CollegeFootballDB: A Distributed Analytics Platform for SEC Play by Play Data

John Roehsler
February 5, 2026

***Abstract*—This project proposes a distributed analytics platform for processing and querying of SEC football data from 2021–2025. The system addresses the scale challenges of analyzing tens of thousands of plays across multiple seasons, demonstrating distributed storage, parallel processing, and query optimization techniques applicable to larger datasets.**

GitHub Repository: https://github.com/johnlroehsler/collegeFootballDB

***Index Terms*—college football, distributed systems, Apache Spark, HDFS, play by play analytics, SEC**

I. Project Overview

A. Domain

College football analytics of the Southeastern Conference (SEC), specifically focusing on offensive and defensive play calling trends, and situational performance.

B. Problem Statement

Current college football analytics tools struggle with processing historical play by play data at scale. Analysts and coaches need to answer complex questions like "How has offensive play calling changed in the recent years of college football?" or "What defensive formations are the most effective?" These queries require processing tens of thousands of plays across 5 seasons, joining play data with contextual information, and delivering fast query response times for analytical questions.

Traditional approaches using tools like pandas are inefficient when performing complex aggregations and joins across multiple seasons. This project builds a distributed system that leverages HDFS storage, Spark processing, and columnar data formats to enable efficient analysis.

C. Scope

In Scope:

- SEC Conference teams (14 teams, Texas and Oklahoma are excluded due to joining the SEC in 2024) for seasons 2021–2025
- Play by play data ingestion from CollegeFootballData.org API
- Batch processing pipeline: JSON ingestion to Parquet conversion to aggregation
- Distributed storage using HDFS or S3
- Spark based processing for team and situational statistics
- SQL query interface for analytical questions

Out of Scope:

- Real time streaming
- Machine learning model training
- All 130 FBS teams
- Video analysis

II. System Description

A. Data Sources

The primary data source is the CollegeFootballData.org API, which provides:

- Play by Play Data: JSON format containing down, distance, yard line, play type, yards gained, scoring plays
- Game Metadata: Venue, attendance, weather conditions
- Team Information: Rosters, recruiting rankings, historical records

Secondary sources include AP Poll rankings and injury reports.

B. Data Characteristics

Volume:

- 14 SEC teams $\times$ 12 games per season $\times$ 180 plays per game = 30,240 plays per season
- 5 seasons (2021–2025) = 151,200 total plays
- Estimated storage: 500MB raw JSON per season, 2.5GB for 5 seasons
- With replication factor of 3 in HDFS: 7.5GB distributed storage
- After Parquet conversion with compression: approximately 1.5GB

Variety:

- Structured: Play by play JSON with consistent schema
- Semi structured: Team rosters, game metadata
- Tabular: Rankings, weather data
- Different levels: Play level, drive level, game level

Velocity:

- Batch processing: Historical data loaded once
- Update frequency: None, the last season (2025) has already ended

C. Stack Layers Engaged

Storage Layer: HDFS or Amazon S3 provides distributed file storage with replication factor of 3. Partition strategy: `/plays/season=YYYY/week=W/`. Block size: 128MB

Syntax and Encoding Layer: Raw ingestion uses JSON from the API. The processing format is Apache Parquet with

Snappy compression. Columnar storage allows for efficient column based queries.

Data Model Layer: Spark DataFrames provide schema enforcement for play records with typed columns: `game_id`, `play_number`, `yards_gained`. This enables SQL like operations with type safety.

Data Store Layer: Parquet files serve as the primary store for immutable historical data. HBase is a stretch goal for indexed access to aggregated team statistics.

Processing Layer: Apache Spark provides the DataFrame API for transformations, MapReduce pattern for aggregations, and lazy evaluation with optimized DAG execution.

Query Layer: Spark SQL offers a SQL interface for analytical queries with predicate pushdown for efficient filtering. Jupyter notebooks provide interactive analysis.

D. Assumptions

- CollegeFootballData.org API remains accessible and free
- Play by play data schema is consistent across seasons
- Network bandwidth sufficient for API data downloads
- Historical data quality is reliable with minimal missing plays

III. Implementation Approach

A. Technology Choices

Design Philosophy: The technology choices prioritize horizontal scalability over single machine optimization. While the 151,200 play dataset could be processed on a typical computer using pandas and SQLite, that approach would make it harder to scale up in the future. For example, if you were to scale to all 136 FBS teams, it would require a complete architecture rewrite.

Table I summarizes the technology stack.

TABLE I
Technology Stack

Layer	Technology	Justification
Storage	HDFS or S3	Distributed, fault tolerant storage
Processing	Apache Spark 3.x	Fast, in memory processing, SQL support
Syntax	JSON to Parquet	Columnar format, faster queries
Data Model	Spark DataFrames	Schema enforcement, optimization
Query	Spark SQL	Familiar SQL syntax, optimizer
Resource Mgmt	YARN	Job scheduling, multi tenancy

Rationale:

- Spark over MapReduce: Faster for iterative queries and a better developer experience
- Parquet over JSON: Columnar storage reduces I/O for analytical queries
- HDFS replication: Ensures data availability despite node failures

B. Processing Model

The system uses a batch processing model with three main pipelines:

Pipeline 1: Data Ingestion. A Python script calls the CollegeFootballData.org API, fetches play by play JSON for each game, and writes raw JSON to HDFS: `/raw/season=YYYY/week=W/*.json`.

Pipeline 2: Format Conversion. A Spark job reads JSON files, applies schema validation, converts to Parquet with Snappy compression, and writes to `/processed/season=YYYY/week=W/*.parquet`.

Pipeline 3: Aggregation (MapReduce Pattern). The map phase extracts (team_id, yards, play_type, success) from each play. The shuffle phase groups all plays by team_id and game_id. The reduce phase calculates per game statistics including total yards, yards per play, success rate, big yard gain play percentage, and third down conversion rate. Results are written to `/stats/team_game_stats.parquet`.

C. Scalability Plan

The architecture is designed to scale horizontally.

Current Scale (SEC, 5 seasons):

- 151,200 plays, 2.5GB raw data
- Single HDFS cluster: 3 DataNodes
- Spark: 1 master, 3 workers (4 cores, 8GB RAM each)

Scaling to All FBS (130 teams):

- 2.4M plays per year, 50GB for 10 years
- Add DataNodes: Scale to 10–20 nodes
- Add Spark executors: 10+ workers for parallel processing
- Partition by conference: `/plays/conference=SEC/season=YYYY/`

Partitioning Strategy:

- Time based partitioning: Queries often filter by season and week
- Hash partitioning by team_id for team specific aggregations
- Enables partition pruning: Skip irrelevant data during queries

D. Metrics

The following metrics will measure system performance:

Throughput: Plays processed per second during the aggregation pipeline. Target: 1,000+ plays per second on a three node cluster.

Query Latency: Time to execute: `SELECT AVG(yards) FROM plays WHERE team='Georgia' AND season=2025`. Compare JSON (baseline) vs. Parquet (optimized). Expected improvement: 10× faster with Parquet.

Storage Efficiency: Compression ratio of JSON size vs. Parquet size. Expected: 3–5× reduction with Snappy compression.

Scalability: Processing time vs. data volume (1 season vs. 5 seasons). Expected: Linear scaling with additional Spark executors.

IV. Literature Review

A. MapReduce

Dean and Ghemawat [1] introduced the MapReduce programming model for processing large datasets across distributed clusters. The model consists of two functions, map, which processes key and value pairs to generate intermediate pairs and reduce, which merges intermediate values.

Application to project: The aggregation pipeline implements the MapReduce pattern. The map phase extracts (team_id, play_metrics) from each play record. The shuffle phase groups all plays by team_id. The reduce phase calculates statistics on a per game basis. Spark's DataFrame API internally implements these MapReduce functions without any need for extra optimization.

B. The Google File System

Ghemawat, Gobioff, and Leung [2] designed GFS to provide fault tolerant storage across many types of hardware. Key concepts include large block sizes (64MB) to reduce metadata overhead, replication for fault tolerance, and architecture for coordination.

Application to project: HDFS directly implements GFS concepts. The play by play JSON files are split into 128MB blocks and replicated across 3 DataNodes. If one DataNode fails, then the NameNode redirects reads to replica blocks, ensuring historical data remains available despite hardware failures.

C. Bigtable

Chang et al. [3] introduced Bigtable, a persistent multi dimensional sorted map. Key design elements include row key design for efficient range scans, column families for related data grouping, and LSM trees for write optimized storage.

Application to project: If the HBase stretch goal is implemented, then the Bigtable concepts will be used. The row key design (team-season-week) enables queries like "get all Georgia stats for 2021–2022." Column families separate offensive metrics from defensive metrics.

D. Spark

Zaharia et al. [4] introduced Resilient Distributed Datasets (RDDs), an abstraction for fault tolerant and parallel data structures. Key innovations include lazy evaluation, in memory caching, and the distinction between transformations and actions.

Application to project: The Spark jobs use DataFrames for processing. Spark builds an optimized execution plan before touching data. This enables query optimization that pure MapReduce lacks.

E. Parquet

Parquet is a columnar storage format designed for analytics workloads. Benefits include column pruning, efficient compression, and predicate pushdown.

Application to project: Converting JSON to Parquet is critical for query performance. A query like `SELECT AVG(yards) WHERE down=3` only reads the yards and down columns, ignoring any other fields. A benchmark on query latency comparing JSON vs. Parquet will demonstrate the speedup from columnar storage.

V. Architecture Diagrams

Stack Architecture Diagram: Fig. 1 shows the layered architecture from storage through querying. Each layer maps to specific technologies and demonstrates how data flows through the transformations.

Fig. 1. Stack architecture diagram.

Data Flow Pipeline Diagram: Fig. 2 illustrates the end to end data pipeline from API ingestion through query results. Parallelization points are marked, showing where Spark distributes work across executors.

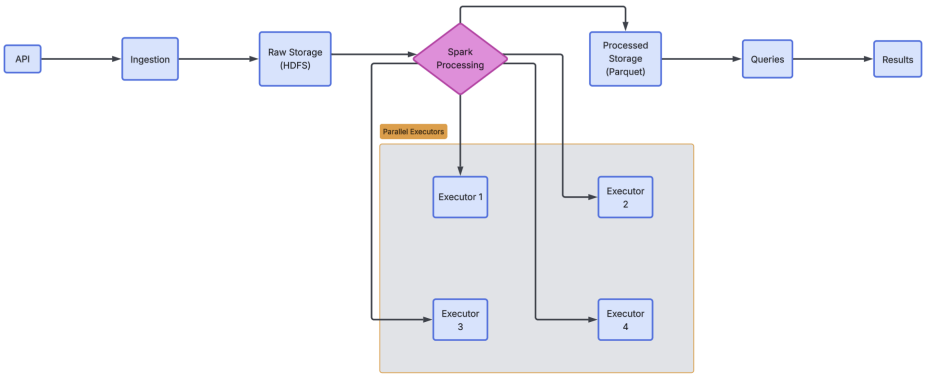

Fig. 2. Data flow pipeline diagram.

VI. Project Timeline

Milestone 1 (Current): Architecture and proposal — define the system architecture, download sample data (2025 season), set up GitHub repository.

Milestone 2: Implementation — set up HDFS or S3 storage, implement ingestion pipeline, build Spark processing jobs, initial query testing.

Milestone 3: Optimization and Analysis — benchmark performance metrics, optimize query performance, generate analytical insights from data, final report and presentation.

References

[1] J. Dean and S. Ghemawat, "MapReduce: Simplified data processing on large clusters," in *Proc. 6th Symp. Operating Systems Design and Implementation (OSDI)*, 2004, pp. 137–150.

[2] S. Ghemawat, H. Gobioff, and S.-T. Leung, "The Google File System," in *Proc. 19th ACM Symp. Operating Systems Principles (SOSP)*, 2003, pp. 29–43.

[3] F. Chang *et al.*, "Bigtable: A distributed storage system for structured data," in *Proc. 7th Symp. Operating Systems Design and Implementation (OSDI)*, 2006, pp. 205–218.

[4] M. Zaharia *et al.*, "Spark: Cluster computing with working sets," in *Proc. 2nd USENIX Workshop Hot Topics in Cloud Computing (HotCloud)*, 2010.

[5] G. Fourny, *The Big Data Textbook: From Clay Tablets to Data Lakehouses*, 2nd ed., 2024.

[6] CollegeFootballData.org API Documentation. [Online]. Available: https://collegefootballdata.com