STEREO-PHOTOGRAMMETRIC DIGITAL ELEVATION/SURFACE MODELS: DATA EXTRACTION, STORAGE, RETRIEVAL AND PROCESSING.

Daryl Lopes
Gautam Kamath
Prasad Hirlikar
Ramakant Khandel
Vinay Rajagopalan
Vishesh Talreja
-Indiana University Bloomington

1. PROJECT BRIEF

"A spatial database, or geodatabase is a database that is optimized to store and query data that represents objects defined in a geometric space. Most spatial databases allow representing simple geometric objects such as points, lines and polygons."[1] The spatial data is related with geographic features and locations, or constructed features like states, cities, water bodies etc. The data in the case of spatial databases is stored in the form of points, coordinates, lines, topology, etc. The spatial data can be either stored in SQL or NoSQL databases.

In our data, each pixel of a geospatial image is either represents the date on which the picture was taken or the elevation or the match depending on the type of image. Well-structured and organized data like SQL decreases the performance with increase in volume of data and is also not scalable. As we were handling large amounts of data we chose to use NoSQL. "NoSQL is centered on the concept of distributed databases, where unstructured data may be stored across multiple processing nodes, and often across multiple servers. This distributed architecture allows NoSQL databases to be horizontally scalable; as data continues to explode, just add more hardware to keep up, with no slowdown in performance."[2]

In our approach, we begin by extracting pixel values of each sub-tile and store them in Hbase using a map-reduce job. We use Geospatial Data Abstraction Library (GDAL) to extract these pixel values. In the end the pixel values are read from Hbase using the ImageJ library and are converted into a .jpeg format.

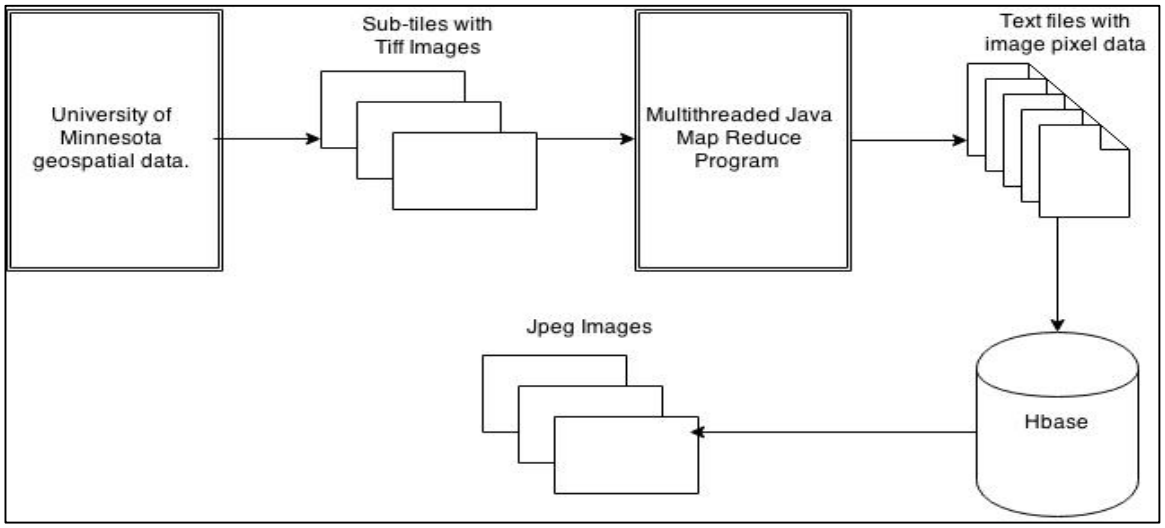

Fig 1.

2. SYSTEM OVERVIEW

The experimental system consisted of various components like data, virtual machines and software.

2.1 DATA

"The Digital Elevation Models (DEMs) are gridded surface elevation data constructed from overlapping pairs high-resolution (~0.5 m) images acquired by the DigitalGlobe, Inc. Worldview-1 and 2 satellites through the NGA EnhancedView license. The DEMs that were used by us was created using the Surface Extraction from TIN-based Search-space Minimization (SETSM) algorithm developed at the Byrd Polar Research Center, Ohio State University with funding from NASA."[3]

The datasets that were available to us are as follows:

- "32-bit GeoTIFF elevation raster - SETSM_<region>_<tile>_<subtile>_DEM.tif
- 32-bit GeoTIFF date raster - SETSM_<region>_<tile>_<subtile>_Date.tif
  Raster values depict the date of the stereo pair used to derive the elevation value. Pixels are integers representing the date in yyyymmdd format.
- Downsampled (32-meter resolution) elevation raster - SETSM_<region>_<tile>_<subtile>_DEM_overview.tif
- Downsampled (80-meter resolution) hillshade image - SETSM_<region>_<tile>_<subtile>_browse.jpg
- 8-bit GeoTIFF Match Raster - SETSM_<region>_<tile>_<subtile>_Match.tif
  Raster values depict which pixels were calculated by the DEM extraction process (value:1) and which were interpolated (value:0).
- Metadata & README file - SETSM_<region>_<tile>_<subtile>_meta.txt "[4]

2.2 VIRTUAL MACHINES

We are using linux systems. We have created thee Virtual Machines on future grid. These three virtual machines intercommunicate between each other using SSH.

Following is the configuration of our virtual machine.

Architecture	x86_64
CPU op-mode(s)	32-bit, 64-bit
CPU(s)	8
Thread(s) per core	1
Core(s) per socket	1
Socket(s)	8
NUMA node(s)	1
Vendor ID	Genuine Intel
CPU family	6
Model	26
Stepping	3
CPU MHz	2933.434
BogoMIPS	5866.86
Virtualization	VT-x
Hypervisor vendor	KVM
Virtualization type	full
NUMA node0 CPU(s)	0-7

2.3 SOFTWARE AND LIBRARIES

Hbase and Hadoop:

- "Apache Hadoop is an open source software project that enables distributed processing of large data sets across clusters of commodity servers."[5]
- "HBase is a column-oriented database management system that runs on top of HDFS. It is well suited for sparse data sets, which are common in many big data use cases."[6]

Gdal:

- The DEMs for a tile included Elevation and Date raster, which were GeoTIFF images. The initial approach was using OpenCV library in order to read the pixel value for the Date and Elevation image. The size of Date and Elevation images was an issue while using this approach. OpenCV does not help us read images that are bigger in size. Gdal on the other hand helped us in overcoming this problem. Gdal is a library that is mainly compatible with C++ application. So we initially had to create a .jar file for the Gdal so that our application could support it and latter on we used the Band and Dataset class provided by Gdal for reading these images successfully.

ImageJ:

- ImageJ is basically Java based image-processing library that can be used via Java plugins. Once the image was read using GDAL and the pixel value was saved in Hbase we were required to recreate the image in Jpeg format so that further analysis of the image can be done easily and the overhead of geotiff images would be overcome. ImageJ library helped us in this part. We used ImageJ library to convert date and elevation value to Jpeg images.

3. IMPLEMENTATION

3.1. INSTALLATION

- **VIRTUAL MACHINES**

  We create virtual machine using openstack Juno Openstack on Future Grid (India Cluster)-cloud infrastructure. "OpenStack is a set of software tools for building and managing cloud computing platforms for public and private clouds."[9]

  Steps involved:

  1. Load the Nova client tools.
  2. Create "Novarc" files for credentials/environmental variables.
  3. Check you're SSH key (For secure login to virtual machine).
  4. If key-pair name does not display your name add it.
  5. See available flavors and images. Flavors are virtual hardware templates in openstack, they define sizes of RAM, disk, number of cores, etc. Images on the other hand are virtual machine templates.
  6. Request your new virtual machine.
  7. IP address of your created virtual machine can be seen using "Nova list".
  8. Connect using the listed IP address.

  We have created three virtual machines: Khandelr-001, Khandelr-002 and Khandelr-003

- **CLUSTER PREPARATION**

  Prior to deploying Hadoop, your nodes must be able to communicate. This will require changes to the /etc/hosts configuration file, and creation and sharing of SSH keys among the nodes of the cluster.

Add lines to the end of your /etc/hosts file, one line for each node in the cluster, listing the IP address, fully qualified host name, and an alias for the host to be used by Hadoop. Here is an example of an /etc/host file with six nodes added. Of course you will use the proper IP addresses and host names for your VMs. You can use the same hosts file for every node in your cluster.

127.0.0.1 localhost

The following lines are desirable for IPv6 capable hosts

::1 ip6-localhost ip6-loopback

fe00::0 ip6-localnet

ff00::0 ip6-mcastprefix

ff02::1 ip6-allnodes

ff02::2 ip6-allrouters

ff02::3 ip6-allhosts

lines added for Hadoop cluster

10.39.1.46 smccaula-101.novalocal hadoop1

10.39.1.47 smccaula-102.novalocal hadoop2

10.39.1.55 smccaula-103.novalocal hadoop3

10.39.1.56 smccaula-104.novalocal hadoop4

10.39.1.57 smccaula-105.novalocal hadoop5

10.39.1.45 smccaula-106.novalocal hadoop6

Your nodes will also SSH authentication to communicate. For each node, you will need to create a pair of SSH keys (as root). The following command will create a key pair in /root/.ssh

` ssh-keygen -t rsa -P ""

You will need to append the public key created (default will be id_rsa.pub) to the authorized_keys file of each node. Test this by verifying that you can SSH from node to node in either direction.

- **JAVA**

  The commands to install Java are as follows:

  1. apt-get install default-jre.
  2. apt-get update.
  3. apt-get install openjdk-7-jdk.
  4. Set the environment variable Java_Home={Directory in which Java is installed} in ".bashrc" file

- **HADOOP AND HBASE**

  We have installed Hadoop 2.6.0 and for Hbase 0.98.9-Hadoop2. Perform the following steps in all the virtual machines.

  Steps involved in Hadoop:

  1. Download the setup "tar.gz files" for Hadoop as per the version specified above in a directory.
  2. Decompress the files using "tar" command.
  3. Set the required environment variable in ".bashrc" file like hadoop home, mapred home, yarn home, conf dir, common home, hadoop prefix.
  4. HDFS configuration:

  - hdfs-site.xml : configure the name node and the date node directory.

- core-site.xml : configure namenode URI

5. Yarn configuration:
   - yarn-site.xml : configure the hostname of the resource manager and its allocation configuration details.
6. Set Java_Home variable in hadoop-env.sh
7. Set the "hosts" file in etc directory with IP address for NameNode and DataNode as follows:
   - Hosts file content:

     IP address namenode

These steps need to be performed only for the namenode.

1. Setup the folders as follows:
   - Format the namenode directory.
   - Start the name node.
   - Start the data node.
2. Start the yarn Daemons:
   - Start the resource manager.
   - Start the node manager.

Steps involved in HBase:

1. Download the setup "tar.gz files" for Hadoop as per the version specified above in a directory.
2. Decompress the files using "tar" command.
3. Edit conf/hbase-site.xml, which is the main HBase configuration file. At this time, you only need to specify the directory on the local filesystem where HBase and ZooKeeper write data.
4. Set the required environment variable in ".bashrc" file like hbase home.
5. Set Java_Home variable in hbase-env.sh
6. Set the "hosts" file located in etc directory with IP address for HMaster and Region Servers as follows:
   - HMaster hosts file content:

     IP address master

     IP address regionserver#1

     IP address regionserver#n
   - Region Server hosts file content:

     IP address master

     IP address regionserver
7. Set the "regionservers" file located in $hbase_home/conf directory with IP addresses of all the region servers.

- **GEOSPATIAL DATA ABSTRACTION LIBRARY**

"GDAL is a translator library for raster and vector geospatial data formats that is released under an X/MIT style Open Source license by the Open Source Geospatial Foundation."[7]

GDAL has successfully been IRIX, BSD, Solaris, Linux and MacOS X. The steps required to build GDAL on UNIX are as given in the Figure below.

```
% cd gdal
% ./configure
% make
% su
Password: ********
# make install
# exit
```

Fig 2. GDAL Installation [8]

After installation it is necessary to set the LD_LIBRARY_PATH to include usr/local/lib so that the shared library can be found.

3.2 ALGORITHM AND APPROACH

Reading:

Image Data:

A script was implemented to download the images form the polar science website. The parent folder has subfolders wherein the name of the subfolder corresponds to the name of the sub-tile. The subfolders has the images corresponding to the sub-tile.

Issues Faced while Reading

The images were copied to HDFS. We implemented a MapReduce job to read the image from the HDFS and store the same in HBase. While running the job we faced 'Native Library Load Failed' exception. This exception occured while reading the pixel values of the images using the GDAL library in a distributed configuration. However, we were successfully able to run the program on a stand-alone node. To ascertain this we tried running the program for a dummy text file which worked fine which guarantees us that there is no configuration issues with HDFS. We tried copying the GDAL jar and '.so' files in HDFS and setting the path of the same in configuration. We also tried setting the gdal.jar in the classpath. We tried to fix this issues but were unable to do. After analyzing the issue, we came to the conclusion that the error was due to the failure to load GDAL JNI for images on the Hadoop cluster.

Approach:

We implemented a multithreaded java program to read the pixel values of geotiff images using GDAL library. These pixel values were then stored in the text files. Each row in the file has a row number followed by the pixel value of the Date, Match and DEM images. The pixel values representing a particular column are comma separated and different columns are separated by semicolon.

The structure of the text file is as follows:

row_number1; pixel1_date_image, pixel1_DEM_image, pixel1_match_image; pixel2_date_image, pixel2_DEM_image, pixel2_match_image; pixel3_date_image, pixel3_DEM_image, pixel3_match_image;.....

row_number2; pixel1_date_image, pixel1_DEM_image, pixel1_match_image; pixel2_date_image, pixel2_DEM_image, pixel2_match_image; pixel3_date_image, pixel3_DEM_image, pixel3_match_image;.....

Storage:

Approach:

The generated text files are then copied into the HDFS. We implemented a MapReduce job to read the text files from HDFS and store the pixel values into HBase. Each line is read from the text files and are split on a semicolon. Each index of the string array corresponds to the pixel values of the images. This data is then stored into Hbase.

The table name in HBase corresponds to the name of the text file in HDFS. The name of the column family is pixels. The name of the column corresponding to the columns family is 'pixel' followed by column number. For eg Pixel1, Pixel2, Pixel3….The row key is the 'row' followed by row number. For e.g. row1, row2, row3….

Extraction:

Issues Faced while Extracting the Pixel Values:

The pixel values stored in Hbase are then read to generate the Jpeg images using ImageJ library. The name of the HBase tables containing the pixel values were stored in the text files and copied to HDFS. These files were then read by the mapper to determine the table name. A mapreduce job is used to read the pixel values from Hbase. Each time the algorithm tried to read the pixels values of the images from HBase, 'java heap space error' occurred. We fixed it by setting the value for the environment variable in eclipse and also setting the value for HADOOP_DATANODE_OPTS along with HADOOP_NAMENODE_OPTS variable. Currently we are facing 'ExitCodeException'. However the program runs successfully on a stand alone machine.

Approach:

We implemented a multithreaded java program to read the pixel values from hbase and generate JPeg images using the ImageJ library.

Commands:

```
# make sure the output directory doesn't exists
./hadoop dfs -rmr output

# create the input directory
./hadoop dfs -mkdir input

# copy the files to the input directory
./hadoop dfs -put /home/ubuntu/Images/* /user/root/input

# view the contents of input directory
./hadoop dfs -ls /user/root/input

# run the word count
./hadoop jar Image.jar com.example.Read input output          #execute command

# view the contents of the output directory
./hadoop dfs -ls /user/root/output/

# view the results
./hadoop dfs -cat output/part-r-00000
```

3.3 AUTOMATION SCRIPTS FOR VIRTUAL MACHINE CREATION AND HADOOP HBASE INSTALLATION:

The below scripts are used for automating the process of creating a working state of Hadoop cluster.

- **Config.file**
  This file includes the key value pair for all the variables that required to be used for logging into the VMs.
- **Create_VM.sh**
  This script creates the virtual machine as well as help you sign in into the virtual machine. The variables used in this script come from config.file.
- **Hdfs-site.sh, Core-site.sh, yarn.sh, Hbase-site.sh**
  These scripts will setup the Hadoop cluster and HBase in a specific configuration needed for running this project

REFERENCES:

1. http://en.wikipedia.org/wiki/Spatial_database
2. https://blog.udemy.com/nosql-vs-sql-2/
3. http://www.pgc.umn.edu/elevation/stereo/
4. http://www.pgc.umn.edu/elevation/stereo/setsm
5. http://www-01.ibm.com/software/data/infosphere/hadoop/
6. http://www-01.ibm.com/software/data/infosphere/hadoop/hbase/
7. http://gdal.org/java/
8. http://trac.osgeo.org/gdal/wiki/BuildingOnUnix
9. http://opensource.com/resources/what-is-openstack