Big Data Pipeline for Public Court Document Analytics

By: Josue Palacios

Repository: https://github.com/josuepalacioss/Public-court-data-pipeline

Project Proposal

Project Overview

Domain: Public court data analytics

This project focuses on processing the structured metadata contained within court cases and legal documents. These public court datasets are readily available through APIs and bulk archives, but they are often fragmented across institutions. By ingesting and analyzing the document metadata we can extract courts, dates, case types, document counts, and completeness indicators. This domain is motivated by the need for scalable infrastructure to support legal research and analysis of judicial activity using large public datasets.

Problem Statement

Public court data exists on a large scale but is difficult to analyze efficiently due to the sheer volume, heterogeneity, and different access constraints. Federal and state court datasets are published by different organizations, through different access methods. The ones tackled here include REST APIs provided by CourtListener and bulk archives provided by the Caselaw Access Project by Harvard Law School. As a result, answering basic questions will require great effort.

This project aims to address the following questions:

- How does the volume of court cases and documents change over time across jurisdictions?
- How do federal and state court datasets differ in coverage and completeness?
- What trends exist in case counts, document counts, and court activity over time?
- How complete are metadata fields across sources?

At scale, these questions introduce several challenges:

- Data volume by the hundreds of thousands to over a million records and the resulting space.
- Heterogeneous ingestion through combining incremental API based data with large bulk datasets.
- Schema variability that offers differing field structures and naming across sources.

Scope

This project focuses on the design and implementation of a distributed, batch-oriented Big Data pipeline for analyzing structured metadata derived from public court documents.

In Scope

Batch ingestion of heterogeneous public court datasets:

- Federal court metadata from the CourtListener REST API.
- State court metadata from the Harvard Caselaw Access Project (Georgia) bulk archives.

Schema normalization and validation to unify heterogeneous JSON sources into a shared Parquet-based data model.

Metadata analytics

- Case and document volume trends over time.
- Court and jurisdiction activity summaries.
- Completeness and coverage statistics for key metadata fields.

Distributed query execution using Spark SQL with partitioned Parquet datasets.

Local execution with a cloud-ready design that supports scaling via additional partitions and executors.

Out of Scope

- Parsing, indexing, or analyzing the contents of court case PDF documents.
- Natural language processing, machine learning, or data mining.
- Legal interpretation or reasoning over court opinions.
- Real time or streaming data ingestion.

System Description

Data Sources

Federal Courts – CourtListener

- Accessed via REST API
- Bounded subset of federal courts (via district/circuits)
- Records from 1600s - Present

State Courts – Caselaw Access Project (Georgia)

- Ingested via bulk data files
- Court documents 1808 - 2018

Data Characteristics:

Volume

At proposal scope, the system is designed to process approximately 300,000 to 700,000 metadata records, with raw input sizes roughly tens of gigabytes of JSON. After normalization and conversion to partitioned Parquet format, storage requirements are significantly reduced, enabling efficient distributed processing. The architecture will eventually support scaling to roughly millions of records through additional partitions and compute resources.

Variety

The ingested datasets differ in structure and representation. CourtListener provides nested JSON metadata retrieved via an API, while the Caselaw Access Project supplies bulk JSON files with distinct schemas and field conventions. These differences require explicit schema definition and normalization before cross-source analytics can be performed.

Velocity

The system follows a batch-oriented model. CourtListener data is ingested as monthly API snapshots, while Caselaw Access Project data is ingested as large, historical batches. The pipeline is designed for repeatable batch execution rather than streaming ingestion.

Stack Layers

Storage Layer - Distributed file system where files split into blocks, stored with replication

Syntax Layer - Data formats and serialization: JSON -¿ Parquet

Data Models Layer – Schemas and validation to turn JSON into query tables

Processing Layer – Parallel execution through DAGs

Querying Layer – Distributed query execution and partitioning

Assumptions

Storage Assumptions

The system assumes a distributed storage model, HDFS architectures, where large datasets are divided into blocks, replicated for fault tolerance, and stored immutably. Durability and replication are provided by the storage layer rather than by application logic, allowing the pipeline to tolerate node or task failures without data loss.

Processing Assumptions

The pipeline assumes a batch distributed processing model based on MapReduce, where large jobs are split into parallel tasks that perform mapping, shuffling, and reduction over partitions of the data.

Querying Assumptions

Analytical queries are assumed to execute over partitioned datasets using distributed query execution. Performance depends on partitioning strategies and query planning that minimize unnecessary data scans and data movement.

Implementation Approach

Technology Choices

Storage Layer

- HDFS
- Potentially AWS S3 for cloud scaling

Syntax Layer

- JSON for raw ingestion of API snapshots and bulk files
- Parquet for columnar storage for efficient scans

Data Models Layer

DataFrames are used to enforce consistent data models across sources, mapping heterogeneous JSON

Processing Layer

MapReduce batch processing, where metadata records are mapped, shuffled by key, and reduced into aggregated analytics

Querying Layer

Spark SQL for analytics queries over partitioned curated Parquet

Processing Model

The system uses a batch processing model. CourtListener data is ingested through periodic REST API snapshots bounded by court and date windows, while Caselaw Access Project data is ingested as a large, bulk files. Both sources follow the same downstream batch transformations into analytics datasets.

Scalability Plan

The system scales horizontally through partitioned storage and distributed batch execution. Data is organized in a data lake and partitioned by source and time, allowing processing jobs and queries to read only relevant subsets as data volume increases. Bulk Caselaw Access Project files are processed in parallel across files, while CourtListener snapshots are ingested in bounded batches and processed downstream. Scaling is achieved by increasing partitions and available computing resources without modifying pipeline logic. The same pipeline supports larger datasets by moving storage from local disk to scalable object storage.

Metrics

System performance and efficiency will be evaluated using the following metrics:

- Ingestion throughput: number of records processed per second for each data source (CourtListener API snapshots and Caselaw Access Project bulk files).
- Processing latency: end-to-end runtime for major pipeline stages, including raw ingestion, normalization to curated datasets, and analytics table generation.
- Data volume processed: total input data size of raw JSON compared to output size curated Parquet, including storage reduction ratios.

Literature Review

Core Concepts and Technologies

HDFS: Introduces block-based storage, replication, and fault tolerance for large-scale batch analytics.[1]. The pipeline adopts these concepts by organizing court metadata in immutable data lake zones and assuming replication and durability are provided by the underlying storage layer rather than application logic.

MapReduce: Defines a scalable batch execution model based on map, shuffle, and reduce phases.[2]. The pipeline follows this model conceptually by mapping metadata records during ingestion, grouping records by keys such as court and time, and reducing them into aggregated analytics without directly implementing Hadoop MapReduce jobs.

DataFrame: Court metadata is represented as structured DataFrames with explicit schemas to enable scalable batch transformations and aggregations.[3]. By enforcing a unified schema at ingestion time, DataFrames allow the pipeline to perform efficient filtering, grouping, and aggregation operations while benefiting from Spark's optimized execution engine and catalyst-based query planning.

Spark SQL: Curated court metadata is queried using distributed SQL execution to compute trends and completeness statistics over partitioned datasets. [4]. Spark SQL enables declarative analytical queries while automatically translating them into parallel execution plans that minimize data movement and exploit partition pruning for performance.

Parquet: Court metadata is stored in Parquet format to reduce storage size and improve query performance for analytical workloads. [5]. As a columnar storage format, Parquet enables efficient compression and predicate pushdown, allowing analytical queries to scan only relevant columns and significantly reducing I/O costs in large-scale batch processing.

References

[1] K. Shvachko, H. Kuang, S. Radia, and R. Chansler, "The hadoop distributed file system," https://hadoop.apache.org/docs/stable/hadoop-project-dist/hadoop-hdfs/HdfsDesign.html, 2010.

[2] J. Dean and S. Ghemawat, "Mapreduce: Simplified data processing on large clusters," in *Proceedings of the 6th Symposium on Operating Systems Design and Implementation (OSDI)*, 2004. [Online]. Available: https://research.google.com/archive/mapreduce-osdi04.pdf

[3] M. Zaharia, R. S. Xin, P. Wendell, T. Das, M. Armbrust, A. Dave, X. Meng, J. Rosen, S. Venkataraman, M. J. Franklin, A. Ghodsi, J. Gonzalez, S. Shenker, and I. Stoica, "Apache spark: A unified engine for big data processing," *Communications of the ACM*, vol. 59, no. 11, pp. 56–65, 2016.

[4] M. Armbrust, R. S. Xin, C. Lian, Y. Huai, D. Liu, J. K. Bradley, X. Meng, T. Kaftan, M. J. Franklin, A. Ghodsi, and M. Zaharia, "Spark sql: Relational data processing in spark," in *Proceedings of the ACM SIGMOD International Conference on Management of Data*, 2015, pp. 1383–1394. [Online]. Available: https://people.eecs.berkeley.edu/~matei/papers/2015/sparksql_sigmod.pdf

[5] Apache Software Foundation, "Apache parquet documentation," https://parquet.apache.org/, accessed: 2025-02-07.