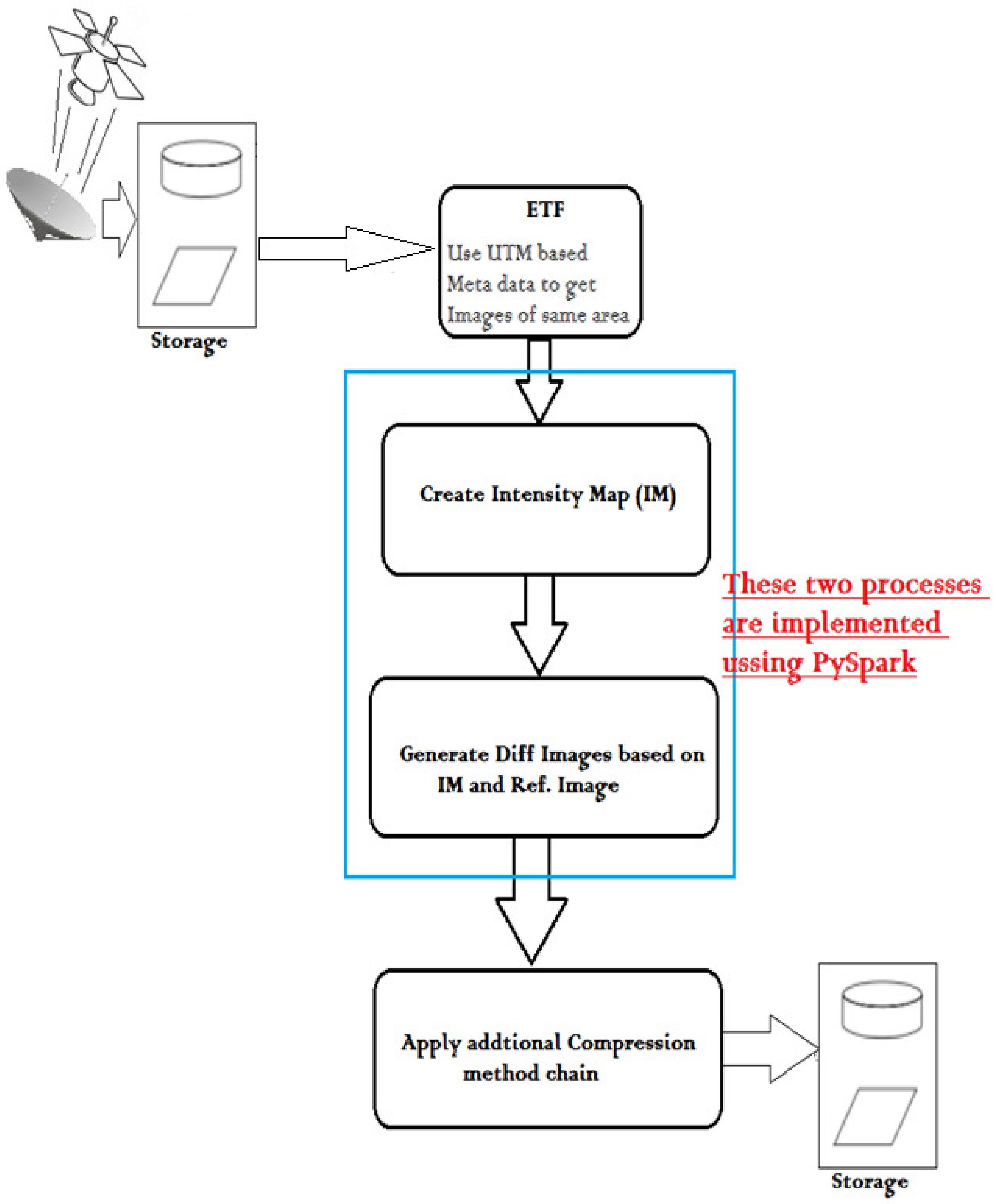
Generally speaking, majority of satellite imagery consists of images of planet Earth collected by satellites. These images are massive in size and often times carry more than three spectral bands. During the initial stage of experiments, it became very clear to me that memory and processing power of conventional computers can be very limiting. To overcome the above mentioned limitation, I am proposing a new **Floating Voxel (FV)** based framework that will use the power of HPC.

2. Introduction:

Data compression is typically achieved by removing redundancy in the given data. For lossless compression of individual images, typical algorithms attempt to remove two types of redundancy—interpixel redundancy and coding redundancy. These redundancies are reduced by a mapper (e.g. wavelet transform) and an entropy coder (e.g. arithmetic coding). For sets of similar images, there is an additional type of redundancy that exists among the images. A set mapper is first applied to reduce this inter-image redundancy. The output of the set mapper is a set of images that can be processed by ordinary image compression algorithms to reduce the remaining inter-pixel redundancy and coding redundancy.

Intensity Mapping (IM) based prediction technique enables efficient way to compress a pair of satellite images that have significant overlap in the underlying spatial areas. This prediction technique is relatively simple can be combined with any existing lossy or lossless compression methods. Furthermore, this technique can also provide an efficient way to compensate seasonal changes observed in satellite images.

Basic Flow Chart:



<u>3. Methodology & Implementation Details:</u>

As depicted in above given flow chart, the PySpark implementation of this implementation expects a set of spatially overlapped images as an input. One way to compute the spatial overlap is to use UTM based coordinates and apply standard polygon intersection algorithm [2] for extracting overlapped regions.

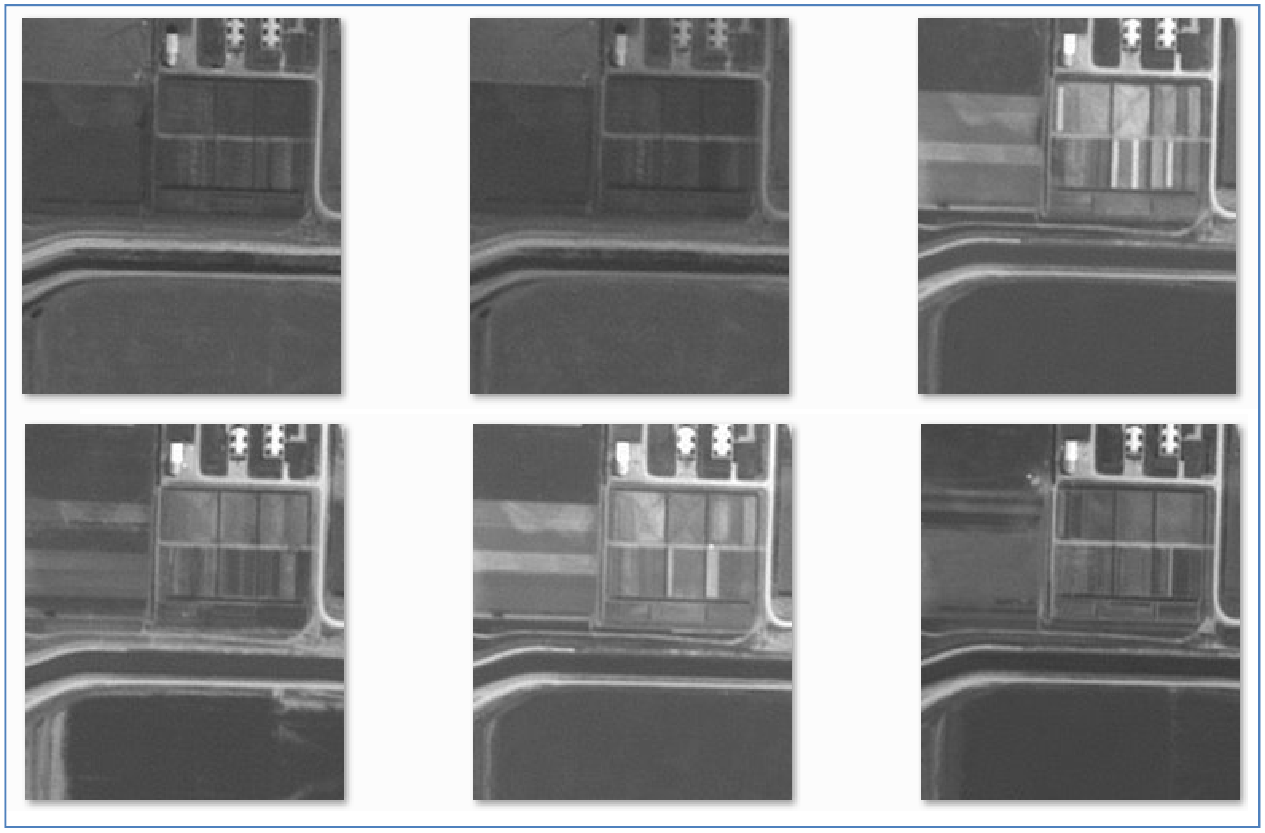


Image:1 Sample Image Set (200x200)

To create IM, the PySpark module first extracts the pixels that represents same location in each image and then adds image slice id. Following table shows the process with respect to pixel[0,0].

2-D Pixels		**3-D Voxel (representing location [0,0])**
ImageId	Pixel[0,0]	((**0, 0**), [(0, 81), (1, 87), (2, 130), (3, 125), (4, 91), (5,98)])
0	81	
1	87	
2	130	
3	125	
4	91	
5	98	

Table:1 Conversion from 2-D Pixel to 3-D Voxel

It is important to preserve the image ID along with pixel location because these voxels can be anywhere in cluster. Hence, the term coined as Floating Voxel (FV). Once we have these voxels in cluster, second step will be to calculate IM using them. To generate intensity based map, the algorithm will first create a map with respect to each image for all the applicable intensities (0-255). These FVs can be very handy for this process. Following table-2 shows the map derived using FVs.

IM Creation with Median values	
IMap for intensity 98	**Ref. Image ID**
(98,[(1, 92), (2, 96), (3, 92), (4, 119), (5, 102), (6, 96)])	**0**
(98, [(0, 99), (2, 97), (3, 133), (4, 123), (5, 111), (6, 89)])	**1**
(98, [(0, 99), (1, 107), (3, 136), (4, 211), (5, 129), (6, 110)])	**2**

Table: 2 - IM Creation

During the first stage, FVs are converted into IM with respect to each image as ref. Image. This process is easy to observe in the above given Table 2a. The IM row of intensity '98' with respect to Ref. Image Id = 0 doesn't carry tuple for that image. This IM is used to create difference images. Following Table:3 show the difference Voxels with respect to ref. Image=4.

Diff. image Voxels	**Cluster Node ID**	**Ref. Image ID**
((43, 3), [(0, -12), (1, -16), (2, -14), (3, 8), (4, 98), (5, 4), (6, 1)])	**32**	**4**
((29, 44), [(0, 3), (1, 4), (2, 1), (3, 1), (4, 127), (5, 4), (6, -1)])	**32**	**4**
((124, 110), [(0, 2), (1, 1), (2, -4), (3, -5), (4, 221), (5, -5), (6, -4)])	**32**	**4**
((115, 96), [(0, 1), (1, 6), (2, 3), (3, -9), (4, 225), (5, -12), (6, 3)])	**57**	**4**

Table: 3 - Diff. voxel with respect to Ref. Image 4

Note that these voxels are still in form of FVs. Table-3 also depicts these diff FV resident cluster nodes.

5. Final Output and Decompression

Finally, the difference images were constructed by grouping diff FVs. These image were stored along with IMAP.json and conf.json files. Down the road, other compression methods can be applied to achieve further compression but that's beyond the scope of this project.

Contrary to compression, the decompression operation expects the above mention difference images along with the conf.json and "imap.json" files. Decompression first loads the IM from "imap.json" which was generated by the compression stage. Then it reads the information about the ref. image information from the "conf.json" and reconstructs the images using IM.

6. Technology Used

- **Thunder** (http://thunder-project.org/): Thunder is a large-scale image processing library built on Spark, a new framework for cluster computing.
- **PIL - Python Imaging Library (**http://www.pythonware.com/products/pil/**):** PIL library supports many file formats, and provides powerful image processing and graphics capabilities.
- **The Spark Python API (PySpark) :** Enables Spark programming model for Python.
- **Tagged Image File Format (TIFF) :** Tagged image format (https://en.wikipedia.org/wiki/Tagged_Image_File_Format) is a popular image format and enable meta-data along with image data.
- **Eclipse IDE** : https://eclipse.org/ide/
- **Dell Cluster** (https://www.dell.com/learn/us/en/555/high-performance-computing-products-services)

6. Summary

In this project, I have tried to implement algorithm that can exploit a power of High Power Cluster(HPC) using Spark framework to enable set compression for massively large satellite images. The implementation uses compensation algorithm for images with significant spatial area overlap and taking into account of variations in seasonal and atmospheric conditions. It generates the prediction error image in two passes. During first pass it constructs the intensity map by scanning the images and in the second pass it uses the intensity map to compute the prediction error from the target image.

Refrences:

[1] V. Trivedi and H. Cheng, “Lossless compression of satellite image sets using spatial area overlap compensation,” in Image Analysis and Recognition, 8th International Conference, ICIAR 2011, Part II, 2011, pp. 243–252.

[2] O’Rourke, J.: Computational Geometry in C. Cambridge University Press, 2nd edn. (1998)