Jon Healy
Jamie Hale
Omowunmi Olalude

CSC 462/562 - Distributed Systems Project
Geospatial Data Hub

TABLE OF CONTENTS

Theme

Lidar and Satellite imagery is becoming more and more important in the world everyday. Companies and organizations are also producing and consuming greater and greater volumes of these types of data. New and exciting applications continue to be discovered for geospatial data. Recently, numerous Mayan ruins were discovered on the Yucatan peninsula in Mexico using Lidar data as laser points are able to pierce through thick jungle foliage to expose things and patterns that normal satellite imagery can not. Being able to view, study, share, manipulate and build algorithms around the corresponding data is a very challenging problem. High resolution files can be massive and storing this data in a useful way is a current and ongoing issue. Our project involved researching, learning and implementing as many solutions as we could as a team in the short period of time we had this semester.

Process

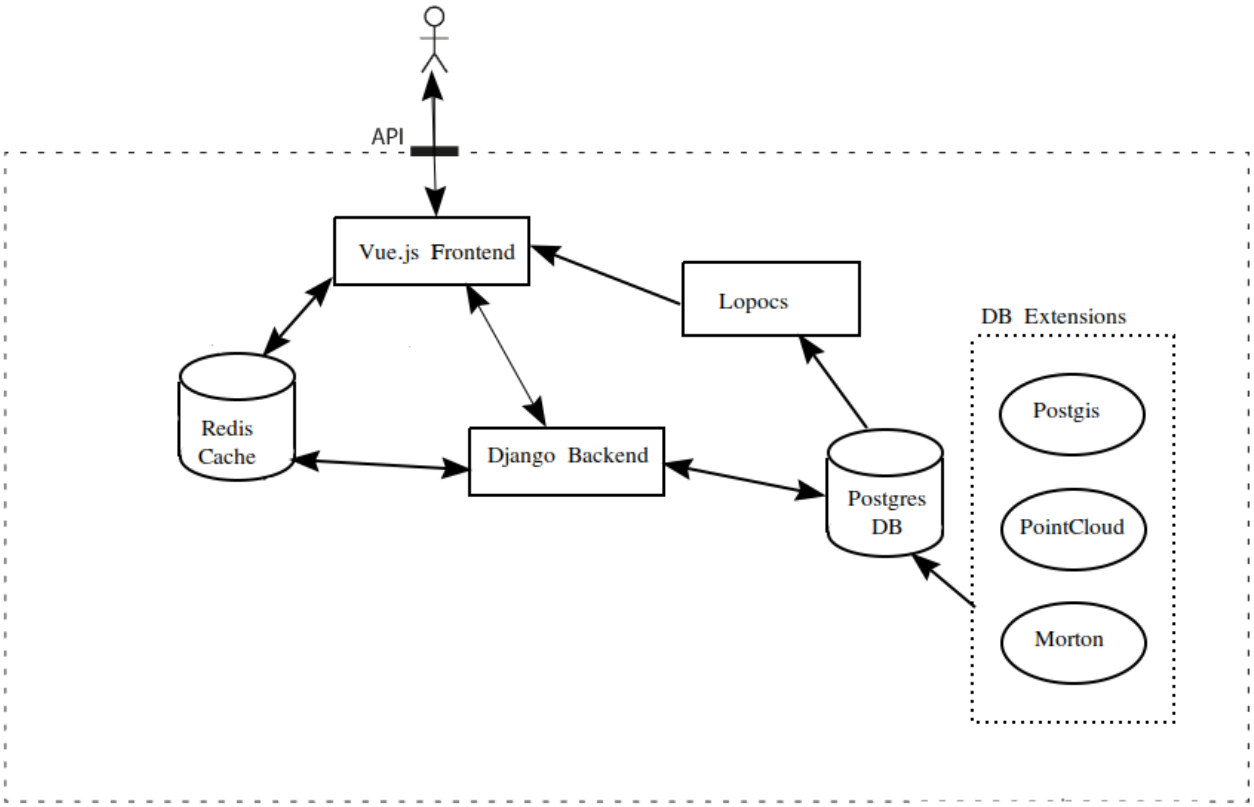
Our team had very little prior experience with geospatial imagery. The beginning of our project involved accumulating knowledge about how to work with this data and how to build a distributed system around it. In the process we ended up building two separate repos. One built into docker containers and ready for distributed deployment and another which is deployed, distributed, and tested.

Research Project : geospatialDataHub

Repo → https://github.com/noonespecial009/geospatialDataHub

Design

Build → Django, Postgres, PostGIS, pgpointcloud, pgmorton, Vue, Lopocs, Redis, Docker



Purpose

The goal of this project was focused on building a full stack application that can be distributed and scaled which has the ability to efficiently store point cloud data in a database in a way that machine learning algorithms can be applied effectively.

Details

From very early on we knew that we wanted to put the various components of our project into Docker containers. Making all the docker services synchronize and configured correctly was the biggest challenge for this project. The lack of resources and similar open source builds made the progress time consuming. We did eventually get our build running in docker containers correctly and running with docker compose. The docker integration process was constant for all the system components. Although we focused a large part of our development time on our other project, the scaffolding was ultimately created for a modern and potentially useful system. The project is containerized and could be distributed and deployed in a short period of time.

Our Django backend was used for producing both the API for the client and the tools for interacting with our PostGIS equipped Postgres database. We used a version of Django called Geodjango which is used in combination with geospatial libraries such as GDAL and PDAL to create data models of various file types. Django can then be used to interact with specific variables of the complex data models.

Our database was the most important area of research that we focused on for this project and also involved the most work. Geodjango suggests using a Postgres database with the addition of the PostGIS extension so that is what we did. At first we didn't have many issues other than standard troubleshooting, but then we discovered the open source point cloud viewer LOPoCS, which uses the Postgres extensions; pgpointcloud, pointcloud_postgis, and pgmorton. We researched and discovered a lot about the data we were trying to store. While you could spend a very long time to understand all the details we realized that the data descriptions of the high resolution 3d data was to big to simply store in an SQL table. These extensions are used to compress, structure, hash and layer the data so it can be efficiently stored in postgres. Also any meaningful analysis, machine learning or AI algorithms would be extremely challenging without using these extensions to structure the data.

Getting the extensions to work locally on a single machine wasn't overly challenging, although there was an exceptional amount of package installs and configuring involved. Also, after a considerable amount of time the team realized that working on Windows was way too time consuming and two team members installed Linux which helped considerably. Installing Linux was also a process as we hoped to run Linux on virtual machines but ran into too many problems that were not easy to reconcile. The most challenging part of our project, however, was integrating all of these extensions into Docker containers. There are no Docker images available that combined all of the Postgres extensions. In the end we used open source repos and built our dockerized database system from scratch. This took dozens of hours but resulted in a unique and valuable build that taught our team many skills.

We never fully implemented our LOPoCS enabled version as we decided to focus on our other project as the due date was drawing closer. LOPoCS is a point cloud server written in Python,

which should allow us to load Point Cloud data from a PostgreSQL database thanks to the pgpointcloud extension. On their github page, LOPoCS claims to be the only 3DTiles server that is able to stream point cloud data for viewing from pgpointcloud but we do not know if this is still the case.

The pgpointcloud extension for Postgres was developed by Paul Ramsey (http://s3.cleverelephant.ca/foss4gna2013-pointcloud.pdf) whose work on this project was funded by Natural Resources Canada. The pgmorton extension uses the Morton algorithm to tile and display data as it loads gradually and is designed to work hand in hand with pgpointcloud which allows us to store billions of points in a Postgres database. Storing points in this way would allow us to do all sorts of complicated querying to do things like maybe search for shapes or anomalies in large images or potentially even in sets of images. Our thoughts were that having this functionality enabled would have significance for many applications.

The LOPoCS repository has not been very active in the last couple of years and there is next to no documentation available. We did not fully realize how sparse the documentation is until we had spent a considerable amount of time getting all of our extensions built from scratch, implemented in Docker. We only had a couple of weeks left until our project was due and we were also having problems getting out torrent client working in Vue so we decided to switch our efforts back to the geospatialDataStore repository.

Our geospatialDataStore repository was started in the first week the project was assigned but we created and began working on the geospatialDataHub repository instead because we got excited about working with GeoDjango. We discovered Entwine and added it to the geospatialDataStore project as it is a far more popular viewer that is now used quite a lot. Compared to working with LOPoCS, Entwine is much easier to use however, it doesn't allow for the same functionality as it does not store point cloud data in Postgres or do anything similar. We did get quite far on this version of our project and it is something that should be worked on in the future. We have everything dockerized and ready to run which is significant as installing all of these things even in Linux without Docker is not trivial.

Future Work

The project is potentially set up for machine learning applications or other data intensive algorithms. Work like this can be coded in Python directly into the Django backend as Django is a Python framework. Having a frontend where clients could upload their data into the system and have algorithms performed and then results visualized and returned is a possibility. Realistically much of the scaffolding is there for a number of different applications that involve 3-dimensional data. We did not work on distributing this project to a swarm or the cloud. We added a point cloud viewer built by Esri but it does not have the punch that Entwine does. This project is dockerized which would allow it to be deployed fairly easily. Any application that used our system would be extremely data intensive.

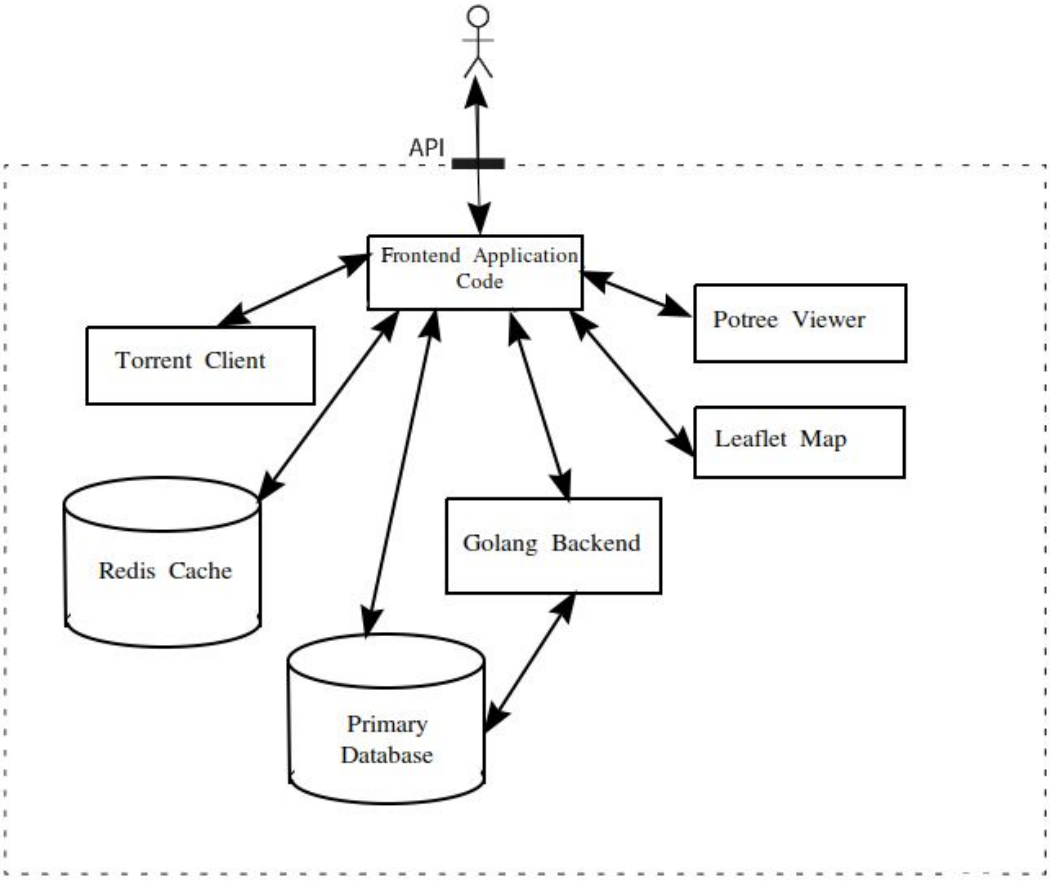
Main Project : geospatialDataStore

Repo → https://github.com/noonespecial009/geospatialDataStore

Design

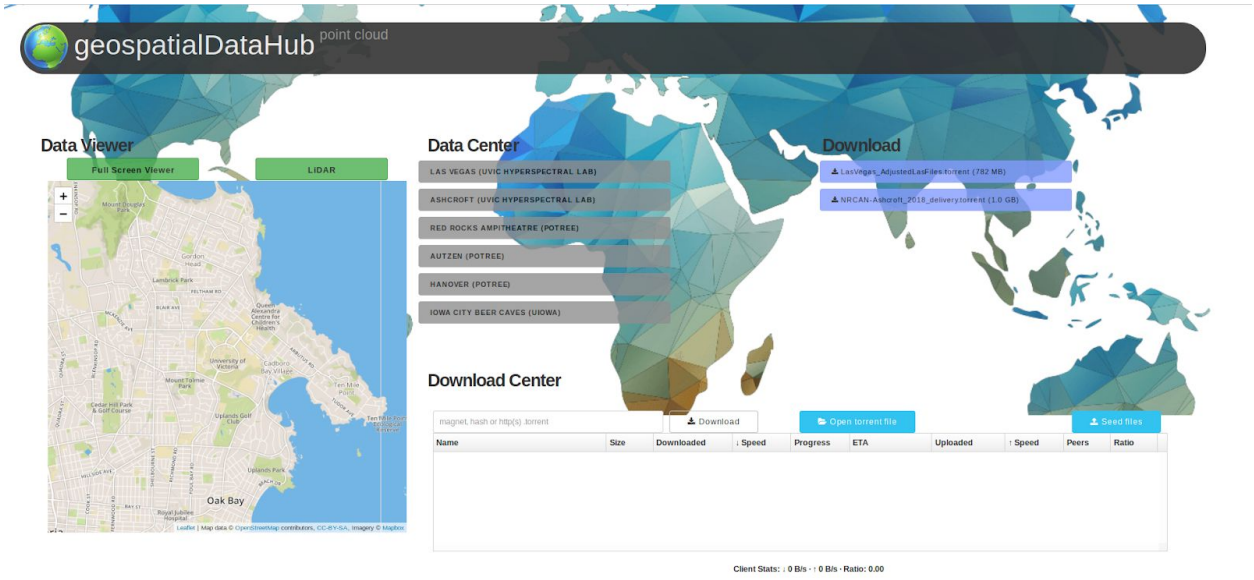
Project 2 : GeoSpatialDataStore

Build → Golang, JS/HTML/CSS Frontend, Potree, Entwine, Redis, Torrent Client, Leaflet

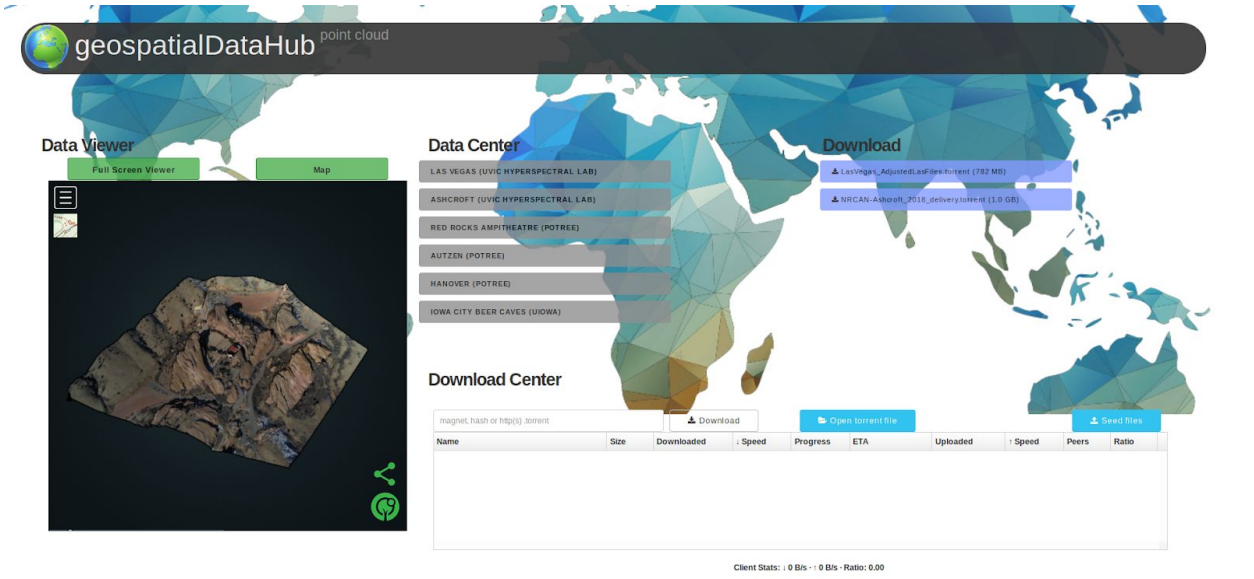


Frontend

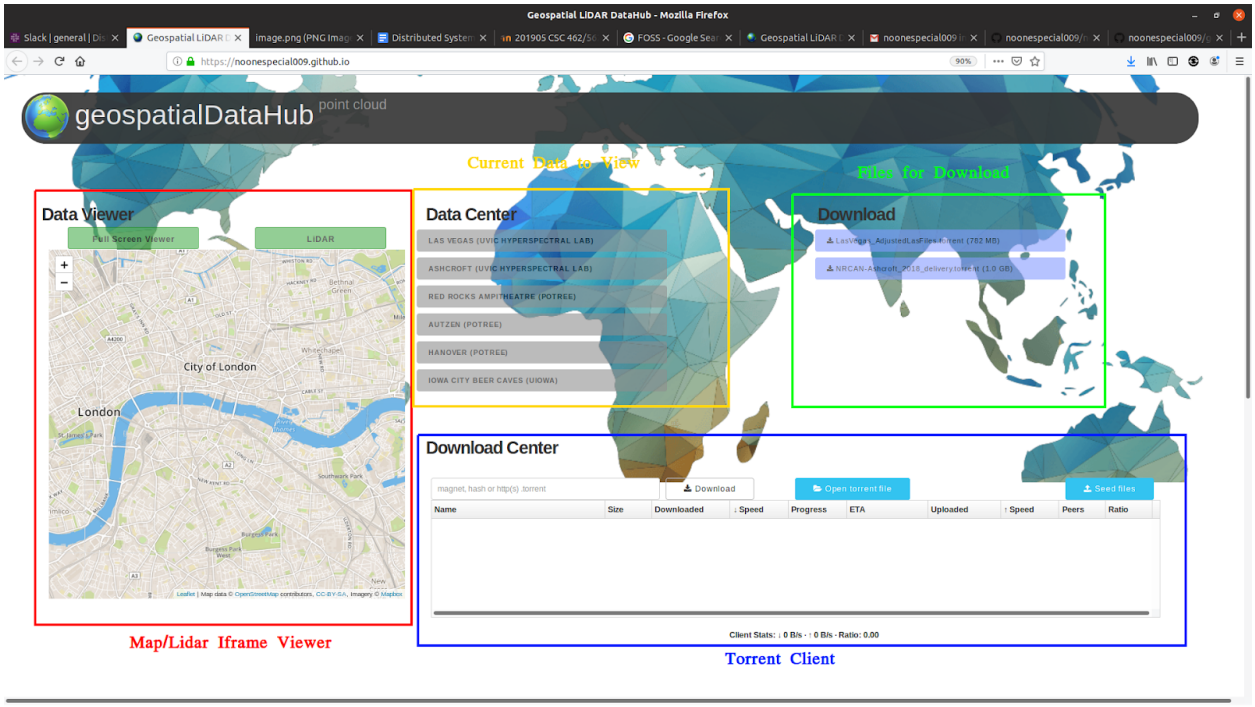
Map showing on iframe -



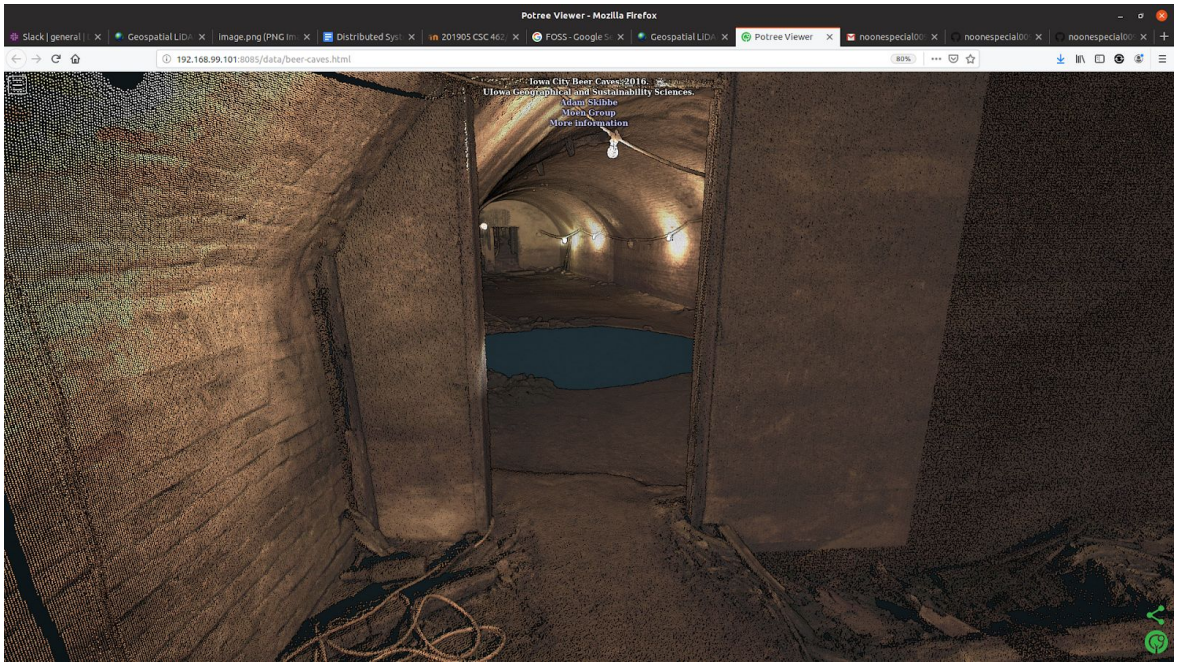
Pointcloud showing on iframe -



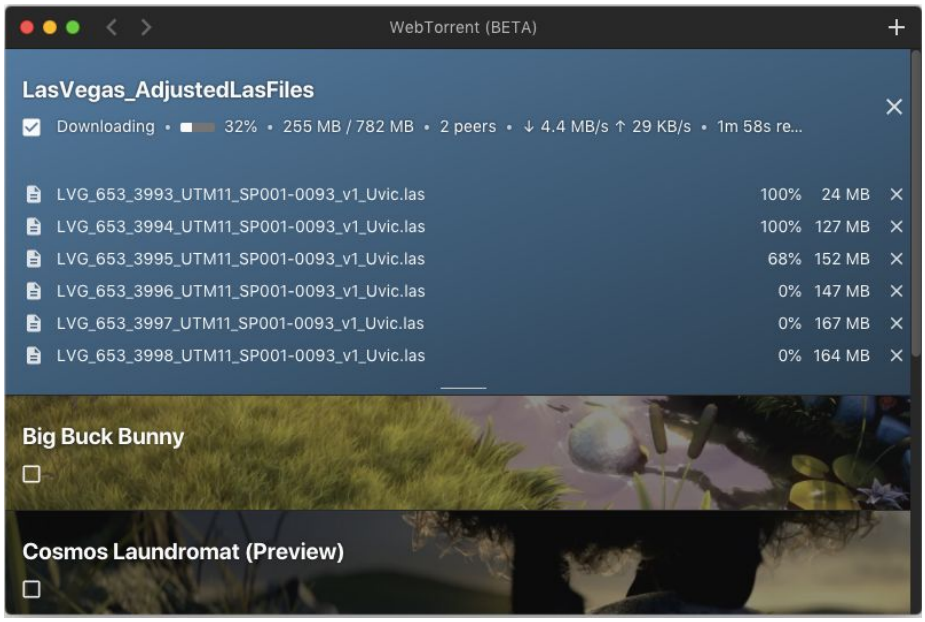
Component layout -



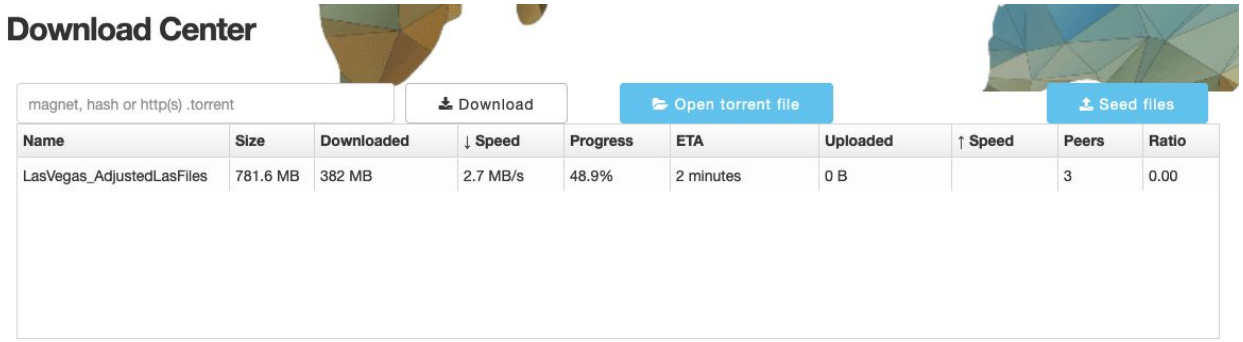
Full screen pointcloud viewer -



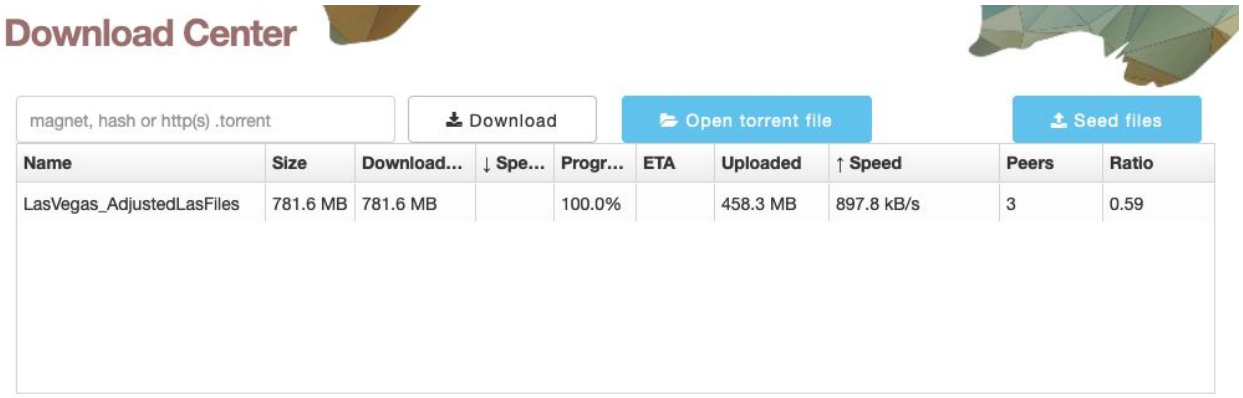
Downloading from outside of our application with Webtorrent - a standalone torrent client built using WebRTC -



Downloading a file -



Uploading an already downloaded file -



Purpose

This system was developed to solve the pain of sharing large amounts of large lidar and satellite images while also providing resources to view the images and see where they are located on a world map.

Details

We immediately began putting the various build components into docker containers and creating a docker-compose script to deploy. We used FOSS software for various components including leaflet for the map, entwine for the pointcloud viewer and a web torrent client from github. We spent many hours on front-end development in html, javascript and css. Our database is a combination of actual entwine formatted files and redis for web information. Further database development is a priority for the project going forward. Multiple databases will likely be used for the completed system. One to store actual 3d data in postgres like in our research project and then a relational database for user data and imagery metadata. We have very little progress on the map functionality at this point. It turns out that the lidar imagery we were working with has the coordinates in UTM format and converting them to latitude/longitude points that can be used with leaflet javascript is more of a process then we had time for. The torrent client is operational and so are the iframe and fullscreen entwine viewers.

We have the project distributed and deployed on docker swarm and began testing performance. The pointcloud viewer is data and cpu intensive and we were interested in rendering times with larger amounts of replicas and worker nodes. Other interests were how download and upload times would be affected by having more worker nodes.

Future Work

<u>Development</u> -

Creating a user sign in page with security and the ability to locate shared files for the torrent client is something that needs to be done. Map functionality needs to be developed. This is an in depth project that involves backend processing to retrieve the files metadata, converting UTM coordinates to lat/long points and then having javascript functions to adjust the leaflet map. Another development project is the ability to take recently downloaded files from the client, convert them to entwine files and view them in the pointcloud viewer. This involves backend processing and will definitely be a challenge but thought has been done on how it will be accomplished.

Azure -

We have researched and deployed our system to azure. However, our free student subscription did not give us the ability to create several worker nodes. This is unfortunate but if we had financial backing and the app required being scaled up we would quickly be able to deploy to a larger azure cluster that would further improve availability, performance and load balancing. This would cause us to pay for data usage, making it a non-priority at this point.

Analysis

Test #1

Goal -

To study how having multiple node affects web browser rendering time of our potree entwine pointcloud viewer.

Procedure -

- ➔ Perform each run for 10-12 seconds
- ➔ Refresh potree viewer at approx 2 seconds
- ➔ Do 5 runs
- ➔ Use firefox performance analyzer

1 node and 1 replica for each service -

- ★ Virtual Machine: 1
- ★ cpus: 0.1
- ★ Memory: 50 Mb

5 nodes and 5 replicas for each service -

- ★ Virtual Machine: 5
- ★ cpus: 0.1
- ★ Memory: 50 Mb

Set-up and Config -

1 node - 1 replica -> set-up

```
root@ootheballsoo-Latitude-E7470:/home/ootheballsoo/Programs/geospatialDataStore# docker service ls
ID                  NAME                MODE                REPLICAS            IMAGE                                                PORTS
3dlo1kqlcnuu        testApp_entwine_conda   replicated          1/1                 00theballs00/entwine_conda:latest                    *:8111->8089/tcp
satrdg56fnxq        testApp_main_server     replicated          0/1                 hyephive/geospatialdatastore_main_server:latest      *:3030->3001/tcp
t2rc18kshhfp        testApp_potree          replicated          1/1                 00theballs00/potree_entwine:latest                   *:8085->80/tcp
6fazgu619g6k        testApp_redis           replicated          0/1                 redis:latest                                         *:6379->6379/tcp
mp5frhzvoyd6        testApp_visualizer      replicated          1/1                 dockersamples/visualizer:stable                      *:8080->8080/tcp
l38p2wmnc0x1        testApp_web_server      replicated          1/1                 00theballs00/web_server:latest                       *:12346->12345/tcp
root@ootheballsoo-Latitude-E7470:/home/ootheballsoo/Programs/geospatialDataStore# docker node ls
ID                            HOSTNAME            STATUS              AVAILABILITY        MANAGER STATUS      ENGINE VERSION
laozn0pmysrmz04flukybnkld     default             Down                Active                                  19.03.1
pjknpt04e5od8tlu6lpr420kw *   myvm1               Ready               Active              Leader              19.03.1
ne42l60brqevrzpkk6sr82ptj     myvm2               Down                Active                                  19.03.1
49u2fk1ily8lyk9ypnrbtw7ffq     myvm3               Down                Active                                  19.03.1
obdxcjzwhyrvt0i8humyhzqu3     myvm4               Down                Active                                  19.03.1
root@ootheballsoo-Latitude-E7470:/home/ootheballsoo/Programs/geospatialDataStore#
```

5 node - 5 replica -> set up

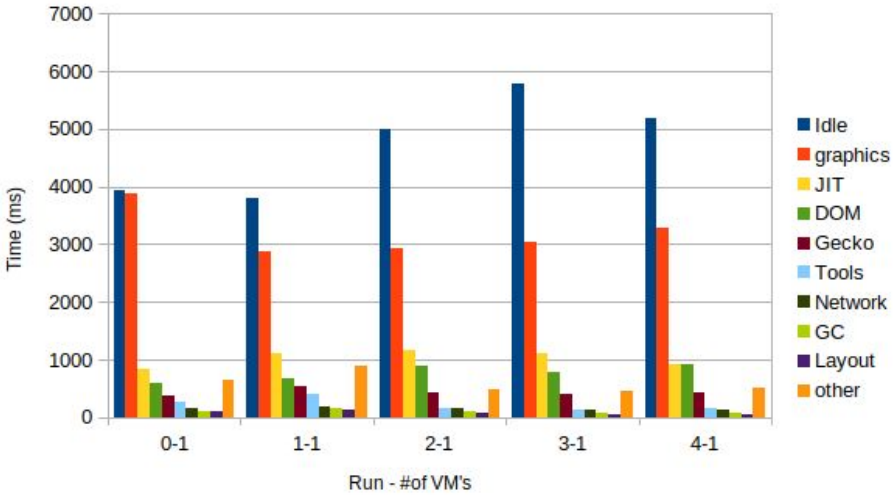
```
root@ootheballsoo-Latitude-E7470:/home/ootheballsoo/Programs/geospatialDataStore# docker-machine ls
NAME      ACTIVE   DRIVER       STATE     URL                         SWARM   DOCKER     ERRORS
default   -        virtualbox   Running   tcp://192.168.99.100:2376           v19.03.1
myvm1     *        virtualbox   Running   tcp://192.168.99.101:2376           v19.03.1
myvm2     -        virtualbox   Running   tcp://192.168.99.108:2376           v19.03.1
myvm3     -        virtualbox   Running   tcp://192.168.99.103:2376           v19.03.1
myvm4     -        virtualbox   Running   tcp://192.168.99.109:2376           v19.03.1
root@ootheballsoo-Latitude-E7470:/home/ootheballsoo/Programs/geospatialDataStore# docker node ls
ID                            HOSTNAME            STATUS              AVAILABILITY        MANAGER STATUS      ENGINE VERSION
agm0gl3vjmfshg3f9ewesdurb     default             Ready               Active                                  19.03.1
zfnp8d9gdole3r4z2gqvk4y5l *   myvm1               Ready               Active              Leader              19.03.1
jd86mhulqem6wxtvc2y0ia9ad     myvm2               Ready               Active                                  19.03.1
ry4tflilk2vnhfq4tcagmr4rh     myvm2               Down                Active                                  19.03.1
w7elxqbce8zgzndcj8kti9xuz     myvm2               Down                Active                                  19.03.1
xenge49kbgi7wpvchd3a4b68u     myvm3               Ready               Active                                  19.03.1
ef9yeu3av5k7kz1w5142uyklb     myvm4               Down                Active                                  19.03.1
b9cpu6ehseb3c3zsdj2kwk66f     myvm4               Ready               Active                                  19.03.1
root@ootheballsoo-Latitude-E7470:/home/ootheballsoo/Programs/geospatialDataStore# docker stack deploy -c docker-compose.yml testApp
Ignoring unsupported options: links, restart

Updating service testApp_potree (id: w8zpnqgxv8ysyq5cp2hpupdve)
Updating service testApp_entwine_conda (id: zptbjr089keqmfelb5tp6s3am)
Updating service testApp_visualizer (id: tw54rlc78nouvn85nl4mfl78t)
Updating service testApp_redis (id: hyl5y3xl2hqy7k1sxaf5yn8h9)
Updating service testApp_main_server (id: 9hjop5exr6p2qefypq8acwnrj)
Updating service testApp_web_server (id: muey7atmje1euhzlqtnvfw1ug)
root@ootheballsoo-Latitude-E7470:/home/ootheballsoo/Programs/geospatialDataStore# docker service ls
ID                  NAME                MODE                REPLICAS            IMAGE                                                PORTS
zptbjr089keq        testApp_entwine_conda   replicated          5/5                 00theballs00/entwine_conda:latest                    *:8111->8089/tcp
9hjop5exr6p2        testApp_main_server     replicated          0/1                 hyephive/geospatialdatastore_main_server:latest      *:3030->3001/tcp
w8zpnqgxv8ys        testApp_potree          replicated          5/5                 00theballs00/potree_entwine:latest                   *:8085->80/tcp
hyl5y3xl2hqy        testApp_redis           replicated          0/1                 redis:latest                                         *:6379->6379/tcp
tw54rlc78nou        testApp_visualizer      replicated          1/1                 dockersamples/visualizer:stable                      *:8080->8080/tcp
muey7atmje1e        testApp_web_server      replicated          5/5                 00theballs00/web_server:latest                       *:12346->12345/tcp
```

5 node - 5 replica Configuration

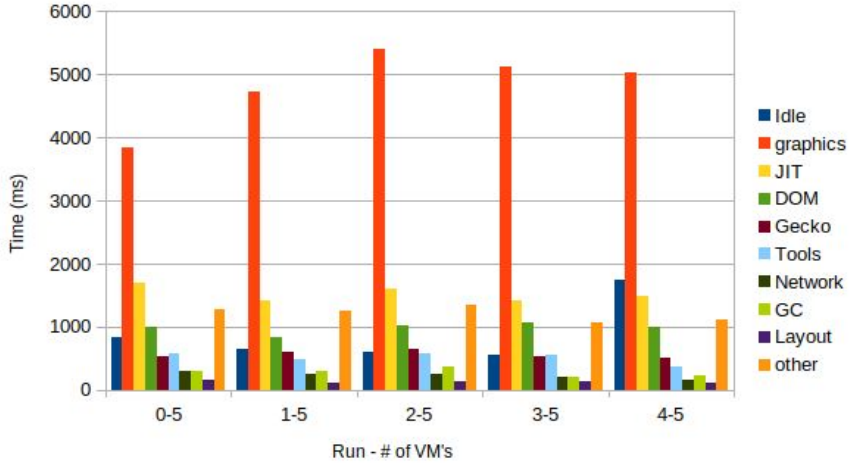
```
web_server:
  image: 00theballs00/web_server
  # build: web_server/
  deploy:
    replicas: 5
    resources:
      limits:
        cpus: "0.1"
        memory: 50M
    restart_policy:
      condition: on-failure
  # volumes:
  #   - .:/go/src/web_server
  ports:
    - 12346:12345
  networks:
    - webnet

potree:
  image: 00theballs00/potree_entwine
  # build: ./potree_entwine
  deploy:
    replicas: 5
    resources:
      limits:
        cpus: "0.1"
        memory: 50M
    restart_policy:
      condition: on-failure
  # restart: always
  ports:
    - "8085:80"
  networks:
    - webnet

entwine_conda:
  image: 00theballs00/entwine_conda
  # build: ./entwine
  deploy:
    replicas: 5
    resources:
      limits:
        cpus: "0.1"
        memory: 50M
    restart_policy:
      condition: on-failure
  # volumes:
  #   - .:/go/src/geospatialDataStore
  depends_on:
    - potree
  ports:
    - "8111:8089"
  networks:
    - webnet
```

Results -

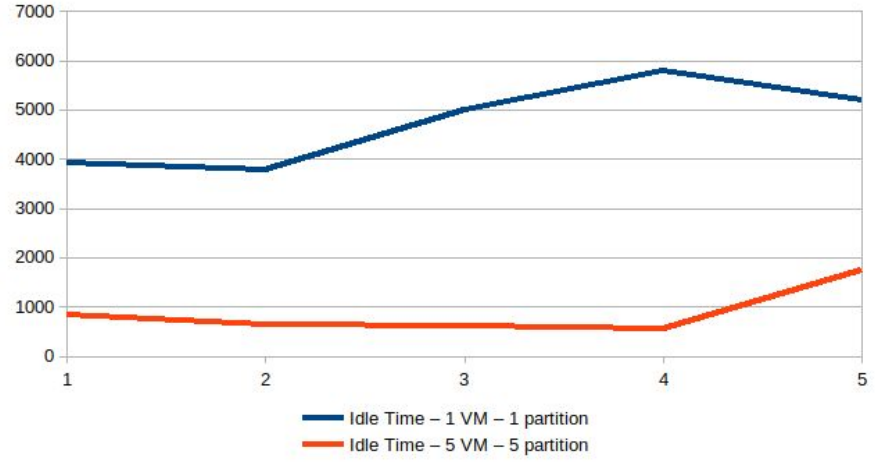
1 node:



5 nodes:



Idle time:



Conclusion -

The results showed that idle time reduced significantly with the 5 node 5 replica system. This allowed the browser to spend much more of its time rendering graphics. The difference was noticeable visually in real time to the client.

Raw Data -

Repo → https://github.com/jamshale/geo-render-test-raw

Scalability and Performance

Docker Swarm -

Docker swarm is an amazing distributed tool. It has messaging and coordination built in and allows the developer to configure how worker nodes are used. Having a ratio of replicas in favor of services that use more computing resources will ensure that these services will be served efficiently. Docker swarm then will schedule nodes into these services ensuring performance.

Azure -

We never used azure anywhere close to its full potential. Azure is a company trying to make profits and there student subscription was extremely limited. That being said, azure is tightly integrated with docker and is easy to deploy. Azure adds to performance by providing load balancing and node scheduling. However, azure is very important for scalability. The ability to add as many nodes as desired and increase computing resources is vital if the project gains any meaningful popularity.

Torrent Client -

The sharing of data files is naturally scalable due to being a peer to peer network. The more peers start using the site and sharing files the higher the bandwidth and the faster the possible download speeds.

Pointcloud Viewer -

Adding more processing nodes to supply entwine layers needed for client web browser rendering can be done by having a higher ratio of entwine replicas to other services. When

there is a lot of client traffic using the pointcloud viewer our system uses docker swarm to shift computing resources to serving entwine files. This allows the clients web browser to spend less time idling and more time rendering graphics.

Reliability

Docker Swarm -

Docker swarm is very useful for reliability. Deploy docker compose settings can be declared to make sure the system remains available. The only way it will go down is if all nodes fail. If the manager fails then it does need to be re-initialized manually or another node needs to be made manager.

Azure -

Azure dramatically improves reliability by starting up new nodes when other nodes fail. Having a massive global system significantly improves uptime at 99.995%.
(https://azure.microsoft.com/en-ca/blog/advancing-microsoft-azure-reliability/)

Conclusion

This project involved a lot of research in a lot of different areas. We learned about geospatial data, how to store it in a database and how to view it in a web browser. We learned how to distribute and scale a software system using tools like docker and azure. The things we learned and the skills we gained during this project will be valuable to our careers in a world of distributed systems.