Technologies used:

Hadoop (In pseudo-distributed mode)	Hadoop 2.7.7
Java	java version "1.8.0_201" Java(TM) SE Runtime Environment (build 1.8.0_201-b09) Java HotSpot(TM) 64-Bit Server VM (build 25.201-b09, mixed mode)
Spark (pyspark shell is used)	Version 2.4.4
Python	Version 3
Flask	Used as framework server
Ubuntu	18.04.1 LTS

Application Architecture:

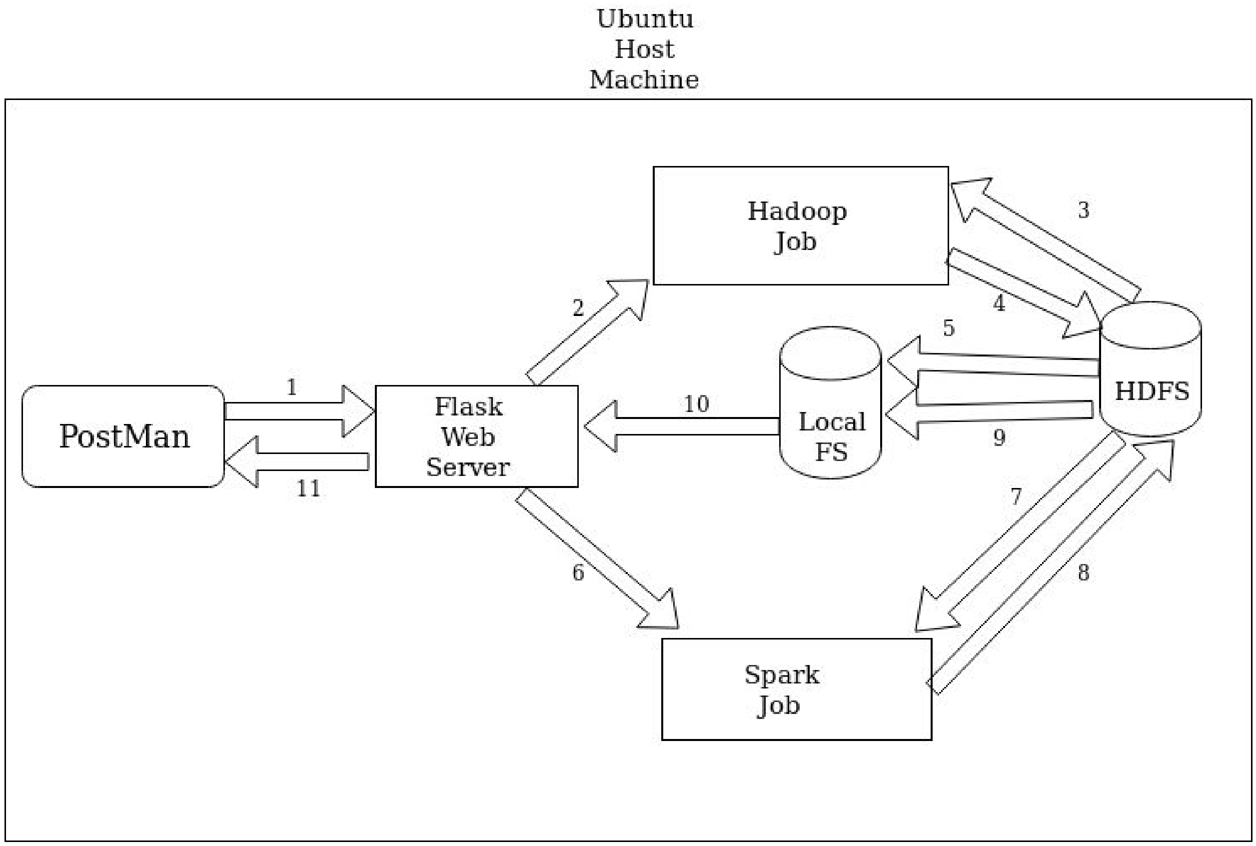


fig. Application Architecture

Description of Application Architecture:

1) Client sending request to Flask Web Server
2) Flask server forwarding the request to Hadoop Job
3) Hadoop Job reading the data from HDFS(Hadoop File System)
4) Hadoop Job writing the data to HDFS
5) Data gets written to local File System
6) Flask server forwarding the request to Spark Job
7) Spark Job reading the data from HDFS
8) Spark Job writing the data to HDFS
9) Data gets written to local File System
10) Response being returned to Flask Web Server
11) Client getting the final output

The Software Application consists of 3 Modules

1. Restful Web Service Module
2. Hadoop Map-Reduce Module
3. Spark Transformation-Action Module

All the modules are explained in detail below:-

Restful Web Service

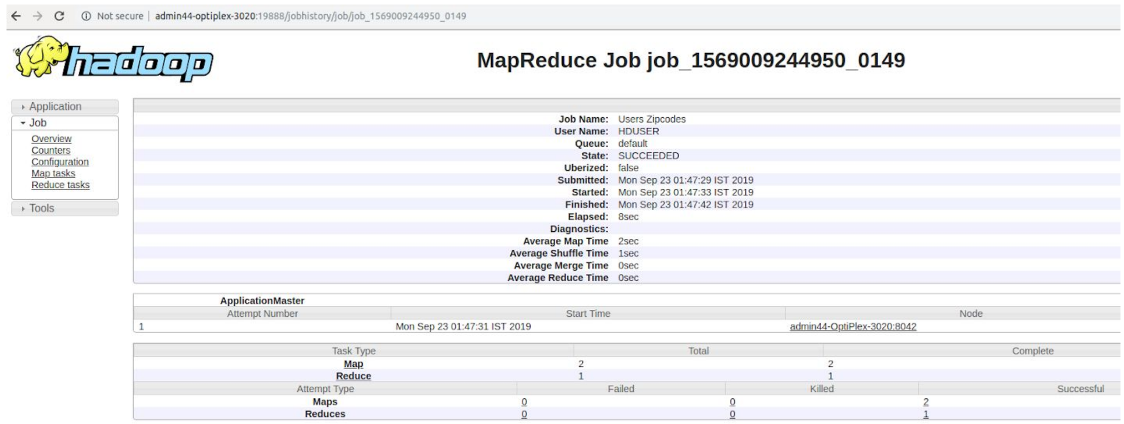
Web Service Configuration:

URI : http://localhost:6006/inputString
METHOD:POST
BODY:

```
{
        "query" : "  Join Query or Group By Query  "
}
```

Sequence of operations at server

1. Query is provided to the server through the above URI using any Web Service Tool such as Postman
2. After the Query String is received at the **Flask Web Server**,it is **parsed** and classified as **Join Query or Group By Query**
3. Next, all the parameters of the query are extracted and saved at the **designated data structures** at the server and **Hadoop(Function-1)** and **Spark(Function-2)** operations are applied in **sequence**.
4. **Function 1 - Hadoop**
    a. First server checks if the hadoop **Name Node** and **Data Nodes** are up or not at **http://localhost:9000**
    b. Next based on the query, server picks up the designated jar and executes it with hadoop along with various **query parameters** extracted previously in step 3.
    c. After the hadoop job execution, server opens the **YARN Resource Manager** running at **localhost:8088** and scrapes the **latest job ID** of the recently run jobs



d. Now as the server has the Job ID for Hadoop Job it opens the URL **http://localhost:19888/jobhistory/job_<job_id>** and scrapes all the **Hadoop Execution Log Parameters** such as Total Elapsed Time, Average Map TIme,Average Reducer TIme etc..

e. All the fetched execution parameters are returned

5. **Function 2 - Spark**
    a. Based on the query, server picks up the designated python script and execute it along with various **query parameters** extracted previously in step 3
    b. All the **Spark execution log parameters** are fetched based on the OS Timestamp.

6. Now the HDFS file system is read which was previously written by Hadoop and Spark Jobs for the required query output.
7. The Query output along with execution parameters are returned to the client.

Hadoop Map-Reduce

Query 1:

Join Template:

SELECT * FROM <TABLE1> INNER JOIN <TABLE2> ON <CONDITION1>

WHERE <CONDITION2>

Example:

SELECT * FROM USERS INNER JOIN zipcodes ON zipcode

WHERE userid = 780

Description:

For this query we are operating on two tables. One is user table and other is zipcodes table. We are first splitting the user table as key-value pair where **zipcode** is selected as **key** and remaining record is selected as value. For zipcode table, we are selecting **zipcode** as **key** and remaining record is selected as value. After Mapping and Filtering,we are getting Mapper output as:-

i) For user table we are getting records in the form

<zipcode> , <userid, age, gender, occupation, zipcode>

ii) For zipcode table we are getting records in the form

<zipcode>,<zipcode,zipcodetype,city,state>

All the records with the same key (zipcode here) are Reduced to to the same Reducer and joined to form the resultant join response.

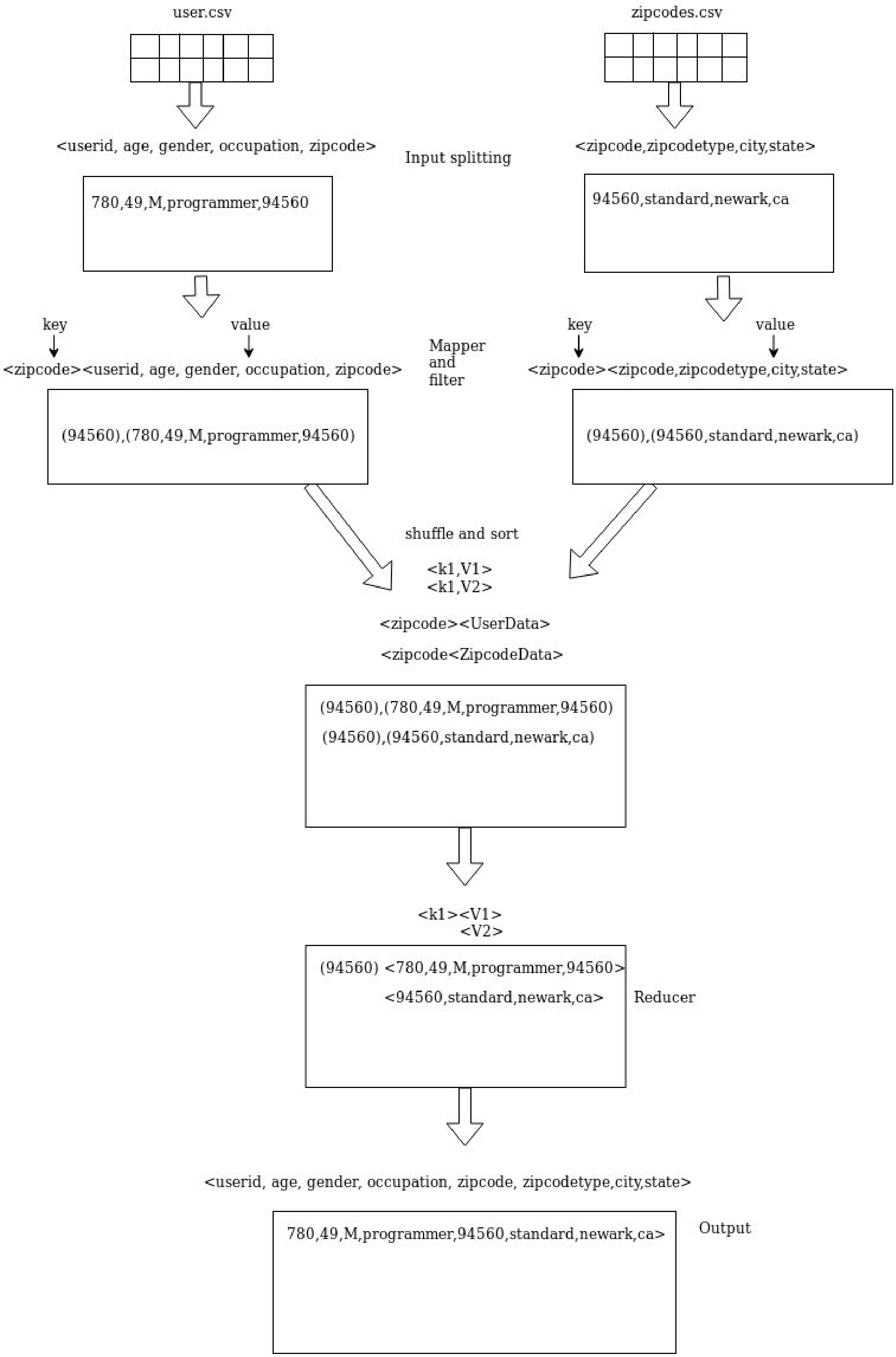

user.csv
zipcodes.csv
<userid, age, gender, occupation, zipcode>
Input splitting
<zipcode,zipcodetype,city,state>
780,49,M,programmer,94560
94560,standard,newark,ca
key
value
<zipcode><userid, age, gender, occupation, zipcode>
Mapper
and
filter
key
value
<zipcode><zipcode,zipcodetype,city,state>
(94560),(780,49,M,programmer,94560)
(94560),(94560,standard,newark,ca)
shuffle and sort
<k1,V1>
<k1,V2>
<zipcode><UserData>
<zipcode<ZipcodeData>
(94560),(780,49,M,programmer,94560)
(94560),(94560,standard,newark,ca)
<k1><V1>
<V2>
(94560) <780,49,M,programmer,94560>
<94560,standard,newark,ca>
Reducer
<userid, age, gender, occupation, zipcode, zipcodetype,city,state>
780,49,M,programmer,94560,standard,newark,ca>
Output

fig. Data Flow Architecture of Map-Reduce: Query 1

Query 2:

Group By Template:

SELECT <COLUMNS>, FUNC(COLUMN1)

FROM <TABLE>

GROUP BY <COLUMNS>

HAVING FUNC(COLUMN1)>X

Example:

SELECT gender, occupation, MAX(age)

FROM users

GROUP BY gender,occupation

HAVING MAX(age) > 45

Description:

For this query we are operating on user table. Once we import the input, we are then choosing **gender and occupation as key** and the rest of the record is chosen as **value** .Therefore for user table we are getting records in the form (**<gender, occupation>,<userid, age, gender, occupation, zipcode>**). We are combining the records using “shuffle and sort”. Now values are ordered according to key. We are then using reducer. As in the “having” clause, we are selecting only those users who have their age greater than 50. Once reducer acts according to logic supplied we are able to get final output.

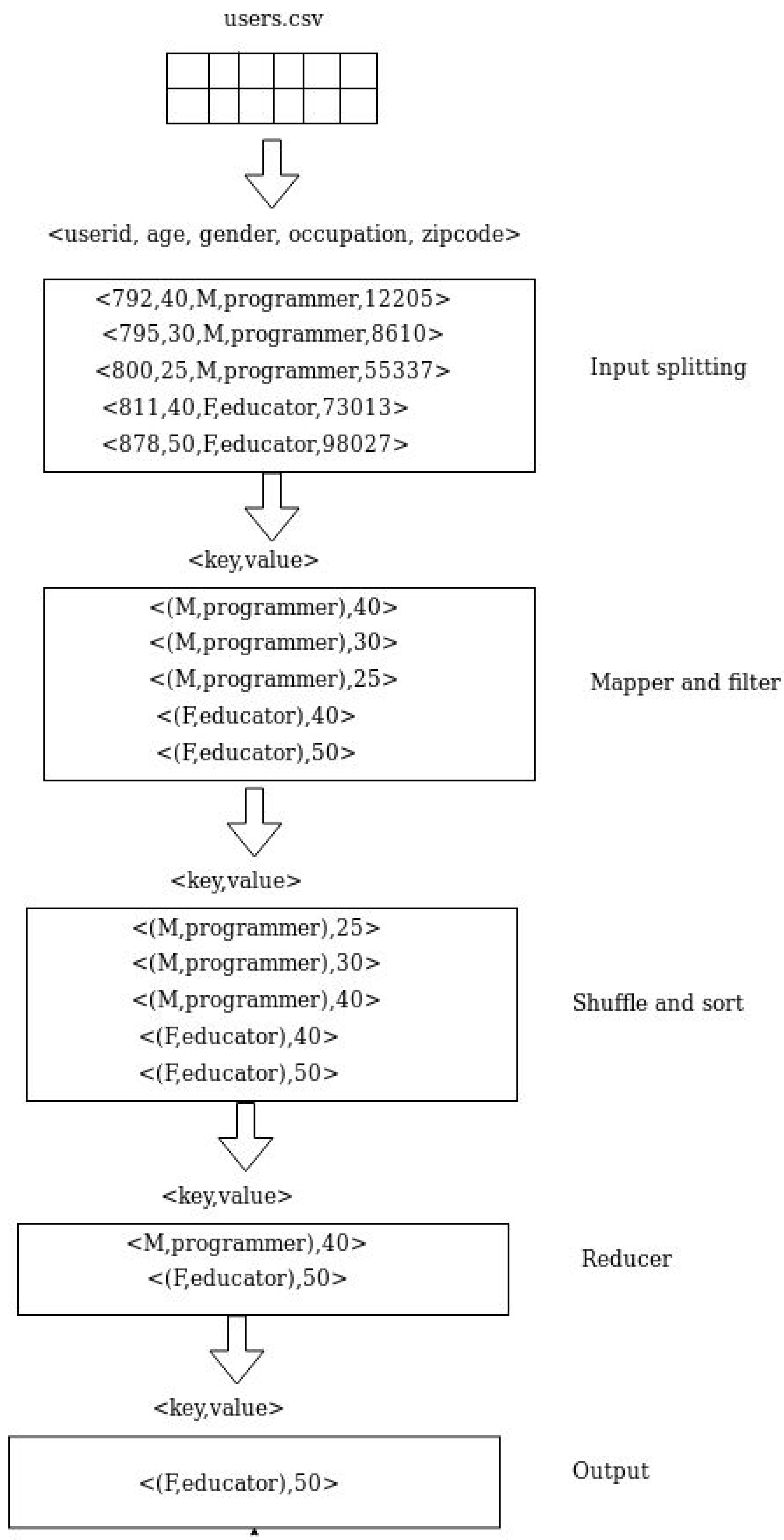

users.csv
<userid, age, gender, occupation, zipcode>
<792,40,M,programmer,12205>
<795,30,M,programmer,8610>
<800,25,M,programmer,55337>
<811,40,F,educator,73013>
<878,50,F,educator,98027>
Input splitting
<key,value>
<(M,programmer),40>
<(M,programmer),30>
<(M,programmer),25>
<(F,educator),40>
<(F,educator),50>
Mapper and filter
<key,value>
<(M,programmer),25>
<(M,programmer),30>
<(M,programmer),40>
<(F,educator),40>
<(F,educator),50>
Shuffle and sort
<key,value>
<M,programmer),40>
<(F,educator),50>
Reducer
<key,value>
<(F,educator),50>
Output

fig. Data Flow Architecture of Map-Reduce: Query 2

Spark Tranformations-Actions

Query 1:

Join Template:

SELECT * FROM <TABLE1> INNER JOIN <TABLE2> ON <CONDITION1>

WHERE <CONDITION2>

Example:

SELECT * FROM users INNER JOIN rating ON userid

WHERE zipcode = 30067

Description

Once we start pyspark shell, sparkContext object (sc) is available for us. We could use this object for further processing. As we know Spark framework provides functions for basic transformations and actions, to carry out this query we have used the following **transformations:**

map()	We have used map function to transform user table into key- value pair. First field is selected as key, and the rest of the fields are treated as values. We have done a similar thing for rating table.
join()	We are using this transformation function to join user table and rating table. As this function requires two datasets to be in the form of (K,V) pair, we have first converted user table and rating table by using map function and then we are applying join function over them.
filter()	To execute where condition in the template, we are using filter function. We are applying

	filter function to the joint table of user and rating (which we obtained in the previous step). To filter function we are providing logic to select those records which have zipcode as "30067".

We have used the following **actions** in this task:

collect()	To print the result of the query, we are using this action provided by spark framework. We are calling collect function on the dataset (RDD in this case) that we need to print.
take()	Sometimes we are just verifying outputs of intermediate steps. Now instead of printing entire result, we are using take function provided by spark framework.
saveAsTextFile()	At the end we are saving final output of the query in a file. For this we are using this function provided by spark framework.

First on user table we are applying **Map** function. We are selecting first field which is **userid as key** of the table. Map function will **transform** user table as

<userid> ,<userid, age, gender, occupation, zipcode>

Similarly we are selecting first field of rating table i.e. **userid as key** of the table. Map function is applied on rating table and table will be transformed to key value pair where key is **userid** and value is (**userid, movieid, rating, timestamp**).

Next we are applying join function on user table and rating table. Condition for join is userid. Now both tables are concatenated. We are then applying filter function on big table. We are selecting users who have zip code as "30067". We are then using "saveAsTextFile" transformation function to save the RDD in HDFS File System.

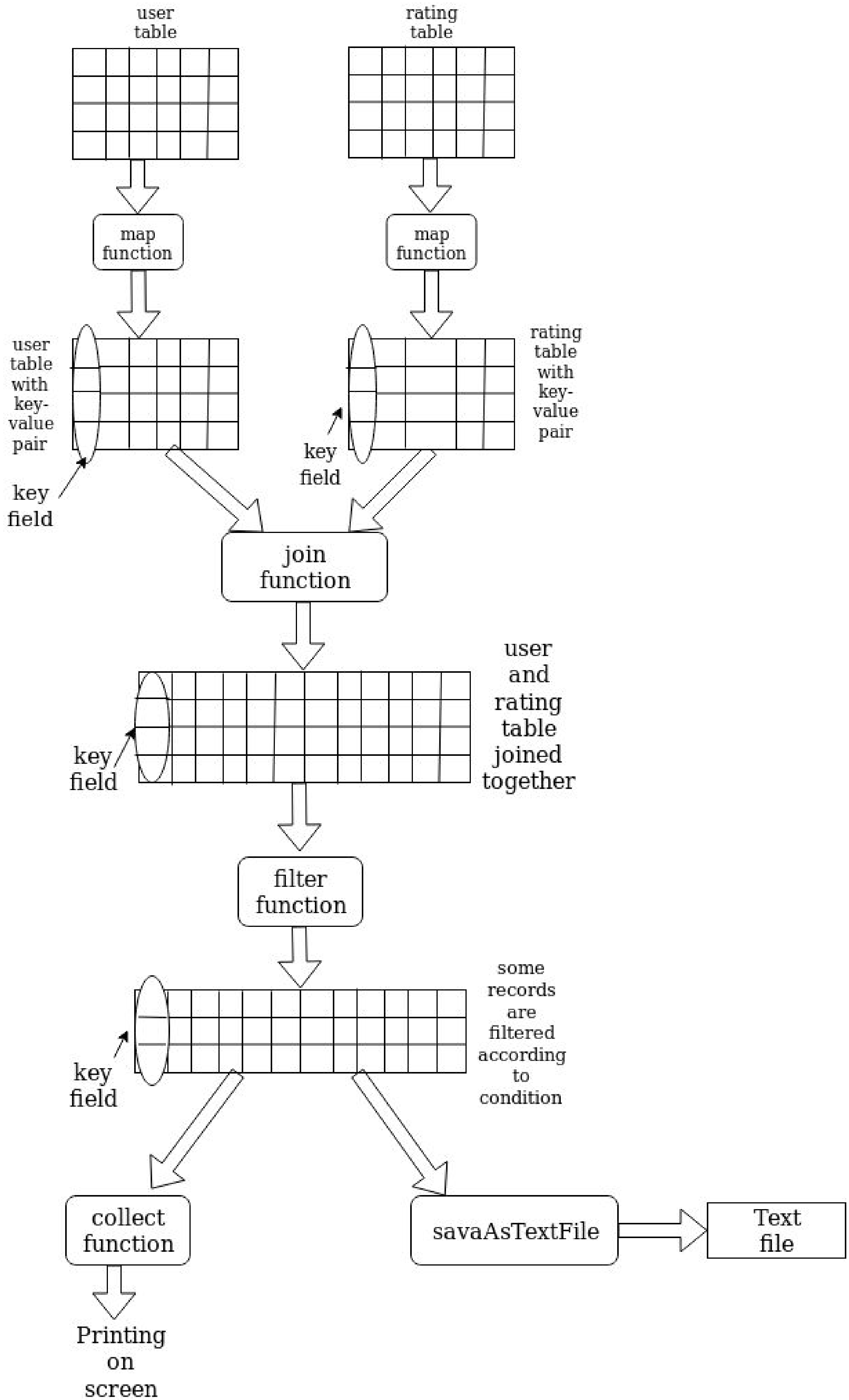

user table
rating table
map function
map function
user table with key-value pair
rating table with key-value pair
key field
key field
join function
user and rating table joined together
key field
filter function
some records are filtered according to condition
key field
collect function
savaAsTextFile
Text file
Printing on screen

Fig. Data Flow Architecture of Spark: Query 1

Query 2:

Group By Template

SELECT <COLUMNS>, FUNC(COLUMN1)

FROM <TABLE>

GROUP BY <COLUMNS>

HAVING FUNC(COLUMN1)>X

Example:

SELECT occupation,gender, SUM(age)

FROM users

GROUP BY occupation,gender

HAVING SUM(age) > 1000

Description

We have used following **transformations** for this task:

map()	We have used map function is to transform user table into key- value pair. Here we are taking multiple columns as key as per given in group by clause.
filter()	As we need to select the records according to condition in having clause, we are using filter transformation for this purpose.
groupByKey()	Once we get the columns which are supposed to by keys, we are using groupByKey() transformation, so that data for that particular key could be aggregated.

We have used the following **actions** for this task:

collect()	To print the result of the query, we are using this action provided by spark framework. We are calling collect function on the dataset (RDD in this case) that we need to print.
take()	Sometimes we are just verifying outputs of intermediate steps. Now instead of printing entire result, we are using take function provided by spark framework.
saveAsTextFile()	At the end we are saving final output of the query in a file. For this we are using this function provided by spark framework.

First on **user** table we are applying **Map** function. As, in group by clause, we have gender and occupation, we are using those fields as key. Map function will transform user table to key value pair as

<gender, occupation>,<userid, age, gender, occupation, zipcode>

We are now applying "groupByKey" function on user table. It returns a dataset of (K, Iterable<V>) pairs.

Now we are iterating over that dataset, finding the sum of ages of users (as they are already grouped by gender and occupation) and mapping it to a new RDD. Now we are applying filter function where condition is mentioned as per the having clause. We are selecting users whose sum of the ages is greater than 1000. We are then using "saveAsTextFile" function to save the output in file.

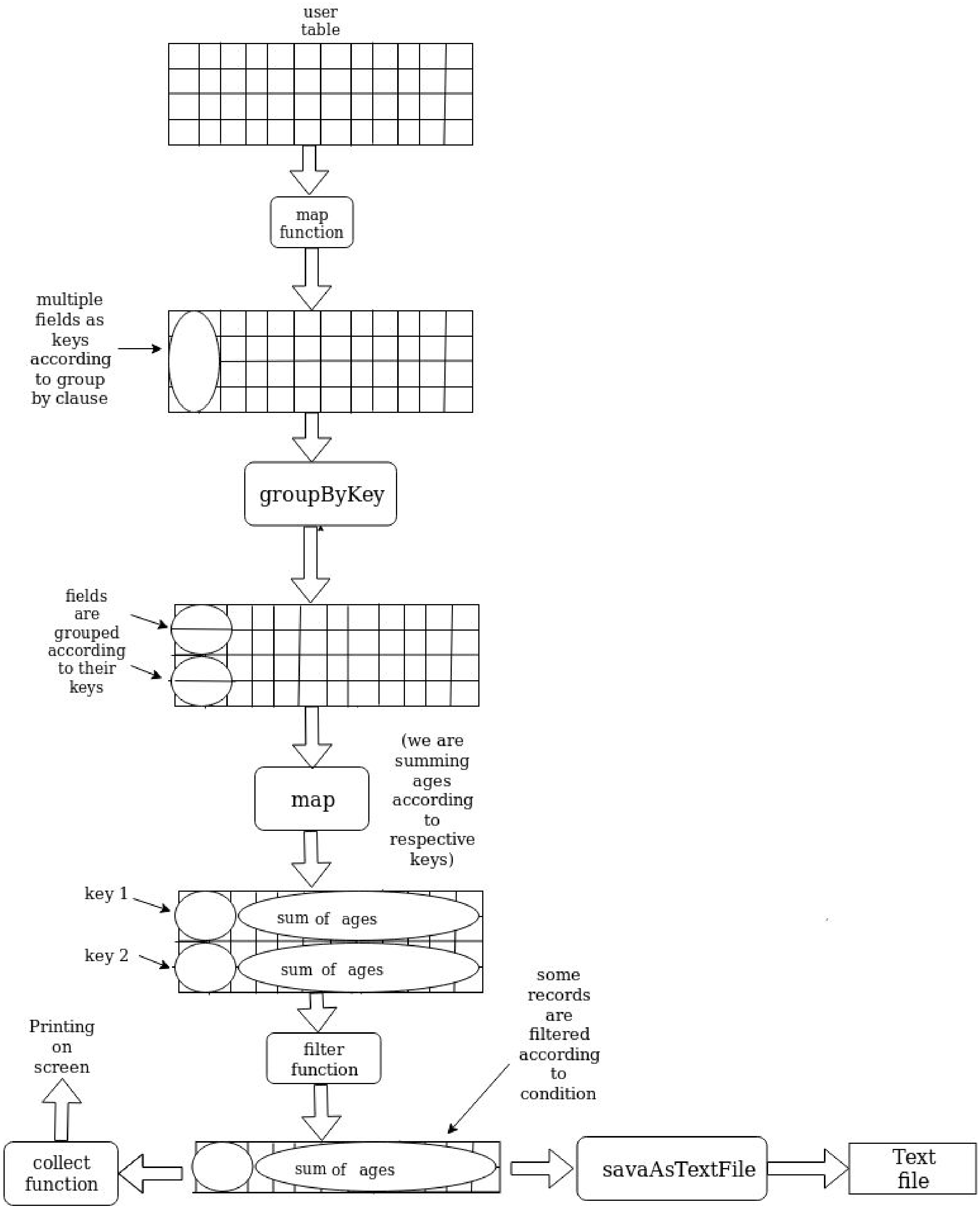


fig. Data Flow Architecture of Spark: Query 2

Outputs

Query 1

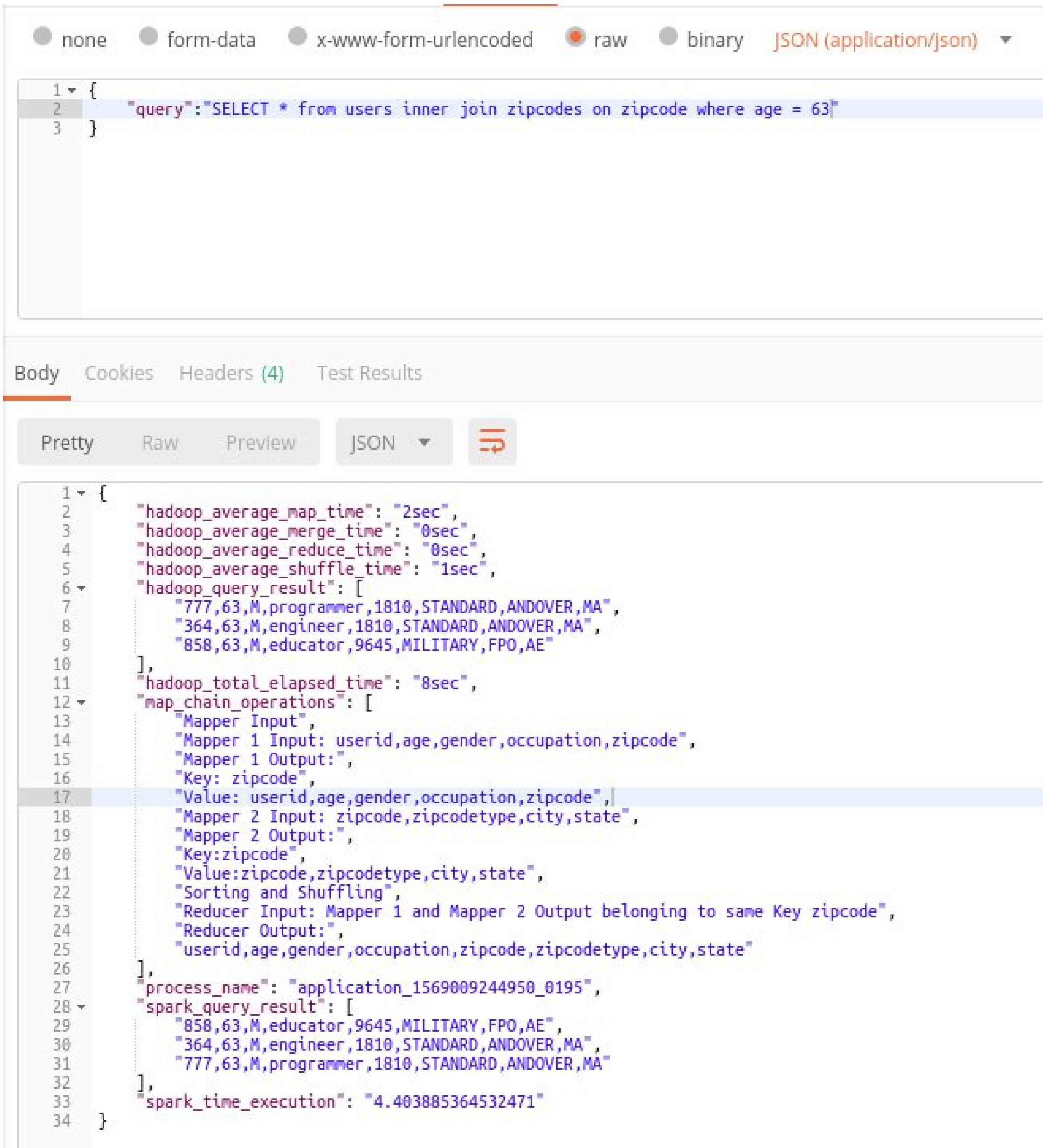


fig. Input Query Request to Server and Response from Server

Query 2

fig.Input Query Request to Server

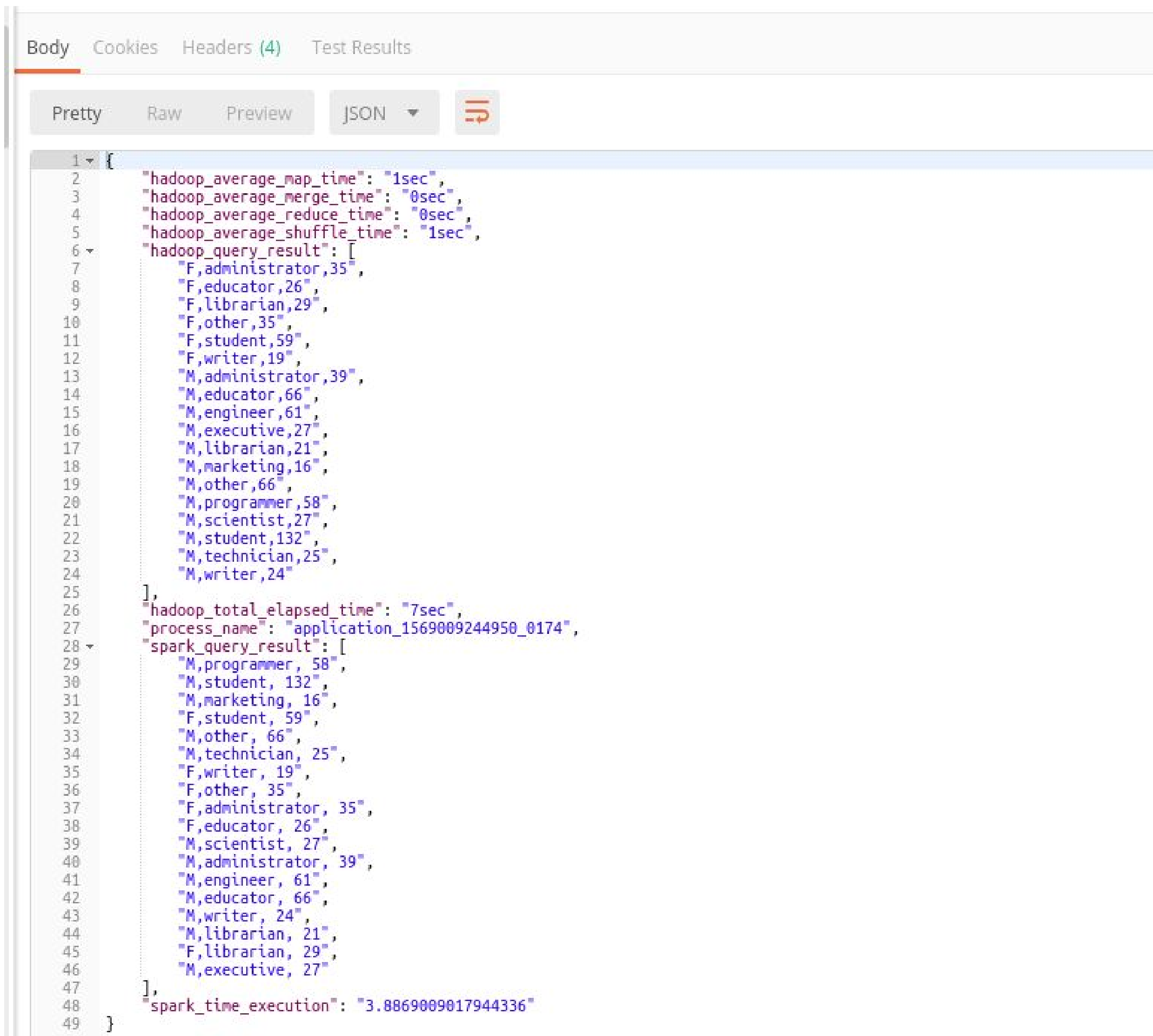


fig. Response from Server