भारतीय प्रौद्योगिकी संस्थान जोधपुर

Indian Institute of Technology Jodhpur

**Software And Data Engineering(CSL7090)**

*A Design Document*

*for*

# IPL Data Analysis using Apache Spark

*by*

Mehul Sharma (M24CSE013)

Shivender Thapa (M24CSE023)

Shreyank Jaiswal(M24CSE024)

# Contents

# 1 Title and Overview

## Title

IPL Data Analytics using Apache Spark

## Overview

This project utilizes Apache Spark for large scale IPL datasets so that people can get trends and insights over player performances, team strategies, and match outcomes. In this project, the main objectives were using the distributed processing and in-memory computation capabilities to handle the complexities of IPL data efficiently. Tasks such as top scorer analysis, economical bowling patterns, toss impact evaluation, and venue-specific trends are addressed in this project. This scope includes performance analysis, pattern identification, and the probability of predictive modelling through Spark machine learning components.

# 2 Architecture

The architecture of the project includes the structured elements listed below, based on the concept of smart processing and analysis of IPL datasets:

## High-Level Design

### 1. Data Sources

The datasets of IPL, mainly come in the CSV format, as match.csv, player.csv, etc. Outside libraries mainly PySpark and Spark SQL are used for processing of data.

### 2. Data Ingestion Layer

The data is loaded into Spark with predefined schemas ensuring type consistency and hence minimizing the overhead of preprocessing.

### 3. Treatment Layer

All the transformations mentioned above, from null-handling to feature and business logic derivation, implement on Spark DataFrames and SQL. Analytic queries should essentially be core operations: joining, aggregating, filtering.

### 4. Analytical Layer

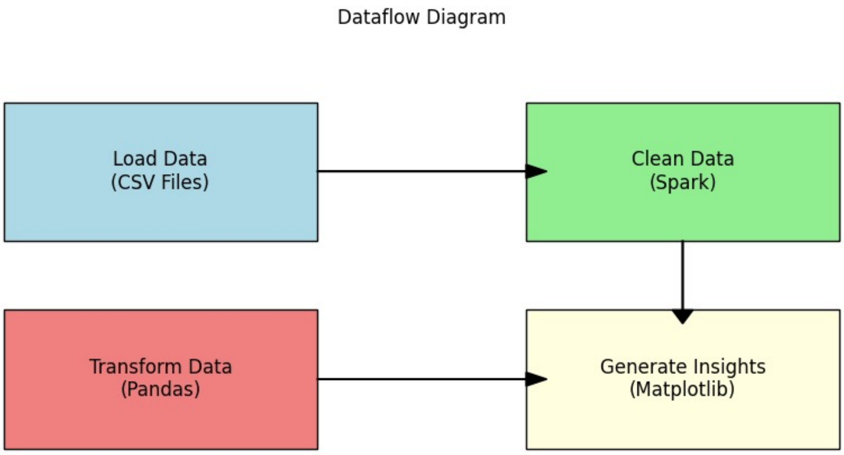
SQL-based analytics of the insights extracted, namely top scorers as well as venue trends. Query results are cached and shown for readability.

### 5. Output Layer

The results can be formatted to visualize or exported for later use. Good extensibility for predictive analytics: usage of Spark MLlib in future releases.

# 3 Diagrams

## Data Flow Diagram



# 4 Module Description

## Summary of Research Paper

A research paper on Apache Spark, architecture, components, and capabilities for big data analytics:

- **Spark Core:** The underlying engine providing RDDs, cluster manageability, fault tolerance, and in-memory processing.
- **Libraries:**
  - The structured data processing case end.
  - MLlib for machine learning.
  - GraphX for graph computation.
  - Spark Streaming for real-time data processing.

## From Code: Observations

Processing the structured cricket match data was done through IPL data analysis using Spark SQL and custom schema on `Ball_By_Ball`, `Match`, and `Player`. The majority of the operations entailed:

- **Data Loading:** Schemas are defined as pre-established data structures.
- **Processing:** Includes mapping transformations and operations to data.
- **Analytics:** Group, Filter, and Join datasets to provide more insight into IPL.

# 5 Data Structures and Schema Database

## From the Code:

- **Custom Data Schemes:**
  - **Ball_by_Ball:** `match_id, over_id, runs_scored, wides, bowler_extras`, etc.
  - **Match:** Fields would include `match_id, team1, team2, venue_name, winner`, etc.
  - **Player:** `Player ID, Player Name, Batsman, Bowling Skill`.
- **Data Integration:**
  - It can be used to store and process DataFrames.
  - Schema ensures consistency across datasets, facilitating ETL processes.

# 6 API Description

## APIs Observed

- **Spark Session:** An entry point to any Spark operation.
- **DataFrame API:** For structured data manipulation.
  - Methods such as `.show()`, `.filter()`, `.groupBy()`, and `.agg()` for querying.
- **PySpark Functions:**
  - Changes: `col`, `when`, `avg`.
  - Aggregations: `sum`, `row_number`.

# 7 Algorithms or Workflows

## From the Research Paper

Apache Spark applies its in-memory processing and DAG execution engine to enable scalable workflows. The paradigms below apply:

- **Data Abstraction:**
  - RDDs: Resilient Distributed Datasets (fault-tolerant, distributed computation).
  - DataFrames: Structured processing at a higher level.
  - Machine Learning Pipelines (MLlib): Stages of the pipeline, including data preparation, feature selection, model training, and testing.
  - GraphX for Graph Analytics: Graph algorithms used include PageRank and connected components.

## From the Code

This IPL data analysis is based on a workflow:

1. **Load Data:** Using PySpark and pre-defined schemas to read CSV files.
2. **Data Preprocessing:**
   - Row filtering, selecting columns, and creating new fields using functions like `when` and `avg`.
   - For instance, parsing date formats and completing missing data for major fields like `win_margin`.
3. **Data Aggregation and Analysis:**
   - Using Spark SQL and the PySpark DataFrame API to derive insights.
   - Aggregate runs scored in matches and identify high-performing players based on metrics like *man-of-the-match* awards.

    - Example: Use `groupBy` and `sum` to calculate team statistics.

4. **Visualization and Reporting:**

    - While not included in the code, processed data could be visualized using Python libraries or Spark SQL integration with BI tools.

## Workflow Example (Inferred from Code)

```
# Initialize SparkSession
spark = SparkSession.builder.appName("IPL Data Analysis"
   ).getOrCreate()

# Load Data
ball_by_ball_df = spark.read.schema(ball_by_ball_schema)
   .format("csv") \
    .option("header", "true").load("Ball_By_Ball.csv")

# Preprocess Data
match_df = match_df.fillna({’win_margin’: 0, ’
   outcome_type’: ’Unknown’})

# Query Data
runs_per_match = ball_by_ball_df.groupBy("match_id") \
    .agg(sum("runs_scored").alias("total_runs"))

# Results
runs_per_match.show()
```

# 8 Technology Stack

## Tools and Frameworks

- **Apache Spark:**
  - For distributed data processing and analysis.
  - Parts:
    - Spark SQL for structured data queries.
    - DataFrame API for fast transformations and aggregations.
- **Databricks:**
  - Shared collaborative workspace for processing and analytics on Spark.
  - Provides managed clusters and notebooks for smooth execution and visualization.
- **Amazon S3:**
  - Cloud Storage Solution for efficient data storage and retrieval.
  - Supports large-scale and secure data storage.
- **PySpark:** Python API for Spark; enables the implementation of workflows in Python.
- **GraphFrames:** Library for graph analytics, such as network analysis of cricket matches or players.

## Programming Languages

- **Python:** Used for writing PySpark scripts, data cleaning, transformations, and querying. Libraries like pandas or matplotlib may also be used for local visualizations.
- **SQL:** Embedded within Spark SQL for querying and data manipulation.

## Platforms and Tools

- **Databricks:** For developing, running, and managing big data applications.
- **Amazon Web Services (AWS):**
  - S3 Buckets for data storage.
  - Provides elasticity for varying data sizes and processing needs.
- **Jupyter Notebooks:** For local development and testing PySpark workflows.
- **Visualization Tools:** Libraries like Seaborn or integration with tools like Tableau or Power BI.

## Environment and Network

- **Cluster Environment:** Deployed on a cluster of machines using YARN, Mesos, or Standalone resource manager for distributed computing.
- **Cloud/Storage:**
  - HDFS: Hadoop Distributed File System for storing large datasets.
  - Optional: AWS S3 or Azure Blob Storage for hosting data.
- **Local Development:** Tools like Google Colab or Jupyter Notebooks for local PySpark execution and debugging.
- **Dependencies:**
  - Spark Packages: GraphFrames for graph processing.
  - Java JDK for running Spark.
  - PySpark with dependencies installed via pip.

# 9 Implementation Plan

## Phase 1: Planning and Preparation

- Define project scope and objectives.
- Review research paper methodologies and align project goals.
- Set up metrics for project success.
- Environment setup:
  - Deploy and install Apache Spark and PySpark.
  - Install necessary packages (GraphFrames, pandas, etc.).
  - Load and validate datasets (`Ball_By_Ball.csv`, `Match.csv`, `Player.csv`).

## Phase 2: Data Preprocessing

- Define schemas using PySpark's `StructType` and `StructField`.
- Data cleansing:
  - Treat null values (e.g., fill `win_margin` with 0).
  - Standardize date formats and data types.
- Data integration:
  - Join datasets (`Ball_By_Ball`, `Match`, `Player`).
  - Derive columns for player performance metrics.

## Phase 3: Data Analysis and Feature Engineering

- Perform Exploratory Data Analysis (EDA).
- Create new variables (e.g., player strike rates, win-loss ratios).
- Apply graph analytics using GraphFrames for network-based insights.

## Phase 4: Advanced Analysis and Reporting

- Apply machine learning using Spark MLlib (optional).
- Generate visualizations and dashboards.
- Prepare a detailed project report summarizing findings.

## Phase 5: Implementation and Evaluation

- Implement and finalize workflows.
- Perform validation and testing (e.g., schema validation, unit testing, integration testing).
- Prepare the final deliverables.

# 10 Risk and Mitigation

## Identified Risks and Mitigation Strategies

1. **Data Quality Issues**
   - **Risk:** Missing, inconsistent, or inaccurate data in datasets (e.g., missing `win_margin` or `player_name` values); incorrect data formatting.
   - **Mitigation:**
     - Perform rigorous data validation checks using PySpark schema validation (`StructType`).
     - Handle null values using `fillna()` for defaults.
     - Standardize date formats with PySpark's `to_date()` function.
     - Cross-verify dataset completeness with IPL statistics from trusted sources.
2. **Large Dataset Processing Bottlenecks**
   - **Risk:** Slow performance due to dataset size or inefficient queries.
   - **Mitigation:**
     - Optimize Spark jobs using caching (`persist()`).
     - Use partitioning strategies to reduce shuffling.
     - Monitor resource usage via the Spark UI.
3. **Environment Setup Challenges**
   - **Risk:** Issues during the setup of Spark environment or dependencies like GraphFrames.
   - **Mitigation:**
     - Follow detailed setup guides for Spark and GraphFrames.
     - Validate the setup using small test queries before full workflows.
4. **Coding Errors or Bugs**
   - **Risk:** Logical errors during data transformations or joins.

- **Mitigation:**
    - Perform unit testing on critical code blocks.
    - Utilize debugging tools like PySpark logs and Spark UI.

5. **Insufficient Knowledge of Spark or PySpark**
    - **Risk:** Difficulty implementing advanced Spark features.
    - **Mitigation:**
        - Allocate time for studying Spark fundamentals using Databricks tutorials and official documentation.
        - Start with basic operations before implementing advanced workflows.
6. **Project Scope Creep**
    - **Risk:** Adding too many features, leading to delays or incomplete work.
    - **Mitigation:**
        - Define project scope and milestones clearly.
        - Use an iterative approach, prioritizing essential tasks first.
7. **Compatibility Issues with Libraries or Datasets**
    - **Risk:** Version mismatches or incompatible dataset formats.
    - **Mitigation:**
        - Test library compatibility during the setup phase.
        - Maintain consistent Spark and Python versions across development.
8. **Visualization Challenges**
    - **Risk:** Difficulty in creating clear, insightful visualizations.
    - **Mitigation:**
        - Use Python libraries like Matplotlib or Seaborn.
        - Begin with simple plots and refine them iteratively.
9. **Time Constraints**
    - **Risk:** Insufficient time to complete all tasks.
    - **Mitigation:**

- Follow the timeline and milestones rigorously.
- Focus on delivering a minimum viable product (MVP) first.

10. **Stakeholder Expectations**

    - **Risk:** Misaligned expectations about deliverables or outcomes.
    - **Mitigation:**
        - Communicate scope, goals, and progress clearly with stakeholders.
        - Share interim results regularly for feedback.

# 11 References and Appendices

## References

- Research Paper:
  - Title: *Big Data Analytics on Apache Spark*
  - Authors: Salman Salloum, Ruslan Dautov, Xiaojun Chen, Patrick Xiaogang Peng, Joshua Zhexue Huang
  - Published in: Int J Data Sci Anal (2016)
  - DOI: 10.1007/s41060-016-0027-9
- Code Base: IPL Data Analysis Code with PySpark and GraphFrames.
- Libraries and Dependencies:
  - PySpark
  - GraphFrames
  - Matplotlib, Seaborn (optional for visualizations)
- Datasets:
  - `Ball_By_Ball.csv`: Contains ball-by-ball data for IPL matches.
  - `Match.csv`: Match-level details including teams, dates, and outcomes.
  - `Player.csv`: Player information such as batting and bowling skills.
- Documentation:
  - Apache Spark Official Documentation
  - PySpark API Documentation
  - GraphFrames Library Documentation
- Tutorials:
  - Databricks Learning Portal
  - PySpark Programming Guide

# Appendices

## Appendix A: Architecture Diagram

Illustrates the data flow in the project, including:

- Input datasets: `Ball_By_Ball.csv`, `Match.csv`, `Player.csv`.
- Data preprocessing pipeline: schema definition, cleaning, transformations.
- Analytics and graph processing: Spark SQL and GraphFrames workflows.
- Output: Aggregated results and visualizations.

## Appendix B: Sample Code Snippets

**Schema Definition:**

```
from pyspark.sql.types import StructType, StructField,
   IntegerType, StringType

ball_by_ball_schema = StructType([
    StructField("match_id", IntegerType(), True),
    StructField("over_id", IntegerType(), True),
    StructField("ball_id", IntegerType(), True),
    StructField("runs_scored", IntegerType(), True),
    StructField("extra_type", StringType(), True)
])
```

**Loading Data:**

```
ball_by_ball_df = spark.read.schema(ball_by_ball_schema)
   .csv("Ball_By_Ball.csv", header=True)
```

**Basic Aggregation:**

```
total_runs = ball_by_ball_df.groupBy("match_id").agg(sum
   ("runs_scored").alias("total_runs"))
total_runs.show()
```

## Appendix C: Example Output

**Sample Aggregated Data:**

```
+---------+-----------+
| match_id| total_runs|
+---------+-----------+
| 123456  | 180       |
| 123457  | 200       |
+---------+-----------+
```

## Appendix D: Troubleshooting Guide

- Common Errors:
  - Schema mismatch: Verify dataset headers and schema definitions.
  - Null values causing errors: Use `.fillna()` to handle missing data.
- Dependency Issues:
  - Ensure compatible versions of PySpark and GraphFrames are installed.
  - Check Spark and Hadoop versions for cluster compatibility.

# GitHub Project Link

You can find the complete project on GitHub at the following link:
`https://github.com/Mehul-420/IPL-DataAnalysis-Using-Spark`