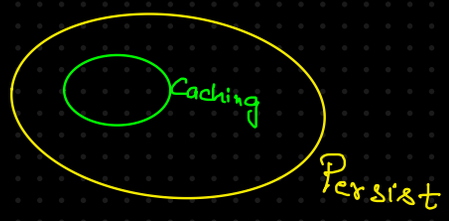
Persist and caching in spark



Apache spark provides mechanism like persist() and cache() to optimize job execution by storing immediate data in memory / disk, allowing faster access for repeated computation

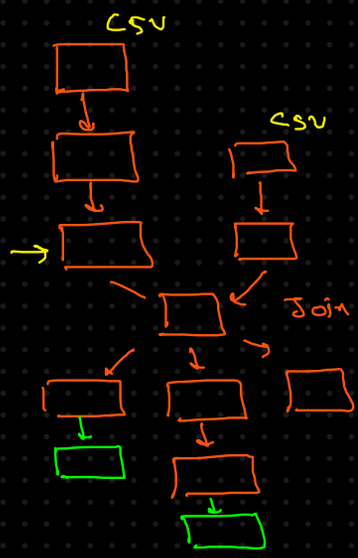
$2^9 = 512$

$2^{10} = 1024$ ⟶ your mind did caching here

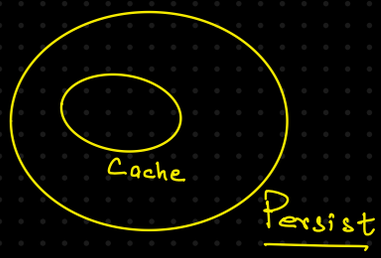
Why caching is needed when Spark does calculation in-memory?

Spark → in-memory processing framework

1. Repeated use of data -
2. Reducing disk reads
3. Lazy evaluation



Difference between cache() and persist()



Ser vs Non-serialized

Feature	`cache()`	`persist(storageLevel)`
Default Storage Level	MEMORY_AND_DISK (PySpark), MEMORY_AND_DISK_SER (Scala)	User-defined storage level
Control Over Storage	No	Yes
Ease of Use	Simpler, quick to implement	More flexible
Replication	No replication	Can replicate across nodes
Serialization	Not customizable	Customizable (serialized/deserialized)

Use cache() when dyault memory-disk behaviour is sufficient

persist() when we need to control how data is stored

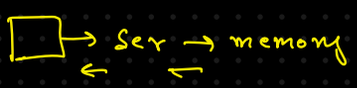
Persistence Level	Description	Storage Location	Fault Tolerance
`MEMORY_ONLY`	Stores the data in memory (RAM). If the data doesn't fit in memory, some partitions will be lost.	In-memory (RAM)	No replication
`MEMORY_AND_DISK`	Stores data in memory. If the data doesn't fit in memory, it will spill to disk.	In-memory (RAM), spills to disk	No replication
`DISK_ONLY`	Stores data only on disk. No data is stored in memory.	Disk (e.g., HDFS, local disk)	No replication
`MEMORY_ONLY_SER`	Stores the data in memory as serialized data. This can be more compact than the default format.	In-memory (RAM), serialized	No replication
`MEMORY_AND_DISK_SER`	Stores data in memory as serialized data. If the data doesn't fit in memory, it will spill to disk.	In-memory (RAM), serialized, and spills to disk	No replication
`OFF_HEAP`	Stores data off-heap (outside JVM heap). This requires enabling off-heap memory in Spark settings.	Off-heap memory (e.g., Tachyon or similar)	No replication
`MEMORY_ONLY_2`	Similar to `MEMORY_ONLY`, but with **replication factor 2**. Each partition is stored on two nodes for fault tolerance.	In-memory (RAM)	Replication Factor 2
`MEMORY_AND_DISK_2`	Similar to `MEMORY_AND_DISK`, but with **replication factor 2**.	In-memory (RAM) and spills to disk	Replication Factor 2
`MEMORY_ONLY_3`	Similar to `MEMORY_ONLY`, but with **replication factor 3**. Each partition is stored on three nodes for fault tolerance.	In-memory (RAM)	Replication Factor 3
`MEMORY_AND_DISK_3`	Similar to `MEMORY_AND_DISK`, but with **replication factor 3**.	In-memory (RAM) and spills to disk	Replication Factor 3

⇒ 10 partitions → 5 par → memory
5 par → disk

Loc
Serialization
replication
↓
persist()

Some Common Questions about caching

Q: How and where does caching happens?

1. Memory: 

if insufficient memory, then spark either drops the least-used partitions or spill them to disk

Spark also does some level of internal caching & we can do that explicitly as well.

.cache()

2. Disk : Data is written to local disk (not HDFS)

Q: Why local disk and not HDFS?

performance and architecture of spark

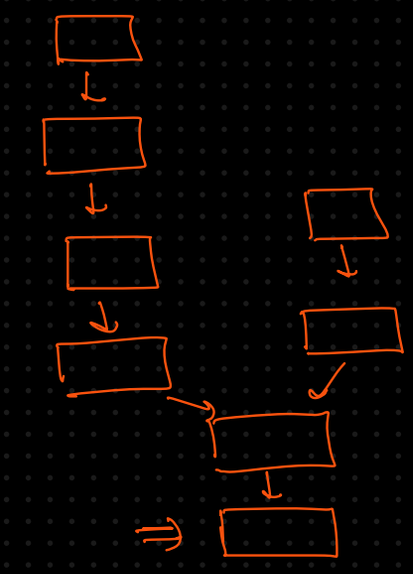
1. Faster access to data
2. Temporary Data storage
3. No need of fault tolerance in Caching → caching data is not ephemeral

Q: When to use Caching?

1. data will be used multible times
2. data is not too large to overwhelm
3. Avoid caching if memory pressure is a concern, degrade cluster performance.

Best practice

1. Cache medium sized dataset that are used repeatedly



Q: where can you see cached data?

- Caching is temporary.
- You can see the cached data on the UI of spark UI storage tab

* only running Jobs

Q: Can we also uncache the data

Yes

Q: Can caching leads to performance issue?

Yes, caching large datasets increase memory pressure.

Caching in RDD - Practical

→ Spark UI

→ always look for execution time

Understanding Spark UI is very imp.

We have a notebook where we ran for small file and a seperate one for larg file.

Completed Jobs (5)

Page: 1 — 1 Pages. Jump to 1 . Show 100 items in a page. Go

final customer-reduced

Job Id	Description	Submitted	Duration	Stages: Succeeded/Total	Tasks (for all stages): Succeeded/Total
4	count at /tmp/ipykernel_31339/2694472165.py:1 count at /tmp/ipykernel_31339/2694472165.py:1	2025/02/06 08:39:27	0.5 s	1/1 (1 skipped)	5/5 (5 skipped)
3	count at /tmp/ipykernel_31339/222424415.py:3 count at /tmp/ipykernel_31339/222424415.py:3	2025/02/06 08:38:58	2 s	1/1 (1 skipped)	5/5 (5 skipped)
2	count at /tmp/ipykernel_31339/3489286519.py:3 count at /tmp/ipykernel_31339/3489286519.py:3	2025/02/06 08:38:00	0.8 s	1/1 (1 skipped)	5/5 (5 skipped)
1	count at /tmp/ipykernel_31339/3631967605.py:3 count at /tmp/ipykernel_31339/3631967605.py:3	2025/02/06 08:35:24	0.8 s	1/1 (1 skipped)	5/5 (5 skipped)
0	count at /tmp/ipykernel_31339/3844578006.py:1 count at /tmp/ipykernel_31339/3844578006.py:1	2025/02/06 08:34:41	13 s	2/2	10/10

hitting cached final data

extra time to cache

for 540 mb file

Spark inbuilt caching

rdd → only memory is used

Spark Dataframe Caching

Big File

540mb file

1. header
2. show(5)] ⇒ faster] ⇒ just reading the Top data
3. show(5)
4. count()] ⇒ slower] ⇒ because all 5 partition
5. cache() → show() → slow as had to do caching] ⇒
6. show() → hit cache → faster] ⇒ as cache hit

Spark only cached the top partition and not the whole data

Storage

1/5 ⇒ 20%

▾ RDDs

ID	RDD Name	Storage Level	Cached Partitions	Fraction Cached	Size in Memory	Size on Disk
31	FileScan csv [customer_id#17,name#18,city#19,state#20,country#21,registration_date#22,is_active#23] Batched: false, DataFilters: [], Format: CSV, Location: InMemoryFileIndex(1 paths)[hdfs://my-cluster-m/tmp/customers_500mb.csv], PartitionFilters: [], PushedFilters: [], ReadSchema: struct<customer_id:string,name:string,city:string,state:string,country:string,registration_date:s ...	Disk Memory Deserialized 1x Replicated	1	20%	76.9 MiB	0.0 B

7. Tail() → Should it hit the cache? ⇒ Slower because cache miss or only a single partition cached.

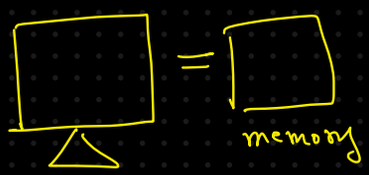
Caching on Spark Table

In spark tables, caching is eager unlike dataframes or rdd.

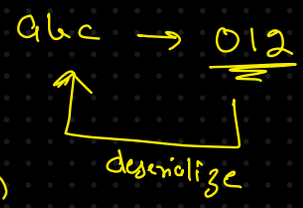
Caching Type	Description	When to Use
Eager Caching	Data is cached immediately after the command.	When data is small to medium in size and frequently accessed.
Lazy Caching	Data is cached only when accessed for the first time.	For large datasets with selective access.

To make the cache lazy in spark SQL use

Cache lazy table <table_name>



deserialized way in memory



disk data is serialized (dep. on platform)