Big Data Project Proposal

Axel Alvarez

I. Project Overview

A. Project Title

Severe Weather Infrastructure Impact Analysis System

B. Domain

Geospatial Analytics and Public Safety / Critical Infrastructure Protection

C. Problem Statement

Extreme weather events cause billions of dollars in infrastructure damage annually, yet correlating meteorological data with ground-level impact remains a significant computational challenge. The core difficulty lies in the "M x N" complexity of mapping thousands of high-velocity "Point" events (such as traffic accidents or power outages) against dynamic, moving "Polygon" geometries (weather storm cells) in near real-time. Traditional relational databases struggle to handle these spatial joins at scale, leading to latency that renders the data useless for immediate decision-making.

D. Scope

This project will implement a distributed Big Data pipeline designed to ingest, process, and correlate real-time weather alerts with infrastructure disruption data.

In Scope:

- Ingestion of live weather alert streams (NOAA) and traffic incident feeds (DOT).
- Distributed spatial joining of point-in-polygon data using Apache Spark and Apache Sedona.
- Aggregation of impact metrics by geographic region (e.g., "Accidents per County per Storm Hour").
- Persistent storage of processed data in a PostGIS data warehouse.

Out of Scope:

- Prediction of future weather events (Meteorological modeling).
- Real-time routing or navigation adjustments for end-users.
- Analysis of data outside the Southeastern United States (to manage resource constraints during the prototype phase).

II. System Description

A. Data Sources

The proposed system aggregates data from three heterogeneous sources to analyze weather impacts on infrastructure:

- **NOAA National Weather Service (NWS) API:** A public REST API providing real-time severe weather alerts. The data includes CAP (Common Alerting Protocol) messages and complex polygon geometries defining storm cells, tornado paths, and flood zones.
- **Department of Transportation (DOT) Traffic Feeds:** State-level data streams (e.g., GA511, Bing Maps Traffic) reporting infrastructure incidents. These feeds provide "Point" data (latitude/longitude) for accidents, road closures, and hazardous conditions.
- **PowerOutage.US (Web Scraping):** Semi-structured HTML data scraped from public utility maps. This serves as a validation layer, providing aggregate outage statistics by county to confirm the severity of weather events.

B. Data Characteristics

- **Velocity:** The system must handle high-velocity ingestion. Weather alerts are updated minutely, and traffic feeds are continuous. The pipeline must synchronize these asynchronous streams to correlate a crash at T_0 with a storm active at T_0.
- **Variety:**
  - *Structured:* Traffic incidents (JSON/XML).
  - *Semi-Structured:* Weather Alerts (GeoJSON/CAP).
  - *Unstructured:* Outage reports (HTML text parsing).
- **Volume:** While individual text alerts are small, the aggregate volume of traffic data points combined with the computational overhead of spatial indexing requires distributed storage and processing, particularly when scaling to a national level.

C. Stack Layers

This project engages three primary layers of the Big Data stack:

1) **Storage Layer (Object Storage):** Raw data is ingested into an S3-compatible object store (e.g., AWS S3 or MinIO) using a "Data Lake" architecture. Data moves from "Bronze" (Raw JSON) to "Silver" (Cleaned/Parquet) buckets.
2) **Processing Layer (Spatial Parallelism):** The core transformation engine is **Apache Spark** extended with **Apache Sedona**. This layer handles the "M x N" complexity of spatial joins. By utilizing *Quad-Tree Spatial Partitioning*, the system distributes distinct geographic regions across worker nodes, allowing for parallel execution of Point-in-Polygon operations.
3) **Data Store Layer (Serving):** Aggregated insights are loaded into **PostgreSQL** with the **PostGIS** extension. This allows for low-latency querying of the final dataset (e.g., "Select all accidents within Tornado Alley in the last hour") without triggering full re-processing.

D. Assumptions

- All input data sources maintain accurate UTC timestamps
- The polygon geometries provided by the NWS API are valid and closed
- The system focuses initially on the Southeastern United States to manage API rate limits during development.

III. Implementation Approach

A. Technology Choices and Justification

The system architecture is designed to handle the high velocity of weather alerts and the volume of traffic data through a loosely coupled, distributed stack:

- **Ingestion Layer (Python & S3):** Lightweight Python scripts (utilizing `requests` and `boto3`) will poll the NOAA and DOT APIs at 60-second intervals. Data is immediately persisted to **AWS S3** (or a local MinIO instance) in its raw JSON format (Bronze Layer). S3 is chosen for its durability and ability to handle unstructured data blobs without schema enforcement.
- **Processing Layer (Apache Spark + Sedona):** The core ETL and spatial joining will be performed by **Apache Spark** (PySpark). To handle the geospatial complexity, the **Apache Sedona** (formerly GeoSpark) library will be used. Sedona extends Spark's RDDs to support "Spatial RDDs" (SRDDs), enabling distributed spatial joins that are orders of magnitude faster than standard SQL Cartesian products.
- **Serving Layer (PostGIS):** Processed and aggregated data is loaded into **PostgreSQL** extended with **PostGIS**. This relational store allows for complex analytical queries (e.g., "Find all traffic accidents within 5km of a Tornado Warning polygon") to be served to a dashboard with sub-second latency.

B. Processing Model

The system will implement a **Micro-Batch Processing** model using Spark Structured Streaming.

- **Why Micro-Batch?** Real-time traffic events arrive continuously, but weather alerts update in discrete windows (minutes). A purely streaming architecture is complex to join with static polygons, while a daily batch is too slow for public safety. Micro-batching (e.g., every 2 minutes) provides the optimal balance between low latency and high throughput.

C. Scalability Plan

The primary scalability challenge is the $O(N \times M)$ complexity of joining N traffic points with M weather polygons. The system addresses this through Spatial Partitioning:

- **Quad-Tree Partitioning:** Apache Sedona will be configured to build a global Quad-Tree index across the cluster. This technique recursively subdivides the geographic map into grid cells based on data density.
- **Data Locality:** Traffic points and weather polygons located in the same geographic grid cell will be shuffled to the same Spark executor. This ensures that spatial joins occur locally in memory without expensive network shuffles across the entire cluster.
- **Horizontal Scaling:** As the geographic scope expands (e.g., from the Southeast to the entire Continental US), additional worker nodes can be added to the Spark cluster to handle the increased number of grid partitions.

D. Performance Metrics

Success will be measured against the following key performance indicators:

- **End-to-End Latency:** The time elapsed between a traffic incident appearing in the raw API feed and its availability in the PostGIS serving layer (Target: < 3 minutes).
- **Throughput:** The number of spatial join operations performed per second during peak weather events.
- **Data Freshness:** The percentage of active weather alerts successfully synchronized with the traffic stream.

IV. Literature Review

A. Core Distributed Systems Concepts

The architecture of this project is grounded in research regarding distributed file systems and parallel processing models.

1) MapReduce and Distributed Processing: Dean and Ghemawat introduced the **MapReduce** programming model to process vast datasets by decomposing tasks into a parallel "Map" phase and an aggregating "Reduce" phase [1].

- **Relevance:** This project adapts the MapReduce paradigm for geospatial data. The "Map" phase filters raw weather alerts and traffic incidents by timestamp, while the "Reduce" phase aggregates these incidents spatially (e.g., grouping all accidents within a specific county).

2) Spatial Resilient Distributed Datasets (SRDDs): While standard Spark RDDs handle text well, they are inefficient for spatial queries. Yu et al. proposed **GeoSpark** (now Apache Sedona), which extends Spark with Spatial RDDs (SRDDs) and distributed spatial indices [2].

- **Application:** I will utilize the *Spatial Join* operator defined in this framework to solve the "Point-in-Polygon" problem across the cluster, using their proposed co-location capability to minimize network shuffling.

B. Spatial Indexing Algorithms

Efficiently querying multidimensional data requires specialized data structures beyond standard B-Trees.

1) R-Tree Indexing: Guttman introduced the **R-Tree**, a dynamic index structure that represents spatial objects by their Minimum Bounding Rectangles (MBRs) [3].

- **Adaptation:** SWIIAS will implement R-Tree indexing within the PostGIS serving layer. This ensures that downstream queries (e.g., "Find active storm cells near Atlanta") execute in logarithmic time $O(\log_M n)$ rather than requiring a full table scan of all active polygons.

C. Data Standards and Protocols

1) Common Alerting Protocol (CAP): The National Weather Service (NWS) utilizes the **Common Alerting Protocol (CAP)**, an XML-based information standard for exchanging emergency alerts [4].

- **Relevance:** This project relies on the CAP v1.2 specification to parse the complex polygon geometries from the NWS API. The hierarchical nature of CAP messages (Alert $\rightarrow$ Info $\rightarrow$ Area) dictates the schema design of the Bronze ingestion layer.

D. Related Work in Intelligent Transportation

Research by Silva et al. demonstrates that Big Data architectures can successfully detect traffic congestion patterns by correlating heterogeneous sensor data [5].

- **Distinction:** While Silva focuses on social media sentiment to validate traffic jams, this project extends that approach by introducing meteorological data as a primary causal factor, requiring a more complex spatial join between moving storm cells and static road segments.

REFERENCES

[1] J. Dean and S. Ghemawat, "MapReduce: simplified data processing on large clusters," *Communications of the ACM*, vol. 51, no. 1, pp. 107–113, 2008.

[2] J. Yu, J. Wu, and M. Sarwat, "GeoSpark: a cluster computing framework for processing large-scale spatial data," in *Proceedings of the 23rd SIGSPATIAL International Conference on Advances in Geographic Information Systems*, 2015, pp. 1–4.

[3] A. Guttman, "R-trees: a dynamic index structure for spatial searching," in *Proceedings of the 1984 ACM SIGMOD International Conference on Management of Data*, 1984, pp. 47–57.

[4] National Weather Service, "NWS Common Alerting Protocol (CAP) v1.2 Documentation," NOAA, Technical Report, 2024. [Online]. Available: https://alerts.weather.gov/

[5] F. Silva et al., "A Big Data Architecture for Near Real-time Traffic Analytics," in *IEEE International Conference on Big Data*, 2017, pp. 230–238.

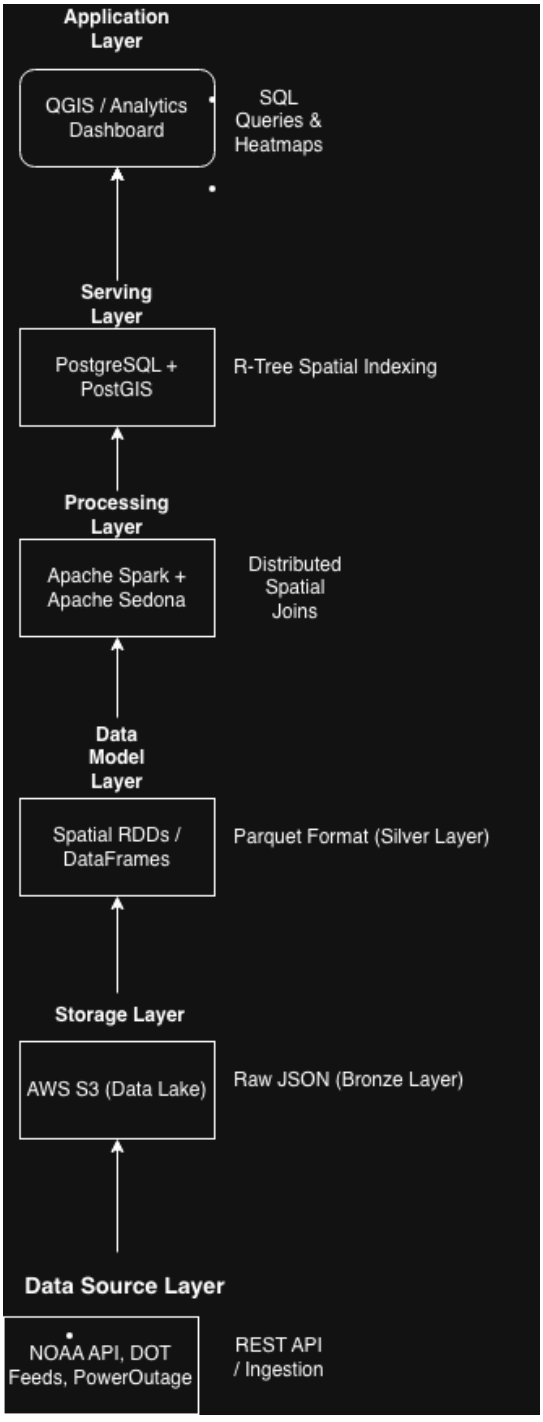

Fig. 1. SWIIAS Stack Architecture Diagram