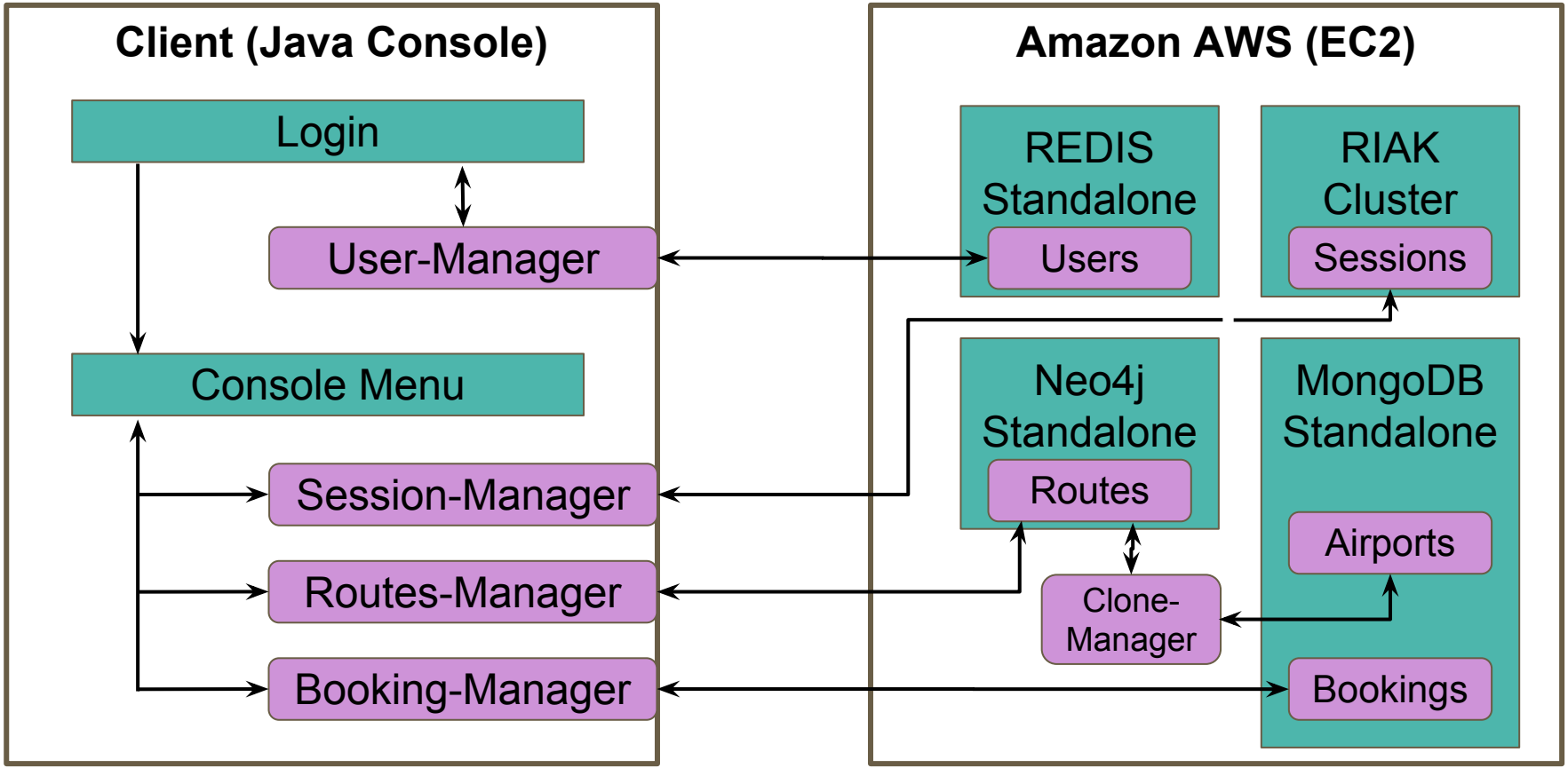
NoSQL Use Case: Airline Reservation System

Florian Muchow
Miro Conzelmann
Yousof Sagr Shaladi

Architecture & Features



Console Menu Structure

```
Mainmenu
        M2 (List all bookings)
                M2.1: List all bookings for the current user from MongoDB
                M2.2: Show booking details of a particular booking
        M3 (Create a new booking based on several settings)
                M3.1: Setting: destination airport (fetch airports from MongoDB)
                M3.2: Setting: departure airport (fetch airports from MongoDB)
                M3.3: Setting: maximum of travel time in hours (relevant for graph traverse in Neo4j)
                M3.4: Setting: possible routes (based on the previous settings; fetched in Neo4j)
        M4 (Reports/MapReduce)
                M4.1: Select a given report
                M4.2: Show the results of the report: number of logins per month
```

REDIS: User-Manager

Task: Validation of user credentials

Implementation:

- Server side: REDIS standalone server (EC2-instance)
- Client side: JAVA-connector via lettuce (version 3.2)
- Example:

```
RedisConnection<String, String> connection = redisClient.connect();
connection.set(newUserID, password);

password = connection.get(userID);
```

Creation of a new user

Get the password of an user

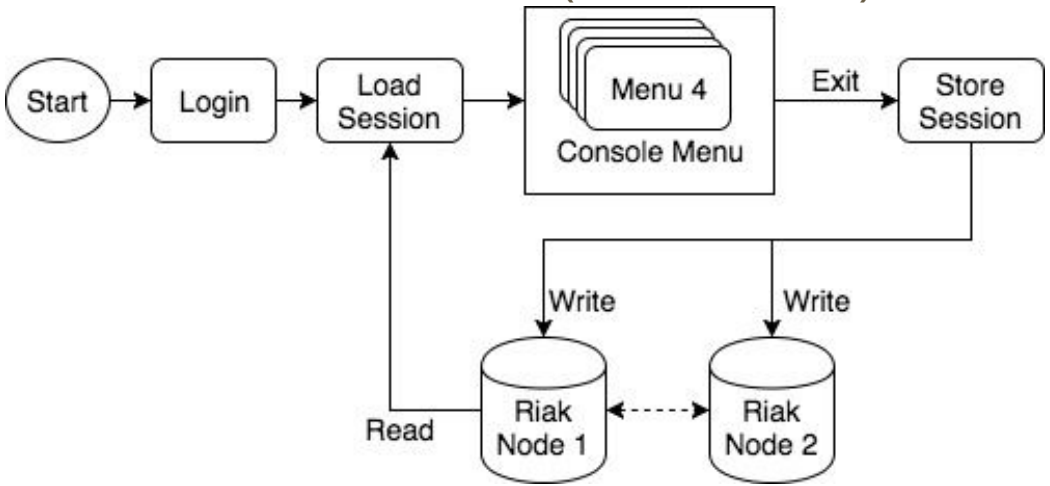
- Snapshot: Every 24 hours

RIAK: Session-Manager

Task: Managing the user sessions to reload/save the last GUI-state

Implementation:

- Server side: RIAK-cluster with two nodes (EC2-instance)
- Client side: JAVA-connector via basho-riak-client (version 2.1.1)
- Implementation and consistency by means of writes:



RIAK: MapReduce based on session data

Task: Report to count the logins per month (session-attribute: date)

Implementation:

- On the server side:
  - Execute the map- and reduce-function based on the date-attribute
- On the client side:
  - Execute a ruby script
  - The result will be processed in JAVA

```
The analysis showed the following result:
01.2017: 94
02.2017: 73
03.2017: 86
04.2017: 76
05.2017: 82
06.2017: 94
```

We have created at least 600 test users for the report

MongoDB: Airport-Manager

Task: Permanent Storage and Management of Airport Information in Mongo

Implementation:

- Storage: Document Collection “Airports” in “ProjectDB” on Server
  - {Name,X-Coordinate,Y-Coordinate,Destinations}
- Client: Java Class “Airports”
  - Methods:
    - constructor sets up Connection to Server and sets MongoClient
    - addAirport(name,X,Y,Dest)
    - getAllAirports()
    - closeConnection()
    - reset()

MongoDB: Booking-Manager

Task: Permanent Storage and Management of Bookings in Mongo

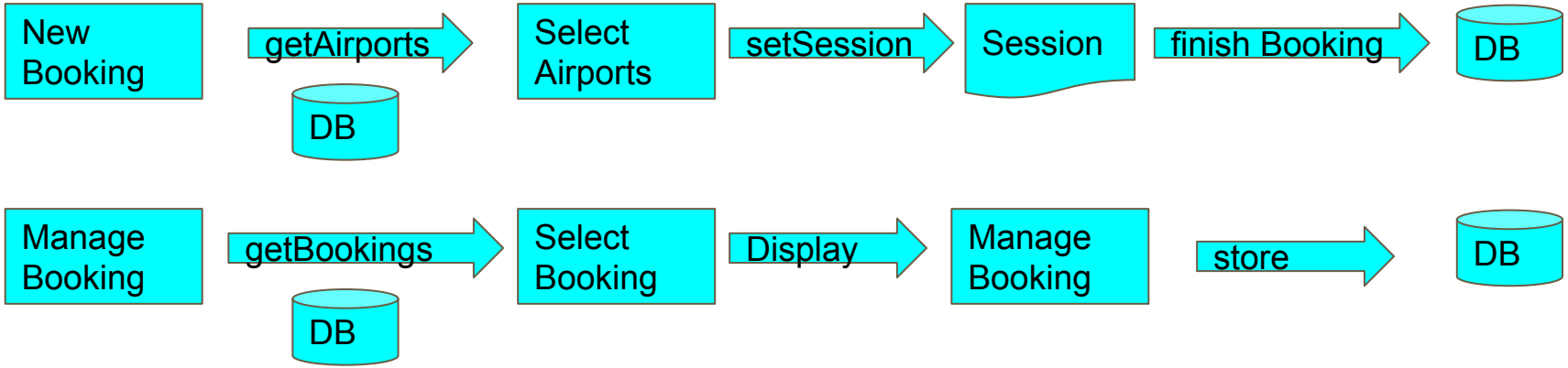
Implementation:

- Storage: Document Collection “Bookings” in “ProjectDB” on Server
  - {Username,DepartureTime,Landingtime,DepartureAirport,DestinationAirp,Stops,Bookingtime,Price}
- Client: JavaClass Bookings
  - methods
    - getBookingsbyName(Username)
    - addBooking(SessionObject)
    - cancelBooking(timeStamp)
    - reset(),closeConn()

MongoDB: Booking-Manager(Console)

Task: Display Airports, make Bookings and Manage Booking

Implementation:



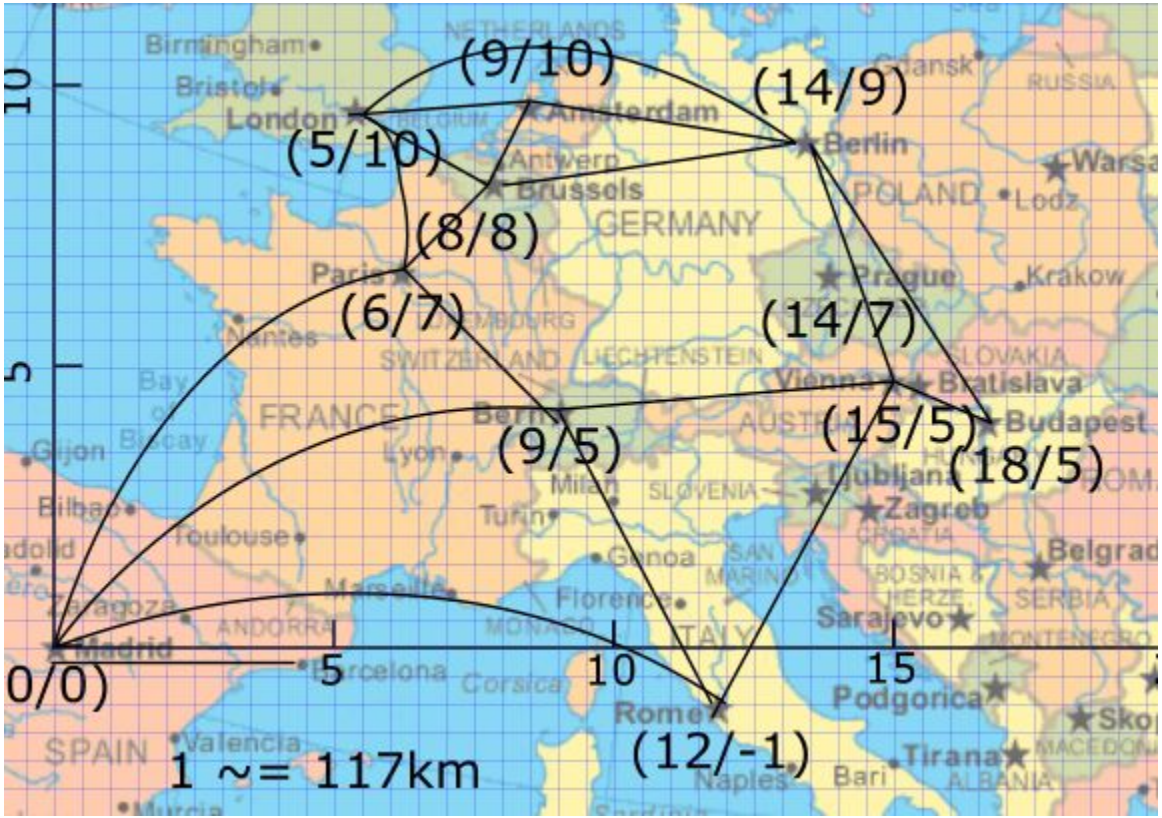
MongoDB: Aggregation

Task: Aggregate all Bookings

Implementation:

- Call: Bookings.aggregateBookings()
- Simple Aggregation Output of all Booking prices

Neo4j: The Scenario



Neo4j: Airports

Task: Create all necessary airports

Ideas:

- Airports are represented through nodes
- These nodes can contain properties (for example public transportation)
- Further, the nodes need to be connected

Solution: Cypher query language for node creation

Implementation:

- Create array of Airports
  - String[] Airports..
- Loop over Array and add properties:
  - for(...) .. session.run("CREATE AIRPORTS" ← Cypher Statement)
- If necessary, extensible though not very flexible

Neo4j: Airport Connections

Task: Create all necessary airport connections.

Ideas:

- Edges connect airport nodes
  - Relationship “flighsTo”
- Connections are undirected
- Edges contain a weighting representing the distance
- Assumption:
  - Each airport can be reached from any airport

Solution: Cypher query language for edge creation

Implementation:

- Create 2d-array of start and destination airports
  - String[] Airports..
- Loop over 2d-Array and add connections:
  - for() .. session.run(Cypher Statement → “CREATE AIRPORTS…”)
- If necessary, extensible though not very flexible

Neo4j: Routing

Task: Find the fastest connections between 2+ airports

Ideas:

- Airport connections have a distance
- Get routes between airports, sort by shortest distance
- Include any layovers in flight path and calculate total price

Solution: Cypher query language for path calculation

Implementation:

- After some research and idea gathering...
- Used the allShortestPath function + UNWIND statement to extract all airports
    - Returns complete route and price based on distance

Neo4j: Example Code

Airport Connections:

```
for(int i = 0; i < airport_start.length; i++) {
   for(int j = 0; j < airport_end[i].length; j++) {
      result = session.run

("match (a:" + airport_start[i].toString().toLowerCase() + "), (b:" +
                    airport_end[i][j].toString().toLowerCase() +") where
    a.Airportname = '" + airport_start[i].toString() + "'
          and b.Airportname = '" + airport_end[i][j].toString() + "'
create (a)-[r:flighsTo {weight:" +  weighting[i][j] +"}]->(b) return r");
   }
}
```

Airport Routing:

```
shortestPath = session.run("MATCH (a:" + startAirport.toLowerCase()
+ "), (b:" + destinationAirport.toLowerCase() + "),
p = allShortestPaths((a)-[r*1..5]-(b)) UNWIND rels(p) AS rel RETURN
extract(n IN nodes(p) | n.Airportname) AS NODES,
toInteger(sum(rel.weight*100*0.5)) " +
"AS PRICE ORDER BY PRICE", parameters(startAirport,
destinationAirport));

        this.storeList = this.storeList(shortestPath);
        System.out.println(this.storeList);

        for (Record record : storeList)
        {
          preise.add(Integer.valueOf(record.get("PRICE").toString()));
          routen.add(record.get("NODES").toString());
        }
```

Airline Reservation System

Livedemo

Airline Reservation System

Backup

MapReduce (RIAK): Ruby Script

```ruby
require 'riak'

# create connection to RIAK via Ruby
client = Riak::Client.new(:nodes => [
    {:host => '13.58.101.13', :pb_port => 8087},
    {:host => '13.59.15.196', :pb_port => 8087}
 ])
mapred = Riak::MapReduce.new(client)

#select RIAK-bucked-list
mapred.add('sessions')

# Map-function of MapReduce
# Aggregate date information
mapFunction = <<-EOF
function (v) {
    var parsed_data = JSON.parse(v.values[0].data);
    var data = {};
    data[parsed_data.dateAsInt] = 1;
    return [data];
    }
EOF

# Reduce-function of MapReduce
reduceFunction = <<-EOF
function(v) {
    var totals = {};
    for (var i in v) {
        for (var dateAsInt in v[i]) {
            if (totals[dateAsInt]) {
                totals[dateAsInt] += v[i][dateAsInt];
            }
            else {
                totals[dateAsInt] = v[i][dateAsInt];
            }
        }
    }
    return [totals];
}
EOF

# Execute MapReduce based on the date information
results = mapred.map(mapFunction).reduce(reduceFunction, :keep => true).run

# Sort the result
puts results[0].sort
```