Hadoop MapReduce Analyse von Flugbewegungen

Dieses Projekt analysiert Flugbewegungen von Verkehrsmaschinen europäischer Flughäfen für die Jahre 2019 und 2024.

Ziel ist es, die Veränderung der Flugbewegungen von 2019 nach 2024 pro Flughafen und pro Land, in denen die Flughäfen liegen, zu berechnen und die Ergebnisse absteigend nach prozentualer Änderung zu sortieren.

Datensatz

Grundlage ist die CSV-Datei **airport_traffic.csv**, in der detaillierte Informationen über Flugbewegungen per Flughafen und Tag enthalten sind.

▸ **Die ersten 10 Zeilen der Datei**

- Die Datei enthält **228.729 Zeilen** mit Flugverkehrsdaten einzelner Flughäfen für die Jahre **2019 und 2024**.
- Nur Datensätze mit einem gesetzten Wert in der Spalte `FLT_TOT_IFR_2` werden für die Analyse berücksichtigt, da diese die bereits zusammengezählten (total) **Flugbewegungen mit Verkehrsmaschinen (IFR)** darstellen.
- Dies entspricht **72.627 Zeilen (ca. 32%)**, welche sich auf Daten zu Passagiers-Flughäfen begrenzen.

▸ **Die in der CSV enthaltenen Werte**

Die Davon relevanten Spalten sind `YEAR`, `APT_ICAO`, `STATE_NAME` und `FLT_TOT_IFR_2`

Projektstruktur

```
mapreduce-airtraffic/
├── input/
│   ├── airport_traffic.csv
├── src/main/java/dhbw/bigdata/
│   ├── Main.java
│   ├── utils/
│   │    ├── MapReduceJob.java
│   ├── preprocessing/
│   │    ├── PrepMapper.java
│   │    ├── PrepCombiner.java
│   │    └── PrepReducer.java
│   ├── countries/
│   │    ├── CountryMapper.java
│   │    ├── CountryReducer.java
│   │    ├── CPercentageMapper.java
│   │    └── CPercentageReducer.java
│   └── airports/
│        ├── APercentageMapper.java
│        └── APercentageReducer.java
├── pom.xml
└── README.md
```

Packages des Maven Projekts

Package	Aufgabe
`preprocessing/`	Filtern der CSV. Aggregation der Flugbewegungen pro Flughafen (inkl. Land). Fasst die jeweiligen Werte von 2019 und 2024 in einer Zeile zusammen.
`countries/`	Aggregation der Flugbewegungen pro Land. Berechnung der prozentualen Veränderung & Mapping nach dieser. Gibt alle Länder, sortiert nach Prozenten, aus.
`airports/`	Mapping nach (in Preprocessing berechneter) prozentualer Änderung. Gibt alle Flughäfen (ICAO-Codes), sortiert nach Prozenten, aus.
`utils/`	`utils.MapReduceJob`: Definiert Mapper, Reducer und Combiner, Input- und Output-Pfade, Output-Key- und Value-Klassen. Objekte dieser Klasse können mit `.run()` ausgeführt werden.

Weitere Dateien in diesem Repository

```
├── output/
│   ├── preprocessing/part-r-0000
│   ├── temp/part-r-0000
│   ├── countries/part-r-0000
│   └── airports/part-r-0000
└── airtraffic.jar
```

- **`/output/`**: Verzeichnisse Analog zu denen im HDFS, inklusive Ausgabedateien der MapReduce-Jobs
  ![output_dirs_screenshot](./public/hdfs_out put_directories.png)
- **`airtraffic.jar`**: vorkompilierte JAR-Datei des Projekts

Anleitung zur Verwendung des Sourcecodes

1. **Repository clonen** (oder ZIP herunterladen und entpacken) & in das Projektverzeichnis wechseln

   ```
   git clone https://github.com/sky-ash/mapreduce-airtraffic.git
   cd mapreduce-airtraffic
   ```

2. **JAR-Datei kompilieren** (`maven` muss installiert sein)

   ```
   mvn clean package
   mv target/airtraffic-3.7-final.jar airtraffic.jar
   ```

   > Alternativ kann auch die vorkompilierte `airtraffic.jar` verwendet werden.

3. **Hadoop starten**

   ```
   start-dfs.sh
   start-yarn.sh
   ```

4. **CSV-Datei in HDFS laden**:
   Der Preprocessing-Job erwartet die CSV im `/input/`-Directory des HDFS.
   Wir erstellen das Verzeichnis und legen die Datei dort ab.

```
hdfs dfs -mkdir /input
hdfs dfs -put data/airport_traffic.csv /input
```

5. **`/output/`-Pfade freilegen**:
   Um sicherzustellen, dass die Ausgabeverzeichnisse leer sind, können sie vor dem Start des Jobs gelöscht werden:

```
hdfs dfs -rm -r /output/preprocessing/
hdfs dfs -rm -r /output/temp/
hdfs dfs -rm -r /output/countries/
hdfs dfs -rm -r /output/airports/
```

Alternativ kann auch das `/output/`-Verzeichnis als ganzes gelöscht werden:

```
hdfs dfs -rm -r /output/
```

6. **Ausführen des Programms**:
   Da die `MapReduceJob`-Klasse für jede Instanz automatisch die richtigen Input- und Output-Pfade setzt, kann die JAR-Datei ohne weitere Argumente ausgeführt werden:

   ```
   hadoop jar airtraffic.jar
   ```

   Die `Main`-Klasse führt automatisch alle MapReduce Jobs in der richtigen Reihenfolge durch. Daraufhin befinden sich die finalen Analyse-Ergebnisse im HDFS unter:
   - `/output/countries/part-r-00000`
   - `/output/airports/part-r-00000`

Funktionsweise des Codes

Im Folgenden wird Schritt für Schritt erläutert, wie die Daten mithilfe der drei relevanten **MapReduce-Packages** verarbeitet werden. Diese Packages sind:

1. **`preprocessing`** – Bereitet die Daten aus der CSV-Datei auf und aggregiert Flugbewegungen je Flughafen.
2. **`countries`** – Fasst alle Flughäfen eines Landes zusammen und sortiert die Länder nach prozentualer Änderung.
3. **`airports`** – Sortiert direkt die Flughäfen selbst nach prozentualer Änderung.

1. Preprocessing

Ziel dieses ersten Jobs ist es, aus der großen CSV-Datei nur die relevanten und gültigen Datensätze herauszufiltern (Jahre **2019** bzw. **2024**, gesetzter Wert in `FLT_TOT_IFR_2`) und pro Flughafen eine aggregierte Zeile bereitzustellen, die sowohl die Gesamtbewegungen von 2019 als auch 2024 enthält.

1.1 PrepMapper

- **Input**: Zeilen der Original-CSV aus HDFS.

- **Funktionsweise**:
  - Überspringt die Header-Zeile.
  - Parst nur Zeilen mit gesetzter IFR-Spalte (`FLT_TOT_IFR_2`).
  - Lässt nur `YEAR=2019` oder `YEAR=2024` zu.
  - Gibt als **Key** den `airportCode` (z. B. `EDDF`), als **Value** den String `<stateName>,<year>,<flightCount>` aus.
- **Ausgabe (Beispielwerte)** :

```
EDDF    Germany,2019,12345
EDDF    Germany,2024,15600
...
```

1.2 PrepCombiner

- **Input**: Ausgabe der jeweiligen lokalen Mapper-Instanz, gruppiert nach Key (`airportCode`).
- **Funktionsweise**:
  - Summiert vorab die Flugbewegungen je `(airportCode, year)` , um die Datenmenge schon **vor** dem Shuffle zu reduzieren.
- **Ausgabe (Beispielwerte)** :

```
EDDF    Germany,2019,27945
...
```

  (Gleiche Struktur wie im Mapper, aber mit bereits aufsummierten Teilergebnissen)

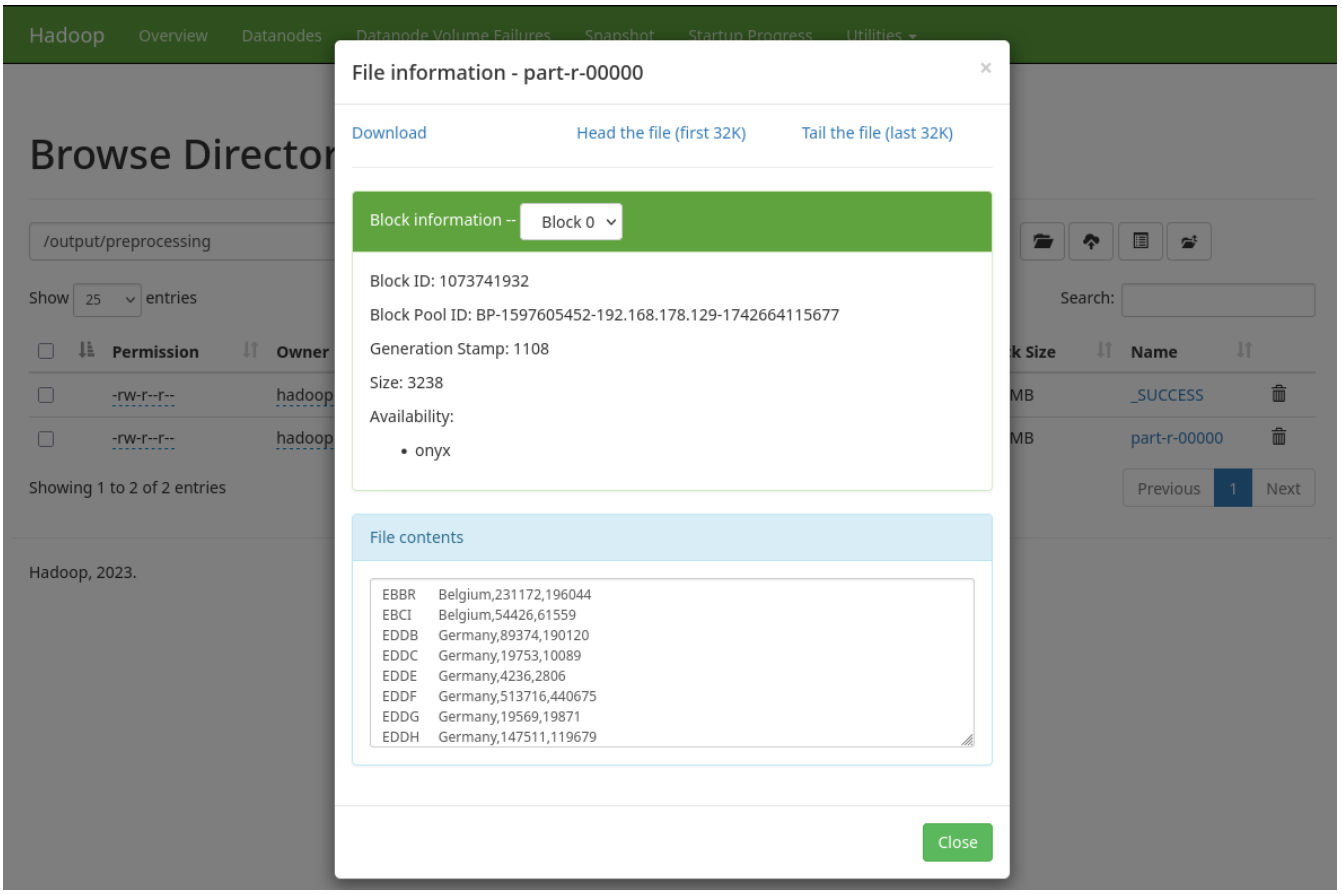
Ein **Combiner** ist sinnvoll, da Summation kommutativ + assoziativ ist und so die Netzwerkbelastung zwischen Mapper und Reducer sinkt.

1.3 PrepReducer

- **Input**: Teilergebnisse der Combiner
- **Funktionsweise**:
  - Finalisiert die Teilsummen der Combiner, sodass je `(airportCode, year)` eine Zeile übrig bleibt
  - kombiniert diese zu Zeilen, die jeweils die Werte beider Jahre enthalten `(total2019, total2024)` , sodass eine Zeile pro Flughafen entsteht
- **Ausgabe part-r-00000** :

```
EBBR    Belgium,231172,196044
EBCI    Belgium,54426,61559
EDDB    Germany,89374,190120
EDDC    Germany,19753,10089
...
```

All diese Zeilen werden in **/output/preprocessing/** geschrieben und bilden die Grundlage für die Auswertungen in den nächsten Schritten.



---

2. Countries

Ziel dieser Job-Kette ist es, **alle Flughäfen eines Landes zu aggregieren** und anschließend nach prozentualer Änderung zwischen den Jahren 2019 und 2024 zu sortieren. Dazu werden **zwei aufeinanderfolgende** MapReduce-Jobs ausgeführt:

1. **Aggregation**: Summiert pro Land die kumulierten Flugbewegungen (2019 & 2024).
2. **Sortierung**: Berechnet den prozentualen Unterschied und gibt die Länder in sortierter Reihenfolge aus.

2.1 CountryMapper

- **Input**: Ausgabe aus dem Preprocessing (Key = `airportCode`, Value = `<stateName>,<total2019>,<total2024>`).
- **Funktionsweise**:
  - Trennt den Value-String am Komma.
  - Übernimmt das `stateName` als neuen **Key**.
  - Erstellt als **Value** ein String-Paar `<total2019>,<total2024>`.
- **Ausgabe**:

```
Belgium    231172,196044
Belgium    54426,61559
Germany    89374,190120
Germany    19753,10089
...
```

2.2 CountryReducer

- **Input**: Alle Werte (Value-Strings) pro Land, gruppiert nach Key (`stateName`).
- **Funktionsweise**:
  - Summiert in einer Schleife alle 2019-Werte und 2024-Werte.
  - Schreibt anschließend das Land als **Key** und `<aggregated2019>,<aggregated2024>` als **Value**.
- **Ausgabe** **part-r-00000**:

```
Austria     281488,246184
Belgium     285598,257603
Bulgaria    59491,58919
Croatia     44331,49663
...
```

Diese Zwischenergebnisse werden z. B. in **/output/temp/** gespeichert und dienen als Input für den Sortierschritt.

2.3 CPercentageMapper

- **Input**: Zwischenergebnis aus dem vorherigen Schritt (Key = `stateName`, Value = `<aggregated2019>,<aggregated2024>`).
- **Funktionsweise**:
  - Parst die beiden Zahlen (2019 und 2024).

- Berechnet den prozentualen Unterschied:

$$\text{percentChange} = \frac{(2024 - 2019)}{2019} \times 100$$

(Sonderfall: Wenn 2019=0, wird ein default-Wert (+100 %) gewählt werden.)

- Übergibt die Prozentzahl als **Key** (IntWritable), sodass im folgenden Sort/Shuffle automatisch nach dieser Zahl sortiert wird, und übergibt `<stateName>,<aggregated2019>,<aggregated2024>` als **Value**.

- **Ausgabe (Beispieldaten)**:

```
-12    Austria,281488,246184
-9     Belgium,285598,257603
0      Bulgaria,59491,58919
12     Croatia,44331,49663
...
```

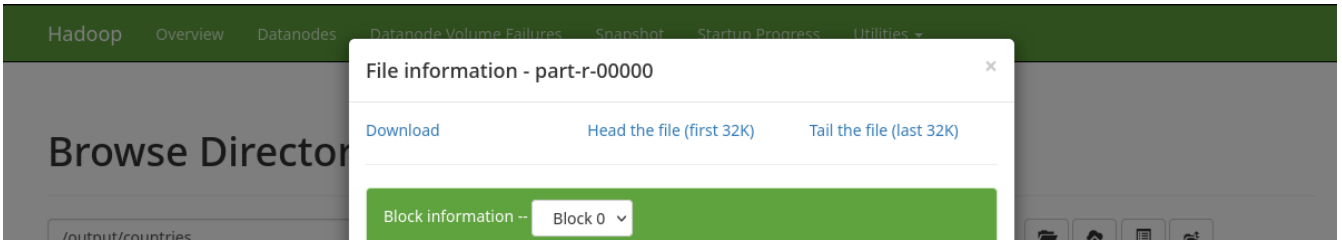
> *(Hier entspricht der Key dem ganzzahligen Prozentwert.)*

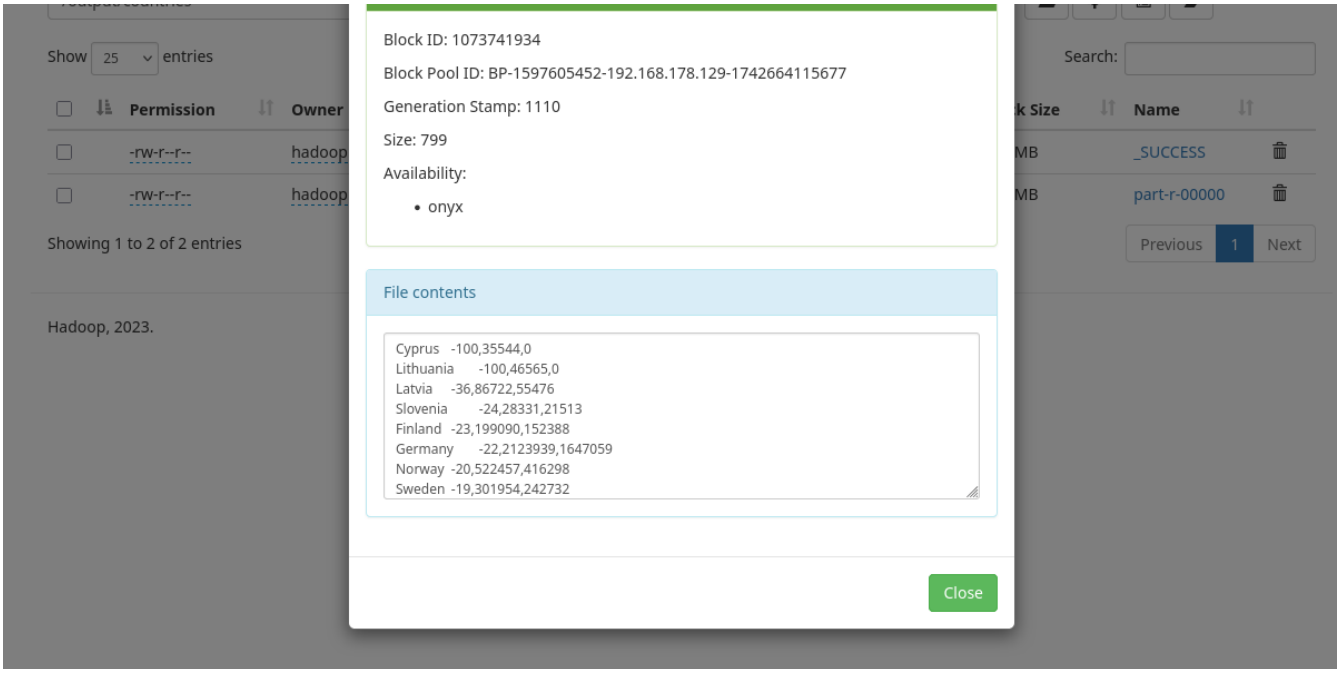
2.4 CPercentageReducer

- **Input**: Wertegruppen, die nach Key (Prozentwert) sortiert eintreffen.
- **Funktionsweise**:
  - Nimmt den Prozentwert aus dem Key.
  - Liest das Land und die beiden Jahreswerte aus dem Value.
  - Gibt das Land als **Key** und `<percentChange>,<aggregated2019>,<aggregated2024>` als **Value** aus.
- **Ausgabe part-r-00000**:

```
Cyprus          -100,35544,0
Lithuania      -100,46565,0
Latvia           -36,86722,55476
Slovenia       -24,28331,21513
...
```

Diese finale Liste der Länder, sortiert nach prozentualer Veränderung, wird in **/output/countries/** hinterlegt.





---

3. Airports

Hier geht es um die **prozentuale Veränderung** der Flugbewegungen **pro Flughafen**. Zusätzlich soll im Endergebnis auch das zugehörige Land ausgegeben werden.

3.1 APercentageMapper

- **Input**: Wieder das Preprocessing-Ergebnis (Key = `airportCode`, Value = `<stateName>,<total2019>,<total2024>`).
- **Funktionsweise**:
  - Parst die beiden Werte für 2019 und 2024.
  - Berechnet die prozentuale Änderung.
  - Gibt diese als **Key** (Integer-Prozentwert) aus und fügt `<airportCode>,<stateName>,<total2019>,<total2024>` in den Value.
- **Ausgabe**:

```
-15   EBBR,231172,196044,Belgium
13    EBCI,54426,61559,Belgium
112   EDDB,89374,190120,Germany
-48   EDDC,19753,10089,Germany

...
```

3.2 APercentageReducer

- **Input**: Key = prozentuale Änderung (Integer), Values = Listen von `<airportCode>,<stateName>,<total2019>,<total2024>`.
- **Funktionsweise**:
  - Liest den Prozentwert aus dem Key.
  - Trennt die Werte aus dem Value.
  - Gibt den `airportCode` als **Key** und `<percentChange>,<total2019>,<total2024>,<stateName>` als **Value** zurück.
- **Ausgabe part-r-00000**:

```
ENSG    -100,2852,0,Norway
EDDT    -100,191591,0,Germany
EGCC    -100,202883,0,United Kingdom
ENAL    -100,8128,0,Norway
...
```

Damit entsteht in **/output/airports/** eine Liste aller Flughäfen, sortiert nach prozentualer Veränderung.