Multi-Source Climate Trend Data Integration Pipeline

Joshua Young
CS 4265 – Big Data Analytics
Kennesaw State University

February 5, 2026

Contents

1 Project Overview **2**
- 1.1 Domain . . . 2
- 1.2 Problem Statement . . . 2
- 1.3 Scope . . . 2

2 System Description **2**
- 2.1 Data Sources . . . 2
- 2.2 Data Characteristics . . . 3
  - 2.2.1 Volume . . . 3
  - 2.2.2 Variety . . . 3
  - 2.2.3 Velocity . . . 3
- 2.3 Big Data Stack Layers . . . 3
- 2.4 Assumptions . . . 3

3 Implementation Approach **4**
- 3.1 Technology Choices . . . 4
- 3.2 Processing Model . . . 4
- 3.3 Scalability Plan . . . 5
- 3.4 Evaluation Metrics . . . 5

4 Literature Review **5**
- 4.1 Core Concepts and Technologies . . . 5
  - 4.1.1 Source 1: The Google File System (GFS) . . . 5
  - 4.1.2 Source 2: Resilient Distributed Datasets: A Fault-Tolerant Abstraction for In-Memory Cluster Computing . . . 6
  - 4.1.3 Source 3: Dremel: Interactive Analysis of Web-Scale Datasets . . . 6
  - 4.1.4 Source 4: The Log-Structured Merge-Tree (LSM-Tree) . . . 7
  - 4.1.5 Source 5: CAP Twelve Years Later: How the "Rules" Have Changed 7

5 System Architecture **8**
- 5.1 Stack Architecture Diagram . . . 8
- 5.2 Data Flow Diagram . . . 9

6 References **9**

1 Project Overview

1.1 Domain

This project focuses on long-term trends in temperature and precipitation in the United States. Climate trends are useful for planning, risk assessment, and public awareness. The main issue is that the relevant data is provided from multiple sources and formats, which require integration and normalization before it can be queried consistently.

1.2 Problem Statement

The system will answer questions about long-term climate trends across 2,000 weather stations in the United States over a 10-year period. The main challenges include ingesting heterogeneous sources, handling authentication and rate limits, discovering and tracking schema differences, and integrating data into a unified schema that supports efficient partitioned querying.

1.3 Scope

The scope of the system will focus on ingesting from 3 heterogeneous sources, which are the NOAA Climate Data Online, the National Weather Service, and Open-Meteo. The system will handle authentication, pagination, and rate limiting. The system will support ingestion with incremental updates, unified daily observation tables, and derived annual and monthly trend metrics accessible through distributed queries and a lightweight output interface.

The system will not include machine learning-based forecasting, global ingestion in real-time, or production deployment infrastructure. The system will be designed to handle a vast number of weather stations. For the scope of this project, it will focus on a smaller subset of 2,000 stations so that the system can remain within API rate limits.

2 System Description

2.1 Data Sources

- **NOAA Climate Data Online (CDO):** Historical daily station observations that include temperature, precipitation, and station metadata. Accessed via authenticated REST API with rate limits and pagination.
- **National Weather Service API:** Operational weather data, such as forecasts and severe weather alerts, in JSON format. Public access with enforced rate limiting and differing schema structure.
- **Open-Meteo API:** Historical and forecast gridded climate data providing temperature, precipitation, and atmospheric variables with alternate field naming and units.

2.2 Data Characteristics

2.2.1 Volume

The data will ingest 10 years worth of daily climate observations across the United States from 2,000 stations. This will give a total of 21 million records that will need to be ingested and processed.

2.2.2 Variety

- **NOAA CDO:** Station-based structured time-series data in JSON format with flat measurement records per station and date, using coded climate variables (e.g., TMAX, PRCP) and separate metadata endpoints.
- **National Weather Service API:** Semi-structured operational weather data delivered primarily in GeoJSON with embedded geospatial geometry and nested properties, with optional JSON-LD and multiple XML-based encodings (DWML, OXML, CAP, ATOM). Data represents time-range forecasts and event-based alerts rather than flat daily records.
- **Open-Meteo API:** Gridded climate data in JSON format organized as array-based time series by latitude and longitude, with alternate variable naming conventions and metric units.

2.2.3 Velocity

Data ingestion will follow a batch-oriented processing model with incremental updates. The large historical datasets are retrieved during the initial ingestion phase, while daily climate observations are periodically collected to continuously provide current data.

2.3 Big Data Stack Layers

This project uses three layers of the big data stack: the storage layer, the processing layer, and the data store layer. For the storage layer, this project makes use of the Hadoop Distributed File System (HDFS), together with Parquet files, to handle large amounts of climate-related data in a distributed fashion. For the processing layer, this project makes use of Apache Spark to perform data cleansing, integration, and trend aggregation in a distributed fashion. For the data store layer, this project makes use of MongoDB to store location-related metadata and trend summaries.

2.4 Assumptions

- The geographic scope of the system is limited to the United States.
- The analysis period is limited to the last 10 years (2016–present).
- Data ingestion targets a representative set of stations near major cities (one or more per state) rather than all available stations nationwide.

- The pipeline operates in batch mode with incremental updates; continuous real-time ingestion is not required.
- Observations are normalized to store one row per day per location. It will not store hourly or minute-by-minute for everything.
- Units will be converted for all temperatures, rainfall, and timestamps into one consistent format.
- Missing or incomplete observations are handled by leaving them blank or ignoring them in calculations.

3 Implementation Approach

3.1 Technology Choices

- **Storage: HDFS** (Hadoop Distributed File System) configured in a single-node cluster to manage data blocks and replication
- **Syntax: Parquet** is used with its columnar structure to reduce disk I/O by allowing the processing engine to retrieve specified columns.
- **Data Model: Spark DataFrames** is used to ensure schema validation and normalization across each data source.
- **Data Store: MongoDB** with the **WiredTiger** engine is used to handle high-frequency writes of processed trends and support efficient query retrieval.
- **Processing: Apache Spark** will act as the parallel execution engine, which executes Directed Acyclic Graphs (DAG) to perform core distributed operations
- **Querying: Spark SQL** will provide a declarative language for high-level data analysis and is used to run queries against the final processed dataset to generate the resulting climate trend reports.

3.2 Processing Model

The system will utilize a batch processing model along with incremental updates. A batch process loads historical climate data from 2016 up until the present in bulk. This data is stored in partitioned Parquet format in HDFS. During subsequent runs of the pipeline, only new data is added, normalized using Apache Spark, and appended to the consolidated daily observation data set. After every batch process, Spark updates the monthly and yearly trend data set, which is later stored in MongoDB.

3.3 Scalability Plan

Scalability will be ensured through partitioned data storage and parallel processing. The Parquet files will be partitioned based on the year which allows Spark to do partition pruning and will be able to read the files present in the specific directory for the query rather than reading the whole 21 million records. This will help to reduce the time taken for query execution as the data volume increases.

3.4 Evaluation Metrics

System performance and data integrity will be evaluated using the following quantitative metrics to assess the efficiency of the distributed architecture:

- **Ingestion Performance:** Evaluated based on the rate of successfully ingested data per second and the number of API requests made per minute.
- **Pipeline Runtime:** Total time required for the execution of both the initial ingestion of historical data and subsequent incremental updates.
- **Storage Utilization:** Overview of the aggregate data stored in HDFS, including 21.9 million records, and the level of compression obtained through Parquet and Snappy mechanisms.
- **Query Performance:** Time needed to execute typical analytical queries such as computing 10-year climate averages.
- **Data Quality:** Evaluated by tracking missing or incomplete values across key climate variables and across each data source.
- **System Reliability:** Evaluated based on API failure rates, retry attempts, and the system's ability to recover from interruptions.

4 Literature Review

4.1 Core Concepts and Technologies

4.1.1 Source 1: The Google File System (GFS)

- **Concept:** This concept focuses on storage management, and GFS does this by dividing files into fixed-size chunks. A master node assigns each chunk an ID and monitors its position, while chunk servers save the bits on local disks. It replicates each chunk to achieve fault tolerance, which allows data to be retrieved even if the storage disk fails.
- **Relevance:** This is relevant to the project because it must be able to process nearly 21.9 million records. A distributed storage system is needed to handle this volume without loading the entire dataset into memory at once. A block-based storage approach allows the system to organize data into manageable pieces which is necessary into order to quickly retrieve data for trend analysis

- **Adaptation:** The system will implement HDFS to emulate the GFS architecture. This allows the system to partition the large dataset into manageable blocks by using a replication factor to ensure data integrity.

4.1.2 Source 2: Resilient Distributed Datasets: A Fault-Tolerant Abstraction for In-Memory Cluster Computing

- **Concept:** This source focuses on Apache Spark and the concept of In-Memory Distributed Computation. Unlike previous disk-based models, Spark keeps data in the system's RAM so that it can avoid slow I/O operations. It also incorporates Lazy Evaluation, where transformations are recorded in a Directed Acyclic Graph (DAG) and are only executed when an 'action' is triggered. This lets the system optimize the whole processing plan before any data is moved.

- **Relevance:** This architecture is essential for processing the large dataset for this project. Single-node processing models are limited by sequential memory allocation, which can create significant overhead and potentially have issues with running out of memory. Spark is able to distribute the memory load and avoid bottlenecks in comparison to single-node models.

- **Adaptation:** This project adopts Spark's in-memory distributed processing model by using Spark DataFrame partitioned by time and location. This allows data cleaning and trend calculations to run in parallel across the cluster. Tasks such as validating records, standardizing fields, and converting daily data into monthly and yearly summaries are executed by using Spark's lazy evaluation pipeline.

4.1.3 Source 3: Dremel: Interactive Analysis of Web-Scale Datasets

- **Concept:** This source introduces the concept of columnar storage, where data is organized by column rather than by row. In traditional row-based formats such as CSV and JSON, the system has to read entire records even if only a few fields are needed for the query. This can lead to queries taking excessive amounts of time to return results. Columnar storage allows the system to read only the required columns, which allows for it operate very quickly.

- **Relevance:** This is relevant because the project involves moving roughly 21 million records, which is a lot to move from the disk to the CPU. Climate trends also involve analytical data, which means that queries will not involve a single specific record but rather averages for a specific variable over a long period. Using Parquet, the system will be much faster because it will shorten the time required to read through all of the data.

- **Adaptation:** The system will configure Spark to use this format by allowing Spark to check the metadata of a Parquet file before opening it. Snappy will also be utilized to compress the data to balance disk space and performance.

4.1.4 Source 4: The Log-Structured Merge-Tree (LSM-Tree)

- **Concept:** This paper proposes an architecture that contains a memory-resident part (C0) and a disk-resident part (C1). Instead of updating the data directly, new data is appended to the C0 part. However, once the C0 part reaches a specified size, the data is migrated to the C1 part via a rolling merge.

- **Relevance:** This is relevant because the rolling merge process is crucial for systems that handle high-volume ingestion. By performing sequential merges rather than random disk writes, the system can maintain high performance even as the dataset grows larger.

- **Adaptation:** The system will adapt this process by using MongoDB's default WiredTiger storage engine, which implements the concepts described in the paper. This will be configured to handle the high-volume daily increments of climate data, which ensures that the ingestion process remains efficient as the dataset grows.

4.1.5 Source 5: CAP Twelve Years Later: How the "Rules" Have Changed

- **Concept:** This article discusses the CAP theorem and how systems can move away from the idea of only choosing two components of CAP. Talk about how designers can achieve the balance of all three components by focusing on handling partitions. The strategy mentioned in the article is to detect partitions, move to a partition mode that limits some operations, and then initiate a recovery process to restore consistency.

- **Relevance:** The relevance of this concept to the project is that it involves an analysis of climate trends and calculations. If there is a connection problem while handling large volumes of data, computations may go on with partial information. Also if one of the sources of information is updated every day while another is updated every hour, it has to decide which information to display.

- **Adaptation:** This will be adapted in the project by strictly applying Write Concern in MongoDB. This will ensure that information is safe if there is a problem with the connection during processing. Spark will be used with checkpoints to ensure fault tolerance. This will ensure that the current state of the process is saved at intervals, thereby allowing recovery of information. Partition tolerance will also be included so that it checks if a record already exists before it is inserted.

5 System Architecture

5.1 Stack Architecture Diagram

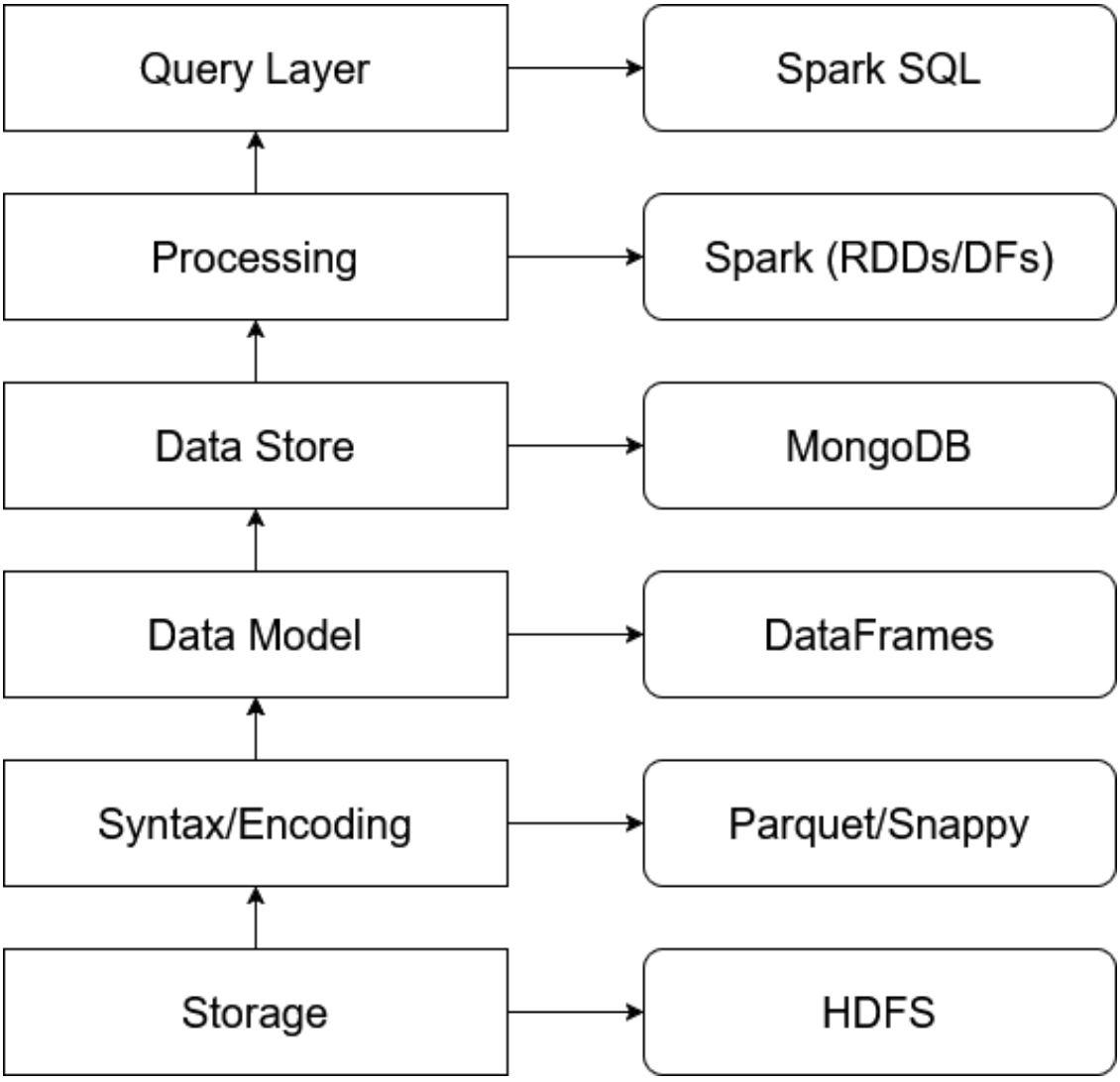

Figure 1: Stack architecture diagram of the Big Data system

5.2 Data Flow Diagram

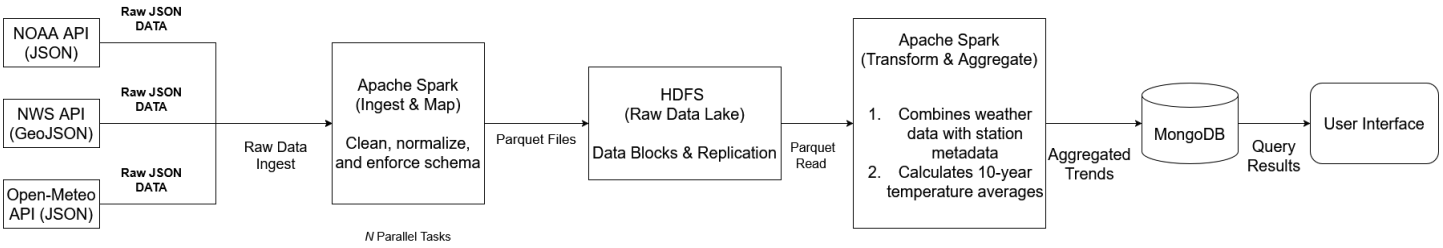

Figure 2: Data flow and processing pipeline

6 References

References

[1] Apache Software Foundation. (n.d.). *Apache Hadoop HDFS Architecture*. https://hadoop.apache.org/docs/stable/hadoop-project-dist/hadoop-hdfs/HdfsDesign.html

[2] Apache Spark. (2019). *Apache Spark Documentation*. https://spark.apache.org/docs/latest/

[3] Brewer, E. (2012). CAP Twelve Years Later: How the "Rules" Have Changed. *InfoQ*. https://www.infoq.com/articles/cap-twelve-years-later-how-the-rules-have-changed/

[4] Ghemawat, S., Gobioff, H., & Leung, S.-T. (2003). The Google File System. *Proceedings of the Nineteenth ACM Symposium on Operating Systems Principles*. https://doi.org/10.1145/945449.945450

[5] Melnik, S., Gubarev, A., Long, J., Romer, G., Shivakumar, S., Tolton, M., & Vassilakis, T. (2010). *Dremel: Interactive Analysis of Web-Scale Datasets*. https://static.googleusercontent.com/media/research.google.com/en//pubs/archive/36632.pdf

[6] MongoDB. (n.d.). *WiredTiger Storage Engine*. https://www.mongodb.com/docs/manual/core/wiredtiger/

[7] National Oceanic and Atmospheric Administration. (2026). *Climate Data Online Web Services API Documentation*. https://www.ncei.noaa.gov/cdo-web/webservices/v2

[8] National Weather Service. (n.d.). *API Web Service Documentation*. https://www.weather.gov/documentation/services-web-api

[9] O'Neil, P., Cheng, E., Gawlick, D., & O'Neil, E. (1996). *The Log-Structured Merge-Tree (LSM-Tree).* https://www.cs.umb.edu/~poneil/lsmtree.pdf

[10] Open-Meteo. (2025). *Historical Weather API Documentation.* https://open-meteo.com/en/docs/historical-weather-api

[11] Zaharia, M., Chowdhury, M., Das, T., Dave, A., Ma, J., McCauley, M., Franklin, M., Shenker, S., & Stoica, I. (2012). *Resilient Distributed Datasets: A Fault-Tolerant Abstraction for In-Memory Cluster Computing.* https://www.usenix.org/system/files/conference/nsdi12/nsdi12-final138.pdf