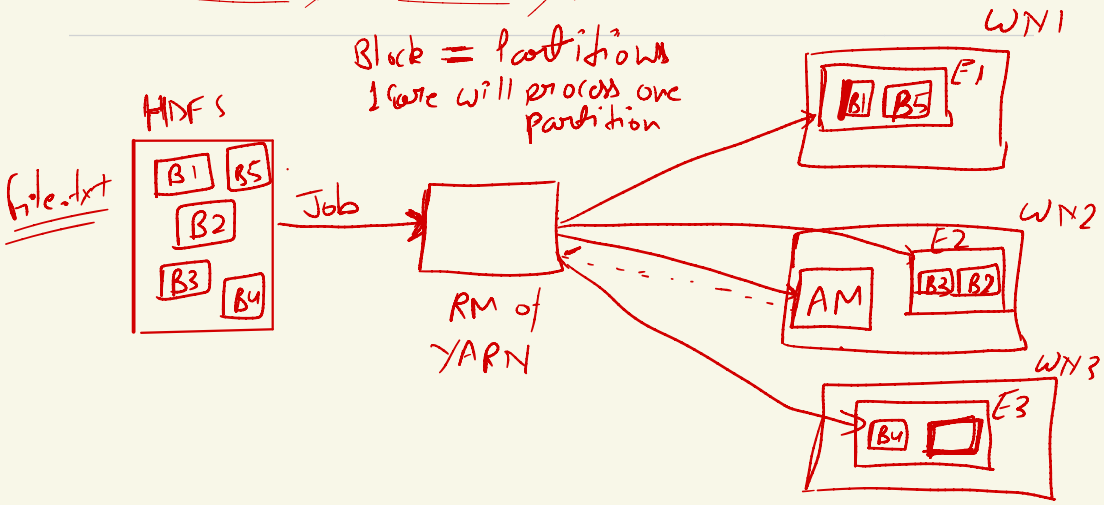
Spark memory management

1 partition process = 1 task

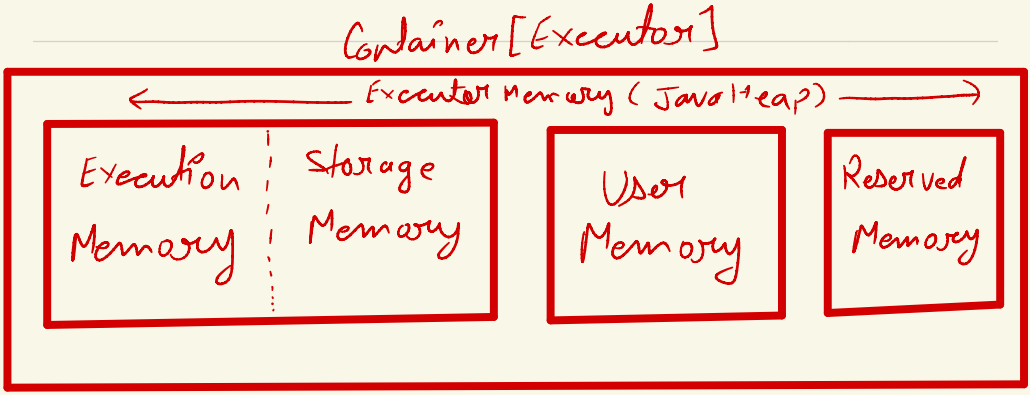


Configuration for Spark Job

1. number of spark Executors
2. memory for each executor
3. CPU cores for each executor

Concurrent = parallel = multiple

Spark Memory management



Parameter

--executor-memory 4GB → (RAM) (Java Heap memory)

We have 1 node with 32 GB RAM (4 memory)

↳ total 8 Executors = 32/4 = 8

Details about each component

① Reserved Memory → Storage for running executor. It will store the data of internal processing which is required to execute container. After spark 1.6+ versions, it is fixed

300 MB → fixed

② User memory →

Stores the Data structures, metadata and user defined data structures.

③ Storage Memory

It stores cache data, data of broadcast variables / RDDs etc.

④ Execution Memory

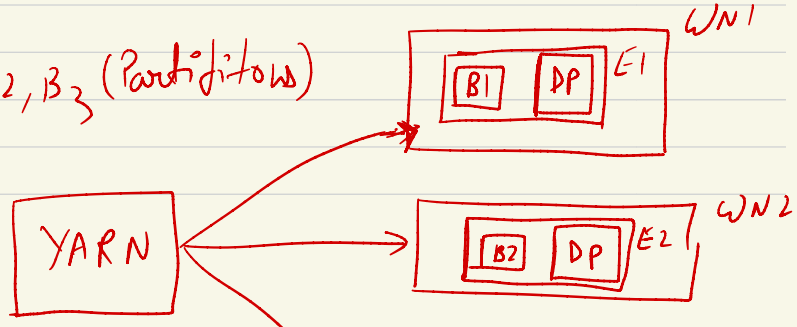
Storage for the data required for task executions, shuffle, join, aggregation

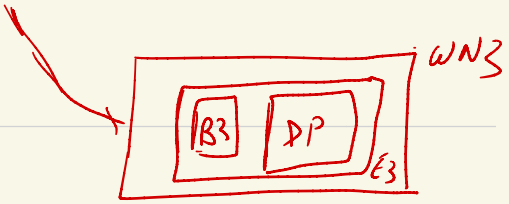
*) Common Error: Executor is Out of Memory.

Broadcast Variables

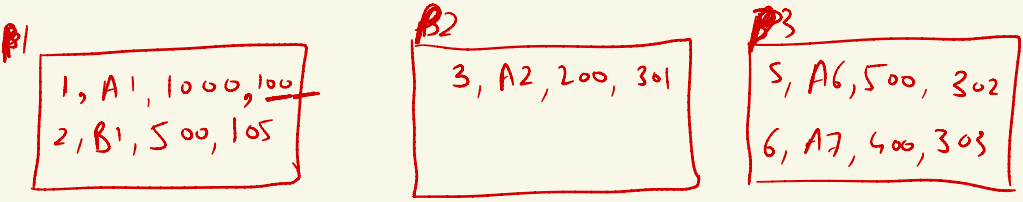
Rdd1 = employee_data
Rdd2 = depart_data
→ Join Operation

→ B1, B2, B3 (Partitions)

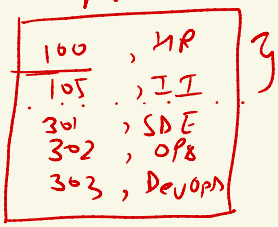




employee Rdd.



department (DP)



rdd1 = rdd2

rdd1.join(broadcast(Rdd2) . . .)

cache(rdd1)

Example explanation for memory distribution

Let say we have given 4GB memory to each executor.

(Java Heap) executor-memory = 4GB

Two parameters which controls the entire memory

Distribution in Spark:

default Values

spark.memory.fraction = 75% (0.75)

spark.memory.storageFraction = 50% (0.50)

① Reserved Memory :

for 4GB or any other executor Memory, value, it will be 300MB

it is a fixed value = 300MB

1GB = 1024B, 4GB = 4096 MB

② User memory -

formula = (Java Heap - Reserved Memory)
*
(1 - spark.memory.fraction)

Calculation = (4096MB - 300MB) * (1-0.75)

user memory = 949 MB

③ storage memory:

formula = (Java Heap - Reserved Memory)
*
spark.memory.fraction
*
spark.memory.storageFraction

Calculation = (4096MB - 300MB) * 0.75 * 0.5

≅ 1423.5MB or 1.4 GB

⑤ Execution Memory.

Formula = (Java Heap − Reserved Memory)
*
spark.memory.fraction
*
(1 − spark.memory.storageFraction)

Calculation = (4096MB − 300MB) * 0.75 * (1 − 0.5)

= (4096MB − 300 MB) * 0.75 * 0.5

≅ 1423.5MB

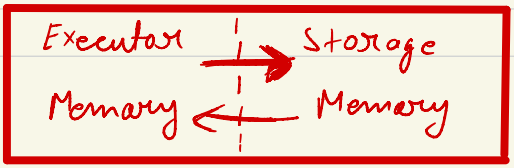
After Calculation if we verify

Reserved Memory = 300MB = 7.32%

User Memory = 949 MB = 23.16%

Spark Memory [Execution / Storage] = 2847MB = 69.5%

Rules for Execution & Storage Memory



*) Storage memory can borrow space from execution memory only if blocks (Memory Blocks) are not used in Execution memory

*) Execution memory can also borrow space from storage memory if blocks are not used in storage memory.

*) If blocks from execution memory is used by storage memory and execution needs more memory, it can forcefully evict the excess blocks occupied by storage memory.

*) If blocks from storage memory is used by execution memory and storage memory needs more memory, it can not forcefull evict the excess blocks occupied by execution memory, It will end up having less memory area. It will wait until spark releases the excess blocks stored by execution memory and then occupies them.

Resource allocation for Spark Application

Values for below mentioned parameters.

→ num-executors
→ executor-memory
→ executor-cores

Case1: Hardware → 6 Nodes, each node have 16 cores & 64GB RAM

x) How many CPU cores for each executor?

Recommended is **5 CPU cores** for each executor.

1 GB & 1 core of each node will be required for operating system.

Left resources for each node after above statement

1 Node Now → **15 cores & 63 GB memory**

x) How many executors?

1 executor will have 5 cores

15 cores = 15/5 executors

1 Node = 3 executors

total nodes = 6

total executors on cluster

= 3 × 6

= 18 executors

1 executor will be created by YARN as Application master.

So final executors which we should mention in

num-executors = 18 - 1 = 17

*) Memory for each executor?

1 Node = 3 executor

1 Node = 63 GB RAM

1 executor = 63/3 = 21 GB

---

spark.memory.overhead = 10% of executor memory

executor-memory = 19 GB

overhead = + 1.9 GB

20.9 GB = 20.9 × 3

= 62.7 GB

Actual formula for overhead memory

= max(384 MB, 0.07 * executor.memory)

As per calculation

1 executor = 21 GB RAM

overhead memory for each executor

↳ max(384 MB, 0.07 * 21 GB)

= max (384 MB, 1.47 GB)

= 1.47 GB

We need to substract overhead memory.

final executor memory = 21 GB - 1.47 GB

Memory for each ≅ 19 GB executor

final results for 6 Node with 16 cores each & 64 GB RAM

5 CPU cores Per executor

17 Executors

19 GB RAM for each Executor

Q.) Case 2 : 4 Nodes, each have 16 cores & 32 GB RAM ?