



WIZELINE ACADEMY

Grow your career:
Free courses in Artificial Intelligence,
Software Development, User Experience and
More

WiFi: Wizeline Academy
Password: academyGDL



@WizelineAcademy



/WizelineAcademy



academy.wizeline.com



Get notified about courses:
tinyurl.com/WL-academy



Week 3 -
Session 1/2

Spark Data Partitioning

Agenda

- **Shuffling**

- Narrow and Wide

- **Partitioning**

- Enforcing

- Default

- **I/O and optimization strategies**

- **Operations**



Shuffling



Shuffling

What is and why it's so important?

- The process of moving the data from partition to partition to aggregate, join, match up, or spread out in some other way, is known as **shuffling**.
- It may occur even if we never call the Shuffle method directly. This happens behind the curtain for some functions.
- This typically involves copying data across executors and machines, making the shuffle a complex and costly operation because of *Latency*!



Wide vs Narrow Dependencies

- **Narrow Dependencies:** Each partition of the parent is used by maximum one partition of the child.
- **Fast!** No shuffle necessary.

Optimizations such as pipelining possible.

- **Wide Dependencies:** Each partition of the parent may be used by multiple child partitions.
- **Slow!** Shuffle necessary for all or some data over the network.

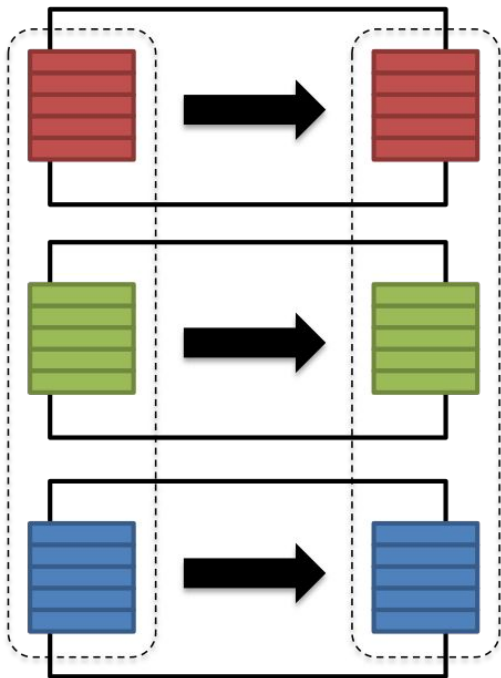


Shuffling

Wide vs Narrow

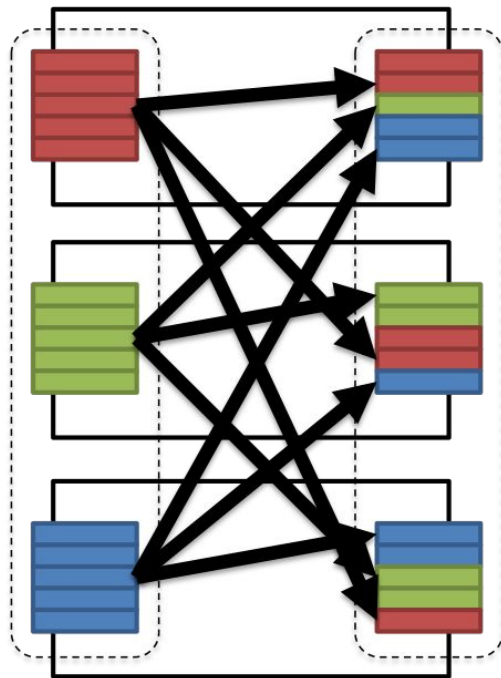
Narrow transformation

- Input and output stays in same partition
- No data movement is needed



Wide transformation

- Input from other partitions are required
- Data shuffling is needed before processing





Partitioning



What is Partitioning?

- Spark manages data using partitions that help parallelize distributed data processing with minimal network traffic for sending data between executors.
- Depending on how you look at Spark (programmer, DevOps, or Admin), an RDD is about the content (developer's and data scientist's perspective) or how it gets spread out over a cluster (performance). Simply put. It means how many partitions an RDD represents.



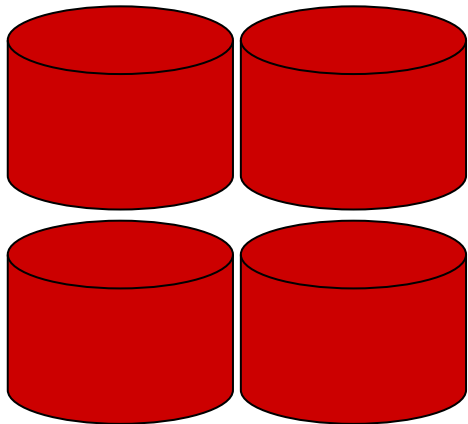
What is Partitioning? (Contd.)

- In general, more numerous partitions allow work to be distributed among more workers, but fewer partitions allow work to be done in larger chunks (and often quicker).
- By default, Spark uses 200 partitions.
- But keep in mind that partitioning will not be helpful in all applications!



Partitioning

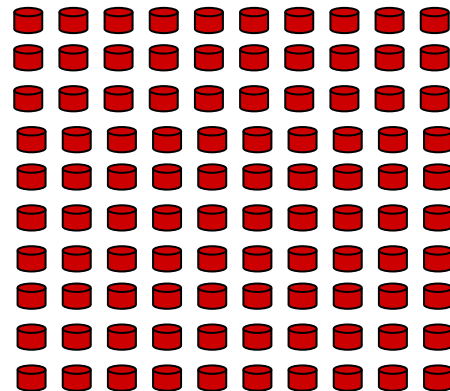
May involve shuffling; can be disabled



Low Partition



Right Partition?



High Partition



Default Partitioning

By default, Spark uses two different approach for partitioning:

- **HashPartitioner**
- **RangePartitioner**



Default Partitioning

HashPartitioner

- **HashPartitioner** works on Java's `Object.hashCode()`.

The concept of `hashCode()` implies that objects that are equal, should have the same `hashCode()`. So, based on this `hashCode()` concept, `HashPartitioner` divides the keys that have the same `hashCode()`.



Default Partitioning

RangePartitioner

- **RangePartitioner:** If there are sortable records, then range partition divides the records almost in equal ranges. The ranges are determined by sampling the content of the RDD passed.
- First, **RangePartitioner** sorts the records based on the key, and then it divides the records into a number of partitions based on a given value.



API Methods

How to work with partitions?

- **spark.sql.shuffle.partitions**
 - Configures the number of partitions that are used when shuffling data for joins or aggregations.
- **spark.default.parallelism**
 - Determines the default number of partitions in RDDs returned by transformations like `join`, `reduceByKey`, and `parallelize`, when not set by the user.

NOTE: This will be ignored by Spark SQL, and is only relevant when working on plain RDDs.



Custom Partitioning

How to work with it?

coalesce(num_partitions, shuffle=False)

and

repartition(num_partitions)

transformations are used for changing the number of partitions.

NOTE: repartition() calls coalesce() with explicit shuffling.



HINT

- The function **glom()** allows you to treat a partition as an array rather as single row at time.
- This allows you speed up some operations with some increased memory usage.

NOTE: This function will be used in the example notebook!



Custom Partitioning

The rules for using are as follows:

- if you are increasing the number of partitions use repartition() (performing full shuffle).
- if you are decreasing the number of partitions use coalesce(). (It minimizes shuffles by default).




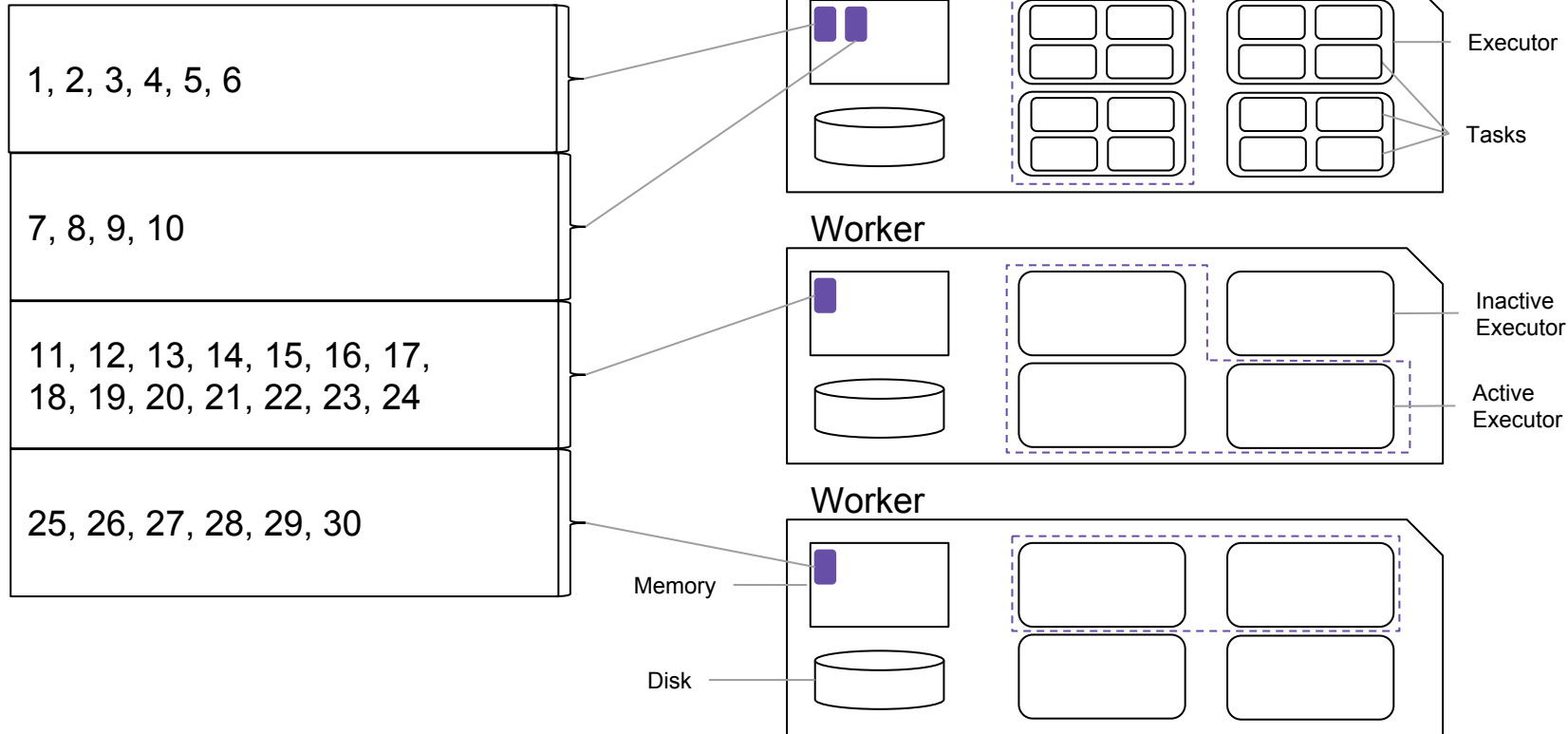
Custom Partitioning

- Spark can only run **1** concurrent task for every partition of an RDD, up to the number of cores in your cluster.
- So, if you have a cluster with 50 cores, you want your RDDs to at least have 50 partitions (and probably 2-3x times that).



Partitioning Example

DataSet broken into 4 partitions 





Recap

What we learn so far:

- What partitioning is
- Wide vs Narrow partitioning
- Spark Default Data partitioning
- Custom partitioning



It's time for...

Questions & Answers



Exercise Time



Activity



Apache Zeppelin

Exercise 1:

See the difference in data distribution, when defining different number of partitions.

Exercise 2:

Partition the data using a different column.

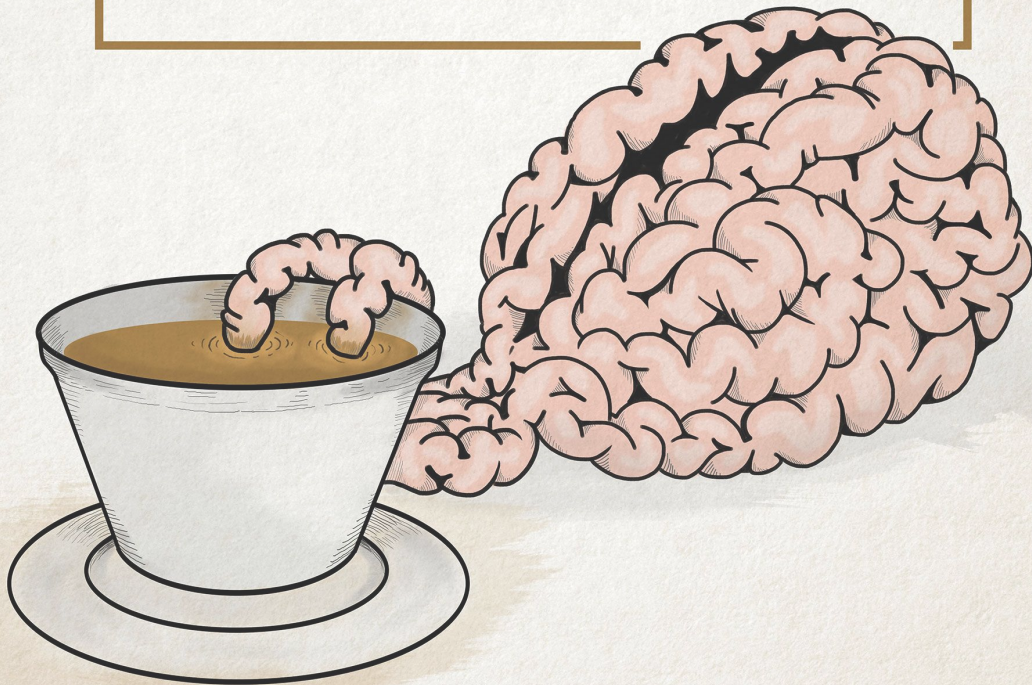
Question: How can you determine the best approach?



Wait!

It's time for...

COFFEEBREAK





I/O and Optimization Strategies



Input and Output

- The way we read or write data could impact performance. Remember that we are talking about distributed computation. So, it is not the same as working with a single file, and things can get complicated as more and more files are added.
- What will happen if we write one big DataSet in just one simple file?



Input and Output

- Writing in one single file, means putting everything in one worker and then write. This approach could drive us into various OUT OF MEMORY problems.
- Remember how some of you ran out of memory during the Big Matrix assignment?



Input and Output

- The same rules apply while reading files.
- Reading a dataset, which is already partitioned may help to improve performance.
- Remember that being in a distributed environment means that multiple workers can do the task at the same time.



It's time for...

Questions & Answers



Exercise Time



Activity



Apache Zeppelin

Exercise 3:

Read and Write a DataSet in a single file and then in multiple files.

Question: How do you determine the best approach?

Exercise 4:

Get the size of each partition by using the DataSet of the flights.



Operations



Transformations with (Usually) Narrow Dependencies

- `map`
- `mapValues`
- `flatMap`
- `filter`
- `mapPartitions`
- `mapPartitionsWithIndex`



Transformations with (Usually) Wide Dependencies

- `cogroup`
- `groupWith`
- `join`
- `groupByKey`
- `combineByKey`
- `distinct`
- `intersection`
- `repartition`
- `coalesce`

NOTE: Might cause a shuffle.



Operations HINT

- One of the best approaches is to use the most simple or basic operation, and then, if needed, to try to optimize with other operation to reduce the cost (Time, Memory, Network, CPU, etc...)



Recap

What we have learned so far:

- Operations
- Input and Output Optimizations
 - Single File
 - Multiple File



It's time for...

Questions & Answers

FURTHER READING



Partitions and Partitioning

<https://jaceklaskowski.gitbooks.io/mastering-apache-spark/spark-rdd-partitions.html>

Databrick Spark Best Practices

https://databricks.gitbooks.io/databricks-spark-knowledge-base/best_practices/REALDME.html

For the Knowledge Hungry...

Spark Memory Management

<https://0x0fff.com/spark-memory-management/>



Assignment

To Work on Your Own at Home

A photograph of a silver laptop on a light-colored wooden desk. To the right of the laptop is a white cup of coffee. The image is partially obscured by a dark overlay and a red triangle in the top left corner.

**Partitioning and
Performance**



Assignment

Partitioning Exploring

Read the stock-order dataset from the Alimazon folder

`gs://de-training-input/alimazon/200000/stock-orders`

You need to partition it by different strategies and find the number of element per partition (similar to what we did in the exercices) and the results should be written to your output bucket

`gs://de-training-output-<username>/assignment-5`

For each partitioning exercise the output should be a **csv** file with a **single column containing the size of the partitions**. The rows should be **sorted in descending order**.



Assignment

Partitioning Exploring

- 1) Write the size of each partition obtained by default after reading the dataset in the folder:
gs://de-training-output-<username>/assignment-5/default-partitions
- 2) Partition the dataset by client_id and write the size of the partitions in the folder:
gs://de-training-output-<username>/assignment-5/client_id-partitions
- 3) Partition the dataset by 15 partitions and write the size of the partitions in the folder:
gs://de-training-output-<username>/assignment-5/fifteen-partitions
- 4) Partition the dataset by 15 partition and client_id and write the size of the partitions in the folder: **gs://de-training-output-<username>/assignment-5/fifteen-client_id-partitions**
- 5) Partition the dataset by 15 partition and product_id and month and write the size of the partitions in the folder:
gs://de-training-output-<username>/assignment-5/fifteen-product_id-month-partitions



Assignment

Output Partitioning

Read the stock-order dataset from the Alimazon folder:

`gs://de-training-input/alimazon/200000/stock-orders`

You will repartition it by several strategies so the number of files written correspond to the number of partitions (Effectively controlling the number of files to be written).

Your output should be written to the folder:

`gs://de-training-output-<username>/assignment-5`



Assignment

Partitioning Exploring

- 1) Repartition the output to generate 10 files and write them to the folder:
`gs://de-training-output-<username>/assignment-5/10-files`
- 2) Repartition the output to generate 150 files and write them to the folder:
`gs://de-training-output-<username>/assignment-5/150-files`
- 3) Repartition the output to generate 800 files and write them to the folder:
`gs://de-training-output-<username>/assignment-5/800-files`

Observe the time it takes for each exercise to run and think on how are we impacting the performance by writing different number of files.

Write a **conclusions.txt** file with your conclusions and the time that took to run each job of the assignment in the folder:

`gs://de-training-output-<username>/assignment-5/`



Where Can I Get this Presentation?

Slack Channel PDF



Your feedback is very valuable to us!

<https://goo.gl/forms/OGL4SGRkGH9Xo0RH3>



THANK YOU

