



МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ  
НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ  
“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ  
імені ІГОРЯ СІКОРСЬКОГО”

Факультет прикладної математики  
Кафедра програмного забезпечення комп'ютерних систем



**Лабораторна робота №6**

з дисципліни: «Технології оброблення великих даних»

на тему: «Розподілені обчислення даних з використанням Spark-кластера у  
середовищі R»

Виконав

студент III курсу каф.  
ПЗКС ФПМ

групи КП-82

Мельничук Олексій  
Геннадійович

Перевірила

доц. каф. ПЗКС ФПМ

Олещенко Л.М.

Київ 2021

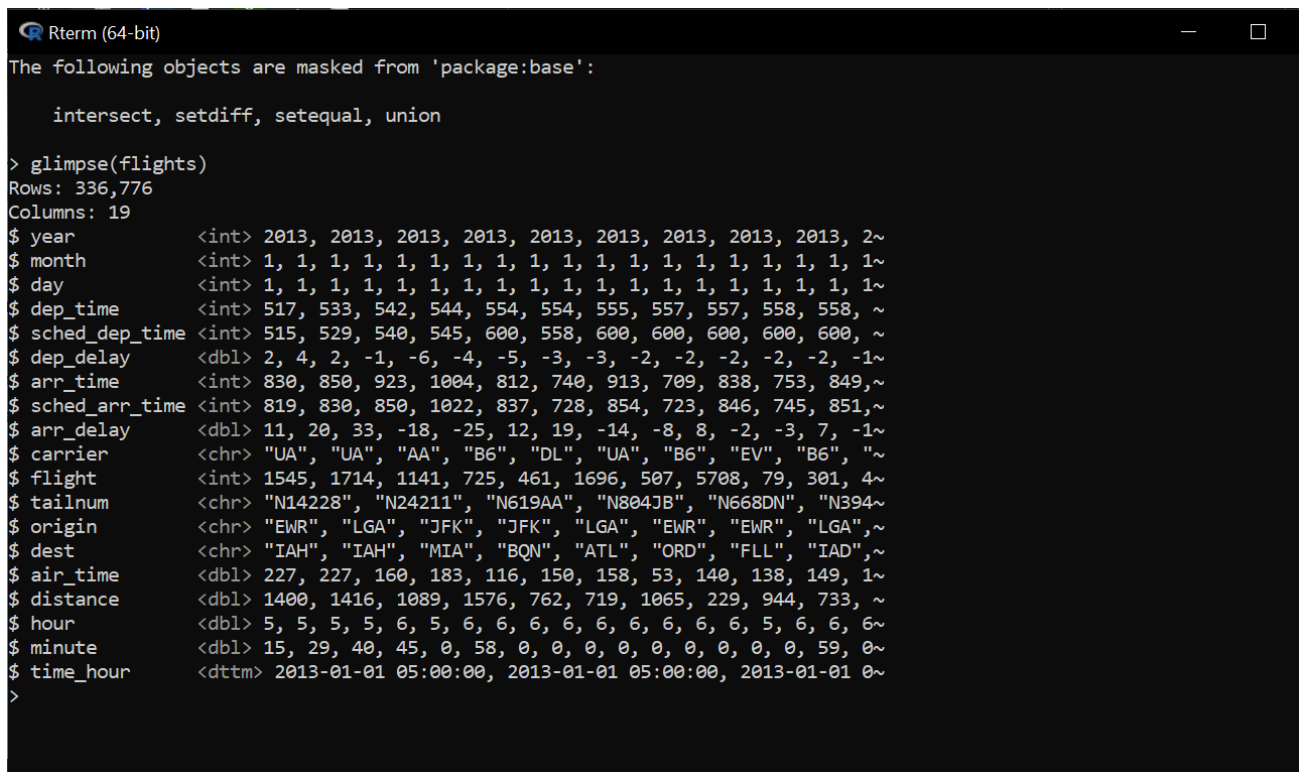
## 1. Індивідуальне завдання

**Мета:** встановити Spark на локальній машині, ознайомитись з можливостями розподілених обчислень для великих даних з використанням Spark-кластера у середовищі R.

## 2. Хід роботи

### Частина 1: Встановлення та підключення до локального Spark-кластеру

Для лабораторної роботи використаємо дані одного з багатьох пакетів даних, пакета `nycflights13`, який містить кілька таблиць з описом 336776 авіарейсів, які стартували з аеропортів Нью-Йорка в 2013 р. Встановіть пакет `nycflights13`



```
Rterm (64-bit)
The following objects are masked from 'package:base':
  intersect, setdiff, setequal, union

> glimpse(flights)
Rows: 336,776
Columns: 19
$ year      <int> 2013, 2013, 2013, 2013, 2013, 2013, 2013, 2013, 2013, 2~
$ month     <int> 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1~
$ day       <int> 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1~
$ dep_time  <int> 517, 533, 542, 544, 554, 554, 555, 557, 557, 558, 558, ~
$ sched_dep_time <int> 515, 529, 540, 545, 600, 558, 600, 600, 600, 600, 600, ~
$ dep_delay <dbl> 2, 4, 2, -1, -6, -4, -5, -3, -3, -2, -2, -2, -2, -2, -1~
$ arr_time  <int> 830, 850, 923, 1004, 812, 740, 913, 709, 838, 753, 849,~
$ sched_arr_time <int> 819, 830, 850, 1022, 837, 728, 854, 723, 846, 745, 851,~
$ arr_delay <dbl> 11, 20, 33, -18, -25, 12, 19, -14, -8, 8, -2, -3, 7, -1~
$ carrier   <chr> "UA", "UA", "AA", "B6", "DL", "UA", "B6", "EV", "B6", "~
$ flight    <int> 1545, 1714, 1141, 725, 461, 1696, 507, 5708, 79, 301, 4~
$ tailnum   <chr> "N14228", "N24211", "N619AA", "N804JB", "N668DN", "N394~
$ origin    <chr> "EWR", "LGA", "JFK", "JFK", "LGA", "EWR", "EWR", "LGA",~
$ dest      <chr> "IAH", "IAH", "MIA", "BQN", "ATL", "ORD", "FLL", "IAD",~
$ air_time  <dbl> 227, 227, 160, 183, 116, 150, 158, 53, 140, 138, 149, 1~
$ distance  <dbl> 1400, 1416, 1089, 1576, 762, 719, 1065, 229, 944, 733, ~
$ hour      <dbl> 5, 5, 5, 5, 6, 5, 6, 6, 6, 6, 6, 6, 6, 6, 5, 6, 6, 6~
$ minute    <dbl> 15, 29, 40, 45, 0, 58, 0, 0, 0, 0, 0, 0, 0, 0, 59, 0~
$ time_hour <dtm> 2013-01-01 05:00:00, 2013-01-01 05:00:00, 2013-01-01 0~
>
```

Для початку потрібно завантажити таблицю flights в локальний Spark-кластер. Для цього служить функція copy\_to (), на яку ми подаємо створений раніше об'єкт sc і копіюваний таблицю flights

```
> f1 <- copy_to(sc, flights)
> f1
# Source: spark<flights> [?? x 19]
   year month   day dep_time sched_dep_time dep_delay arr_time sched_arr_time
   <int> <int> <int>   <int>         <int>         <dbl>   <int>         <int>
1  2013     1     1    517           515           2     830           819
2  2013     1     1    533           529           4     850           830
3  2013     1     1    542           540           2     923           850
4  2013     1     1    544           545          -1    1004          1022
5  2013     1     1    554           600          -6     812           837
6  2013     1     1    554           558          -4     740           728
7  2013     1     1    555           600          -5     913           854
8  2013     1     1    557           600          -3     709           723
9  2013     1     1    557           600          -3     838           846
10 2013     1     1    558           600          -2     753           745
# ... with more rows, and 11 more variables: arr_delay <dbl>, carrier <chr>,
#   flight <int>, tailnum <chr>, origin <chr>, dest <chr>, air_time <dbl>,
#   distance <dbl>, hour <dbl>, minute <dbl>, time_hour <dtm>
>
```

Для виконання SQL-запиту потрібно пакет DBI і входить до його складу функція dbGetQuery (). Якщо ви використовували цей пакет раніше для роботи з віддаленими базами даних, то не зустрінете нічого незвичайного: на функцію dbGetQuery () потрібно подати об'єкт sc з інформацією для підключення до Spark- кластеру, а також сам запит:

```
> require(DBI)
Loading required package: DBI
> q <- "SELECT `origin`, count(*) AS `N`
+ FROM `flights`
+ GROUP BY `origin`"
> dbGetQuery(sc, q)
   origin      N
1    LGA 104662
2    JFK 111279
3    EWR 120835
>
```

Використання SQL майже потрібно при виконанні складних запитів, які повинні бути оптимізовані для забезпечення швидкодії. Однак в більшості випадків для виконання стандартних операцій над даними використання функцій з пакета dplyr буде більш зручним. Більш того, якщо ви раніше працювали з dplyr (а сьогодні навряд чи знайдуться користувачі R, які не знають про цей пакет), то ви не побачите майже ніяких відмінностей:

```
> result <- fl %>%
+ group_by(origin) %>%
+ summarise(N = n())
> result
# Source: spark<?> [?? x 2]
  origin      N
  <chr>    <dbl>
1 LGA      104662
2 JFK      111279
3 EWR      120835
> _
```

Для отримання даних з Spark в середу R слід скористатися функцією collect():

```
> true_result <- result %>% collect()
> true_result
# A tibble: 3 x 2
  origin      N
  <chr>    <dbl>
1 LGA      104662
2 JFK      111279
3 EWR      120835
> class(result)
[1] "tbl_spark" "tbl_sql"   "tbl_lazy"  "tbl"
> class(true_result)
[1] "tbl_df"     "tbl"       "data.frame"
> _
```

## Частина 2: Аналіз даних в Spark-кластері за допомогою пакета dplyr в R

Підрахуйте загальну кількість польотів, виконаних кожною авіакомпанією за 2013 р., а потім виберіть 5 авіакомпаній з найбільшим числом польотів:

```
Rterm (64-bit)
> airlines_tbl <- copy_to(sc, airlines, "airlines")
> require(dplyr)
Loading required package: dplyr

Attaching package: 'dplyr'

The following objects are masked from 'package:stats':
  filter, lag

The following objects are masked from 'package:base':
  intersect, setdiff, setequal, union

> flights_tbl %>%
+ group_by(carrier) %>%
+ summarise(N = n()) %>%
+ arrange(desc(N)) %>%
+ head(5)
# Source:      spark<?> [?? x 2]
# Ordered by: desc(N)
  carrier      N
  <chr>      <dbl>
1 UA        58665
2 B6        54635
3 EV        54173
4 DL        48110
5 AA        32729
> _
```

Функція `show_query()` з пакету `dplyr` дозволяє переглянути SQL-запит, який формується з відповідного коду R. Для наведеного вище прикладу отримуємо:

```

> flights_tbl %>%
+ group_by(carrier) %>%
+ summarise(N = n()) %>%
+ arrange(desc(N)) %>%
+ head(5) %>%
+ show_query()
<SQL>
SELECT `carrier`, COUNT(*) AS `N`
FROM `flights`
GROUP BY `carrier`
ORDER BY `N` DESC
LIMIT 5
>

```

## Об'єднання таблиць

Так, в наступному прикладі виконаний LEFT JOIN таблиці flights\_tbl з таблицею airlines\_tbl по полю carrier :

```

> flights_tbl %>%
+ left_join(airlines_tbl, by = "carrier") %>%
+ glimpse()
Rows: ??
Columns: 20
Database: spark_connection
$ year      <int> 2013, 2013, 2013, 2013, 2013, 2013, 2013, 2013, 2013, 2~
$ month     <int> 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1~
$ day       <int> 1, 1, 2, 2, 3, 3, 4, 4, 5, 5, 6, 6, 7, 7, 8, 8, 9, 9, 1~
$ dep_time  <int> 833, 1716, 827, 1728, 835, 1933, 834, 1831, 835, 1726, ~
$ sched_dep_time <int> 835, 1730, 835, 1730, 835, 1730, 835, 1730, 835, 1730, ~
$ dep_delay <dbl> -2, -14, -8, -2, 0, 123, -1, 61, 0, -4, -5, -8, 0, -7, ~
$ arr_time  <int> 1134, 1947, 1120, 1952, 1102, 2131, 1059, 2029, 1057, 1~
$ sched_arr_time <int> 1102, 1953, 1102, 1953, 1102, 1953, 1102, 1953, 1102, 1~
$ arr_delay <dbl> 32, -6, 18, -1, 0, 98, -3, 36, -5, -5, -7, -7, 4, 15, 1~
$ carrier   <chr> "F9", "F9", "F9", "F9", "F9", "F9", "F9", "F9", "F9", "~
$ flight    <int> 835, 511, 835, 511, 835, 511, 835, 511, 835, 511, 835, ~
$ tailnum   <chr> "N203FR", "N263AV", "N211FR", "N201FR", "N203FR", "N261~
$ origin    <chr> "LGA", "LGA", "LGA", "LGA", "LGA", "LGA", "LGA", "LGA", ~
$ dest      <chr> "DEN", "DEN", "DEN", "DEN", "DEN", "DEN", "DEN", "DEN", ~
$ air_time  <dbl> 257, 242, 239, 238, 219, 208, 220, 213, 237, 236, 236, ~
$ distance  <dbl> 1620, 1620, 1620, 1620, 1620, 1620, 1620, 1620, 1620, 1620, ~
$ hour      <dbl> 8, 17, 8, 17, 8, 17, 8, 17, 8, 17, 8, 17, 8, 17, 8, ~
$ minute    <dbl> 35, 30, 35, 30, 35, 30, 35, 30, 35, 30, 35, 30, 30, 30, ~
$ time_hour <dtm> 2013-01-01 13:00:00, 2013-01-01 22:00:00, 2013-01-02 1~
$ name      <chr> "Frontier Airlines Inc.", "Frontier Airlines Inc.", "Fr~
>

```

## Функції Hive Query Language

Наприклад, для обчислення медіанного значення змінної `dep_delay` (затримка рейсу, хв) з таблиці `flights_tbl` ми не можемо просто скористатися базовими функціями R `median()` або `quantile()` – це призведе до помилки. Але ми без проблем можемо застосувати Hive-функцію `percentile()`:

```
> flights_tbl %>%
+ summarise(median = percentile(dep_delay, 0.5))
# Source: spark<?> [?? x 1]
  median
  <dbl>
1      -2
> _
```

Коли при перекладі коду R на SQL `dplyr` зустрічає незнайому функцію, то він просто включає її в SQL-запит "як є":

```
> flights_tbl %>%
+ summarise(median = percentile(dep_delay, 0.5)) %>%
+ show_query()
<SQL>
SELECT percentile(`dep_delay`, 0.5) AS `median`
FROM `flights`
> _
```

Hive-функція `percentile()` дозволяє одночасно обчислити кілька процентилей. Для цього на неї потрібно подати масив (`array()`) з необхідними значеннями процентилей:

```
> flights_tbl %>%
+ summarise(perc = percentile(dep_delay, array(0.25, 0.5, 0.75)))
# Source: spark<?> [?? x 1]
  perc
  <list>
1 <dbl [3]>
>
```

Щоб автоматично отримати ці значення зі списку служить Hive-функція `explode()`:

```
> flights_tbl %>%
+ summarise(perc = percentile(dep_delay, array(0.25, 0.5, 0.75))) %>%
+ mutate(perc = explode(perc))
# Source: spark<?> [?? x 1]
   perc
  <dbl>
1    -5
2    -2
3    11
```

Нижче підрахунок пропущених значень виконаний для всіх стовпців таблиці `flights_tbl` за допомогою команди `summarise_each()` з пакету `dplyr` в поєднанні з анонімною функцією, яка задає логіку обчислень:

```
> flights_tbl %>%
+ summarise_each(list(~sum(as.integer(is.na(.))))) %>%
+ glimpse
Rows: ??
Columns: 19
Database: spark_connection
$ year      <dbl> 0
$ month     <dbl> 0
$ day       <dbl> 0
$ dep_time  <dbl> 8255
$ sched_dep_time <dbl> 0
$ dep_delay <dbl> 8255
$ arr_time  <dbl> 8713
$ sched_arr_time <dbl> 0
$ arr_delay <dbl> 9430
$ carrier   <dbl> 0
$ flight    <dbl> 0
$ tailnum   <dbl> 2512
$ origin    <dbl> 0
$ dest      <dbl> 0
$ air_time  <dbl> 9430
$ distance  <dbl> 0
$ hour      <dbl> 0
$ minute    <dbl> 0
$ time_hour <dbl> 0
```



Яка максимальна кількість пропущених значень? Скільки це становить від загального числа спостережень в таблиці?

Максимальна кількість пропущених значень = 9430.

Це становить приблизно 2.8% (9430/336776)

Якщо частка пропущених значень невелика, ми можемо видалити відповідні рядки з таблиці без особливого ризику вплинути на якість подальшого аналізу. Для цього можна скористатися базовою функцією `na.omit()`:

```
> flights_full <- flights_tbl %>% na.omit()
* Dropped 9430 rows with 'na.omit' (336776 => 327346)
> flights_full %>% sdf_dim()
[1] 327346    19
> _
```

Оскільки нас цікавлять рейси, затримка яких склала від 15 до 30 хв (включно), далі нам потрібно відфільтрувати дані відповідним чином.

Паралельно додамо новий стовпець `target` зі значеннями залежної змінної:

```
> flights <- flights_full %>%
+ filter(dep_delay >= 15, dep_delay <= 30) %>%
+ mutate(target = as.integer(arr_delay <= 0))
> flights %>% sdf_dim()
[1] 24515    20
>
```

Розрахуємо медіанне значення цієї відстані для обох класів залежною змінною:

```

> flights %>%
+ group_by(target) %>%
+ summarise(median_dist = percentile(distance, 0.5))
# Source: spark<??> [?? x 2]
  target median_dist
  <int>      <dbl>
1     1         1089
2     0          820
>

```

## **Висновки**

В ході виконанні лабораторної роботи було освоєння використання Spark кластера на локальній машині для виконання обчислень для великих даних в середовищі R.