Caching

Objective

Leverage fast memory access of intermediate computations

Understand cache mechanics

Compare different caching strategies

Make the correct tradeoffs for caching

Why

Spark applications have jobs

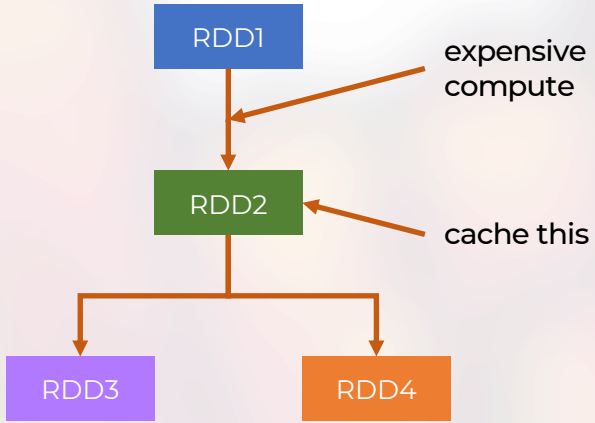
- each action triggers a job
- a job executes the query plan
- all dependencies in the query plan are evaluated

Save expensive computations

- the same RDD can be evaluated multiple times
- if RDD is expensive – job is expensive multiple times

Cache expensive RDDs

- the RDD lineage is kept
- the RDD data is kept in memory
- subsequent dependencies will fetch the cached data



Caching Mechanics

Can cache into

- memory in heap
- memory off heap (with Tungsten)
- disk
- memory + disk

Caching is done by executors on worker nodes

Beware of JVM limits

- min JVM memory 4-8GB
- max JVM memory 40GB
- the more JVM memory, the more time needed for GC
- large JVM heap may lead to decrease in perf

Caching Mechanics

Persistence levels

- memory-only, deserialized
- memory and disk
- memory-only, serialized
- memory and disk, serialized
- disk only
- memory-only 2 (replicated)
- memory and disk 2 (replicated)
- off-heap

```
mySuperRDD.cache()
```

can also use persist()

```
mySuperRDD.unpersist()
```

can't use uncache()

Caching Recap

Memory-only storage

- very CPU efficient
- can increase the risk of memory failures

```
mySuperRDD.cache(StorageLevel.MEMORY_ONLY)
```

Disk storage

- memory efficient
- slow to access

```
mySuperRDD.cache(StorageLevel.DISK_ONLY)
```

Serialization

- more CPU intensize
- 3x – 5x memory saving

```
mySuperRDD.cache(StorageLevel.MEMORY_ONLY_SER)
```

Replication

- 2x memory/disk usage
- fault tolerance

```
mySuperRDD.cache(StorageLevel.MEMORY_ONLY_2)
```

Off-heap

- free executor memory
- needs to be configured

```
mySuperRDD.cache(StorageLevel.OFF_HEAP)
```

```
spark.memory.offHeap.enabled = true
spark.memory.offHeap.size = 10485760
```

Caching Tradeoffs

Raw objects

- consume 3x-5x more memory (either RAM or disk)
- take 20x less time to process in RAM
- take more time to read from disk

Serialized objects

- max memory efficiency
- CPU intensive
- take less time to read from disk

Fault tolerance

- failed nodes will lose cached partitions
- cached partitions will be recomputed by other nodes (unless replicated)

Caching Recommendations

Only cache what's being reused a lot

- don't cache too much or you risk OOMing the executors
- the LRU data will be evicted

If data fits in memory, use MEMORY_ONLY (default)

- most CPU efficient

If data is larger, use MEMORY_ONLY_SER

- more CPU intensive, but still faster than anything else

Use disk caching only for really expensive computations

- simple filters take just as much (or even less) to recompute than reread from disk

Spark rocks