VRIJE
UNIVERSITEIT
BRUSSEL

ANALYZING BASKETBALL GAME DATA

A Parallel Processing Approach

Hatim Draoui

23rd December 2024

Sciences and Bio-Engineering Sciences

Problem Description

This project focuses on analyzing basketball game data to discover useful insights and trends. This research was carried out using a CSV file containing basketball game records. Each record contains a variety of properties, including the game date, teams participating, scores, and other useful information.

For the analysis, we used Akka Streams and followed the pipe-and-filter architectural paradigm. The project involved solving four different queries:

1. **Victories on Sundays:** Calculating the number of victories for each side in games played on Sunday.
2. **Close Game Victories:** Determine the number of victories with a score differential of less than 5 points.
3. **Quarterfinals or Beyond:** Counting how many times a team has advanced to the quarterfinals.
4. **Losses in the 1980s:** Calculating each team's amount of losses between 1980 and 1990.

To achieve these objectives, we used specific filters and transformations for each query. The sections that follow will provide a detailed description of our solution's primary components and flow structure.

Main Components

Now, I will discuss the main components of the system:

Source

The source is a combined component that reads data from the CSV file and converts it into a `GameRecord`. This transformation facilitates the extraction of data associated with each game, making it easier to process and analyze the information.

Motivation

- Enables efficient handling of large datasets by streaming data into the pipeline incremently

Main Processing Component

This component is where the primary processing occurs. It handles all queries in parallel and is composed of several flows, which are divided into three categories: **Filter Flows**, **Conversion Flows**, and **Counting Flows**.

Counting Flows

These flows are responsible for counting the number of teams that have achieved specific tasks, such as winning or losing. They are essential because they enable the conversion of `GameRecords` into Maps, which simplifies the process of writing data to files.

Motivation

- Transformation of GameRecord objects into maps simplifies the aggregation logic and prepares data for further analysis

Filter Flows

These flows allow us to filter the data stream based on specific criteria. After filtering, the data passes through the counting flows, enabling the visualization of results based on these filtered subsets.

Conversion Flows

These flows perform the final transformation before the data is merged into the sink. They are tasked with converting the Maps into `GameStatistics`, which allows the sink to easily write the data in the correct format and to the appropriate file.

Motivation

- Immutability of data in Akka Streams ensures that transformations are safe and predictable.

Sink

The sink consists of a writer flow that writes the processed data to a file. This flow ensures that the output is correctly formatted and saved to the designated file.

Motivation

- Akka Streams' Sink abstraction supports multiple output mechanisms

Solution

Flow Description

Our system, as shown in the graphic above, is designed to process basketball game data efficiently. The workflow begins with direct input from the dataset, which is immediately turned into GameRecord objects. This transformation enables the organized extraction of essential data for each game.

The GameRecords are then distributed to different branches, each of which is dedicated to processing a single query. Within each branch, data is sent through a series of filters to extract important subsets, followed by transformations to prepare it for analysis. These changes eventually yield GameStatistics objects, which contain the results in a structured and formatted manner.

Finally, the sink processes the GameStatistics objects and writes the results to the appropriate output files. This methodical methodology ensures that the data is correctly filtered, processed, and stored while adhering to the Pipe-and-Filter architecture paradigm.

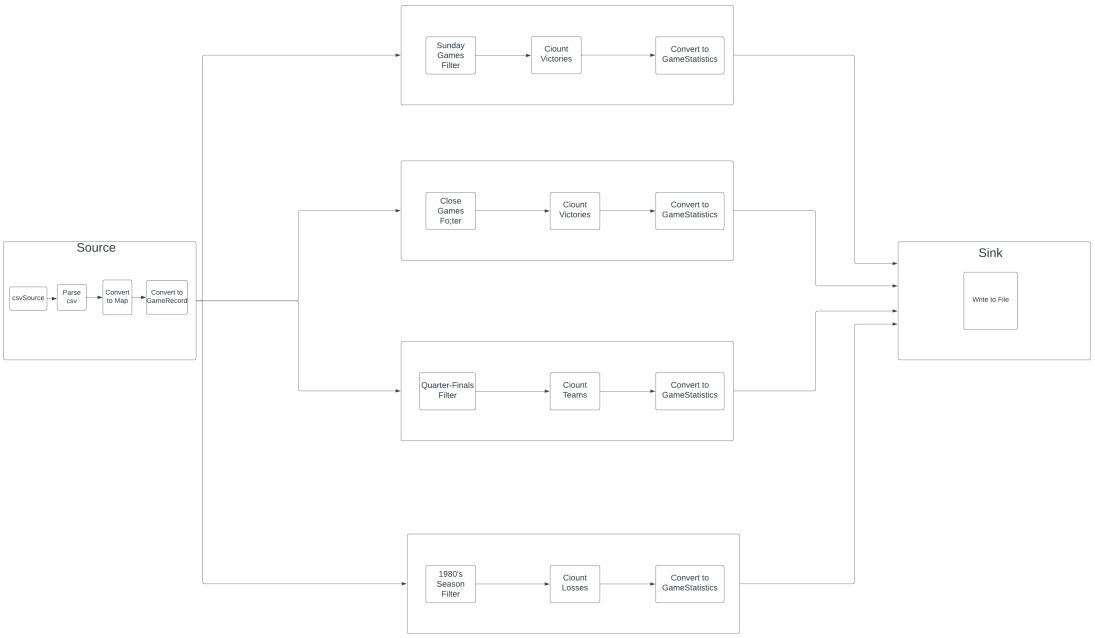

Figure 1: Illustration of our solution

Advantages

- Parallelism: The solution can handle multiple queries concurrently.
- Backpressure: Ensures a smooth flow of data through the pipeline.
- Modularity: Each flow is independently defined, making the solution easier to test, maintain, extend.
- Scalability: Can scale horizontally to handle larger datasets or increased query complexity.