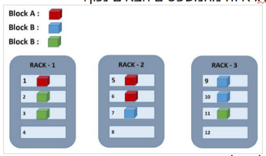
Finals Notes

יום שלישי 04 יולי 2023 07:32

2. איזה מהמשפטים הבאים **אינו** נכון לגבי חישוב בעזרת שיטת MapReduce:
   - A. הפלט של תהליך MapReduce נשמר בקובץ במערכת קבצים לוקלית (לא HDFS). לכן, אם יש צורך לשרשר מספר תהליכי MapReduce, אז צריך כל פעם להעביר את הנתונים ל HDFS והעברה זאת אמורה להיות יקרה בזמן.
   - B. השלב של Map של MapReduce אינו דורש העברות נתונים דרך הרשת.
   - C. השלב של Reduce של MapReduce אינו דורש העברות נתונים דרך הרשת.
   - D. השלב של Reduce אינו יכול לעבוד נכון ללא השלב של Sort & Shuffle.
   - E. השלב של Mapper חייב להכין מילון עבור השלב של Sort & Shuffle.

2. נניח שהאיור המופיע למטה (נלקח מהמצגת) מתאר שמירה של קובץ שלם בגודל 192 מגה ב- HDFS. איזה מהמשפטים הבאים נכון?



A. גודל הבלוק הוא 128 מגה.
   a. Default in hadoop 2.0, but not necessarily the configuration

B. הקובץ מורכב מ- תשע בלוקים שונים.-- 3, as stated in the top

C. הקוביות באותו צבע (כגון אדום) מתייחסים לאותם הנתונים.
   a. Correct, these are replications for data resiliency

D. הקובץ שמור על שלושה דיסקים קשיחים בלבד.(depends on config)

E. מחשבים עם מספרים 4, 8 ו- 12 אינם חלק מהאשכול.

F. הקונפיגורציה באיור זה מונעת לחלוטין אפשרות של איבוד נתונים.

The statements that not true about calculations using the MapReduce method are:

A. This is not entirely accurate. The output of a MapReduce process is typically saved to the Hadoop Distributed File System (HDFS)**,** not a local file system. However, there might be cases where you save the output to a local file system, but it's not the standard process. If multiple MapReduce jobs are chained, intermediate data can be stored in HDFS, but Hadoop tries to schedule subsequent jobs on the same nodes where data resides to minimize network transfer.

B. This statement is true. The Map phase of MapReduce does not generally require data transfers through the network, because it operates on local data blocks.

C. This statement is true. The actual network transfers are mainly done during the Shuffle and Sort phase, which comes between the Map and Reduce phases in the MapReduce model. The Shuffle phase is responsible for taking the outputs of the Map tasks (the intermediate key-value pairs), shuffling them so all key-value pairs with the same key go to the same Reduce task, and then transferring these pairs over the network. By the time we get to the Reduce phase, all of the necessary data has already been transferred to the correct location, and the Reduce tasks can just read the data from local disk or memory.
.

D. This statement is true. The Reduce phase cannot operate properly without the Shuffle and Sort phase in MapReduce. The Shuffle and Sort phase is necessary to sort and transfer the output of the Map tasks to the Reduce tasks.

E. This statement is true. The Map phase outputs key-value pairs.

1. באיזה מהמצבים הבאים לא נשתמש ב- Hadoop:

1. הנתונים שמורים בדיסקים קשיחים של עשר אלף מחשבים.
2. למחשבים העומדים לרשותנו זיכרון מרכזי קטן.

   Hadoop can be used. Hadoop stores data in a distributed file system that allows it to be processed in parallel, making it less dependent on the machine's central memory.

3. אפשר לבטא חישוב בצורת שאילתא בשפת SQL.
4. דרוש שימוש במחשב על.

   Supercomputers are highly specialized systems designed to handle complex computational tasks very quickly. If such resources are available, it might be more efficient to use these instead of a distributed system like Hadoop, depending on the task.

5. המחשבים העומדים לרשותנו אינם נמצאים באותו מקום גאוגרפי.
6. המחשבים העומדים לרשותנו משתמשים בסוגים שונים של חומרה (מעבד, לוח אם וכו').

   Hadoop can be used. Hadoop can run on different hardware configurations. As long as the machines meet Hadoop's minimum requirements, it should be able to function across various types of hardware.

אלו לא נכונים לגבי MapReduce:

1. MapReduce מונע באופן מוחלט את הצורך להעביר נתונים דרך הרשת, כי אנחנו מעבירים חישובים לנתונים ולא להפך.
   False. The mapper function s brought to the data itself in hdfs, andr un on the storage nodes, but the outputs of the mapper go over the network to the reducer function.
2. כאשר משתמשים ב- MapReduce, המתכנת צריך לכתוב שלוש תכניות – Mapper, Sorter ו- Reducer.

   The "Sort and Shuffle" phase that happens between the Map and Reduce stages is handled by the MapReduce framework itself, and the programmer typically does not need to explicitly write any code for this phase. This phase ensures that all data belonging to a single key is directed to the same reducer, which is essential for the reduce step to work correctly.

נכון או לא נכון: ד. HDFS אינה אחראית על חלוקת משבאים חישוביים באשכול Hadoop.

True, HDFS is the storage system, and the computation is handled by frameworks like MapReduce

LINUX

```
pwd                      # Print current directory path
ls                       # List directories
ls -a|--all              # List directories including hidden
ls -l                    # List directories in long form
ls -l -h|--human-readable # List directories in long form with human readable sizes
ls -t                    # List directories by modification time, newest first
stat foo.txt             # List size, created and modified timestamps for a file
stat foo                 # List size, created and modified timestamps for a directory
tree                     # List directory and file tree
tree -a                  # List directory and file tree including hidden
tree -d                  # List directory tree

cd foo                   # Go to foo sub-directory
cd                       # Go to home directory
cd ~                     # Go to home directory
cd -                     # Go to the previously chosen directory
pushd foo                # Go to foo sub-directory and add previous directory to stack
popd                     # Go back to directory in stack saved by `pushd`

cp foo.txt bar.txt                 # Copy file
mv foo.txt bar.txt                 # Move file
cp -R|--recursive foo bar          # Copy directory
```

```
echo "foo" > bar.txt      # Overwrite file with content
echo "foo" >> bar.txt     # Append to file with content

ls exists 1> stdout.txt   # Redirect the standard output to a file
ls noexist 2> stderror.txt # Redirect the standard error output to a file
ls > out.txt 2>&1         # Redirect standard output and error to a file
ls > /dev/null            # Discard standard output and error
ls -l |                   # Detailed list of files

read foo                  # Read from standard input and write to the variable foo

rmdir foo                 # Delete non-empty directory
rm -r|--recursive foo     # Delete directory including contents
rm -r|--recursive -f|--force foo  # Delete directory including contents, ignore nonexistent files and never prompt

Nano foo.txt   # open a new file in nano
touch foo.txt     # Create file or update existing files modified timestamp
Cat foo.txt    #open existing file

mv foo bar                        # Move directory
mv folder1/subfolder1 . (move subfolder to current directory
```

SPARK

Transformations:

- map(func): Returns a new RDD by applying a function to each element of the RDD.
- filter(func): Returns a new RDD by selecting only the elements of the original RDD on which func returns true.
- flatMap(func): Similar to map, but each input item can be mapped to multiple output items.
- union(dataset): Returns a new RDD that contains the union of the elements in the source RDD and the argument.
- distinct([numTasks])): Returns a new RDD that contains the distinct elements of the source RDD.
- groupByKey([numTasks]): When called on a dataset of (K, V) pairs, returns a dataset of (K, Iterable<V>) pairs.
  - rdd = sc.parallelize([('A', 1), ('B', 1), ('A', 1), ('A', 1), ('B', 1), ('B', 1)], 2) # Use groupByKey to group values by key
  - grouped = rdd.groupByKey() # Use mapValues to sum the values in each group
  - result = grouped.mapValues(sum) - > **Output: [('A', 3), ('B', 3)]**
- reduceByKey(func, [numTasks]): When called on a dataset of (K, V) pairs, returns a dataset of (K, V) pairs where the values for each key are aggregated using the given reduce function.
  - rdd = sc.parallelize([('A', 1), ('B', 1), ('A', 1), ('A', 1), ('B', 1), ('B', 1)], 2) ; result = rdd.reduceByKey(lambda a, b: a + b)
  - result.collect() - > Output = [('A', 3), ('B', 3)] . Within each group, reduceByKey applies the lambda function to reduce the data. **It takes two values at a time (represented by a and b)** and adds them together. It continues this process until all values in the group have been combined into a single value.
- glom(): transforms each partition into a tuple (immutabe list) of elements. It creates an RDD of tuples.
  - **rdd = sc.parallelize([('A', 10), ('B', 20), ('C', 30)], 2) ; rdd.glom().collect() => [[('A', 10)] , [('B', 20), ('C', 30)]] (length of 2, for the 2 partitions created.**

Actionss

- reduce(func): Aggregate the elements of the RDD by applying a function that takes two arguments and returns one. The function should be commutative and associative.
- collect(): Return all the elements of the RDD as an array to the driver program. Be careful with this operation as it can cause the driver to run out of memory if the RDD is too large.
- count(): Return the number of elements in the RDD.
- first(): Return the first element of the RDD.
- countByKey(): Only available on RDDs of type (K, V). Returns a hashmap of (K, Int) pairs with the count of each key.

Spark vs MapReduce

- **Data Processing:** MapReduce reads from and writes to the disk after each map and reduce operation. This can be slower due to the time it takes to read and write from disk. On the other hand, Spark performs in-memory data processing, it can store intermediate data in memory rather than disk, which makes it much faster for iterative and interactive computing tasks.
- **Fault Tolerance:** Both MapReduce and Spark are fault-tolerant, but they achieve it in different ways. MapReduce writes data to disk after each task, so it can recover data from disk if a task fails. Spark, on the other hand, keeps track of the lineage of each RDD (Resilient Distributed Dataset) so it can recompute lost data if a task fails.
- **Real-Time Processing**: MapReduce is designed for batch processing and does not support real-time data processing. Spark supports batch processing, interactive queries, streaming, and machine learning, which makes it suitable for real-time data processing.
- Lazy Loading: Lazy evaluation in Apache Spark means that data transformations are not immediately executed when they are defined, but are instead recorded for later execution. The actual computations are triggered only when an action is called, allowing Spark to optimize the overall data processing workflow.
    - So, in the following example:
      data = sc.textFile("non_existent_file.txt") # Load a non-existent text file
      words = data.flatMap(lambda line: line.split(" ")) # Split each line into words
      filtered_words = words.filter(lambda word: word.startswith("a")) # Filter words that start with "a"
    - If the .txt file doesn't exist, then no error will be thrown at transformation time. Only when a command like **count = filtered_words.count()** will the error get thrown.

REGEX

Anchors
^, Start of string, or start of line in multi-line pattern
\A, Start of string
$, End of string, or end of line in multi-line pattern
\Z, End of string
\b, Word boundary
\B, Not word boundary
\<, Start of word
\> End of word

Character Classes
\c, Control character
\s, White space
\S, Not white space
\d, Digit
\D, Not digit
\w, letter or digit

Quantifiers
*, 0 or more, {3}, Exactly 3
+, 1 or more, {3,}, 3 or more
?, 0 or 1, {3,5}, 3, 4 or 5

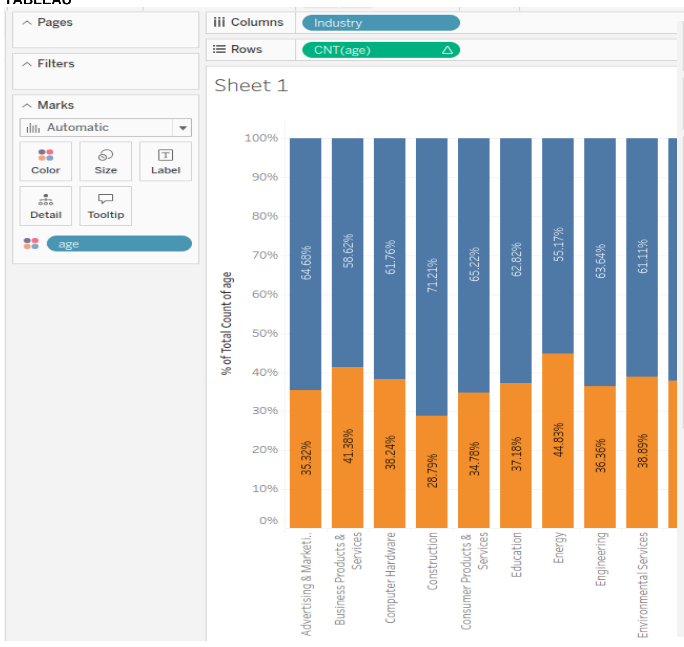
Groups and Ranges (ranges inclusive
.   Any character except new line (\n)
(a|b), a or b
(...), Group
(?:...), Passive (non-capturing) group
[abc], Range (a or b or c)
[^abc], Not (a or b or c)
[a-q], Lower case letter from a to q
[A-Q], Upper case letter from A to Q
[0-7], Digit from 0 to 7
\x, Group/subpattern number "x"

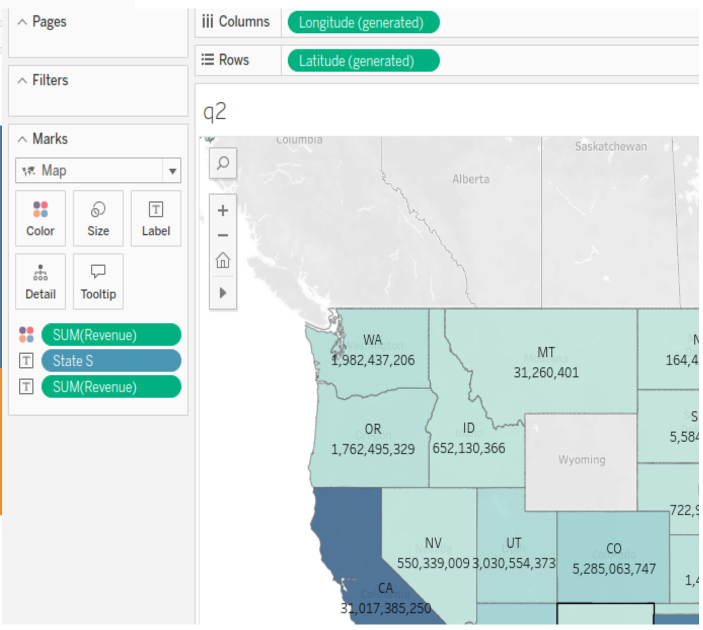
Sub-expressions
?=, Lookahead assertion
?!, Negative lookahead
?<=, Lookbehind assertion
?!= or ?<!, Negative lookbehind

"Wellbeing" ralph@gmail.com (+1)234-345-3434534
"7 Habits" john@gmailcom (+1)555-053-1234a67
"GTD" victor@.co.il 02-765432
"Some denis@yahoo.ca 09-234234Z3
"Jewish history" Josh.gmail.com "(+999)1-1223321"

1. All landline phones in the Israel (ie, without a country code). - > **0[0-9]-.***
2. Each appearance of the letter "J" must be found as the main title of a book when it appears in Gershiim. **((?<=\")J**
3. Find every line that does not end with a number.
4. All the correct email addresses must be found. (Your expression must be precise enough to catch the errors that appear in the file). Hint: a dot is matched, so write \. to match a regular dot.
5. Find the entire title that appears in commas and is longer than 7 characters (not including the commas).
6. Any email address with 4 or 5 characters before @ must be found.
7. Find every email address that doesn't have an @ in it.

TABLEAU


Pages
Filters
Marks
Automatic
Color
Size
Label
Detail
Tooltip
age
Columns
Industry
Rows
CNT(age)
Sheet 1
% of Total Count of age
100%
90%
80%
70%
60%
50%
40%
30%
20%
10%
0%
64.68%
35.32%
58.62%
41.38%
61.76%
38.24%
71.21%
28.79%
65.22%
34.78%
62.82%
37.18%
55.17%
44.83%
63.64%
36.36%
61.11%
38.89%
Advertising & Marketi..
Business Products & Services
Computer Hardware
Construction
Consumer Products & Services
Education
Energy
Engineering
Environmental Services


Pages
Filters
Marks
Map
Color
Size
Label
Detail
Tooltip
SUM(Revenue)
State S
SUM(Revenue)
Columns
Longitude (generated)
Rows
Latitude (generated)
q2
Columbia
Saskatchewan
Alberta
WA
1,982,437,206
MT
31,260,401
164,4
OR
1,762,495,329
ID
652,130,366
Wyoming
5,58
NV
550,339,009
UT
3,030,554,373
CO
5,285,063,747
722,9
CA
31,017,385,250