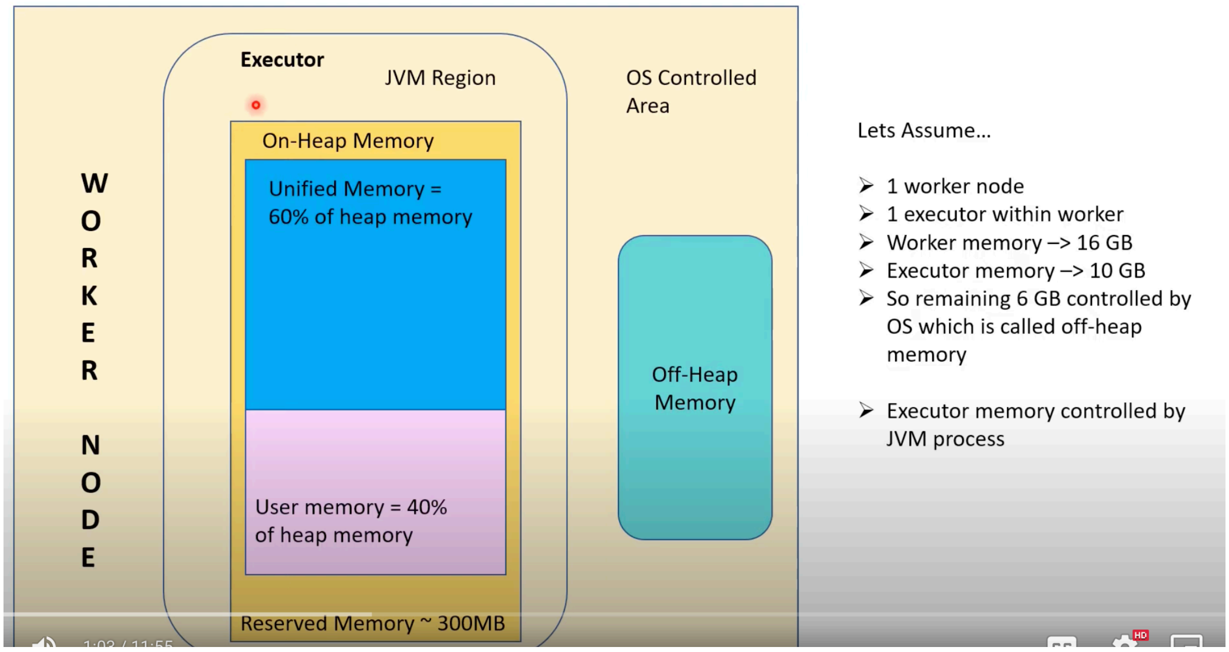
04. On-Heap vs Off-Heap| Databricks | Spark | Interview Question | Performance Tuning

Wednesday, 6 November 2024 10:39 AM

[04. On-Heap vs Off-Heap| Databricks | Spark | Interview Question | Performance Tuning]()

On-Heal vs Off-Heap memory management



Lets Assume...

- 1 worker node
- 1 executor within worker
- Worker memory –> 16 GB
- Executor memory –> 10 GB
- So remaining 6 GB controlled by OS which is called off-heap memory
- Executor memory controlled by JVM process

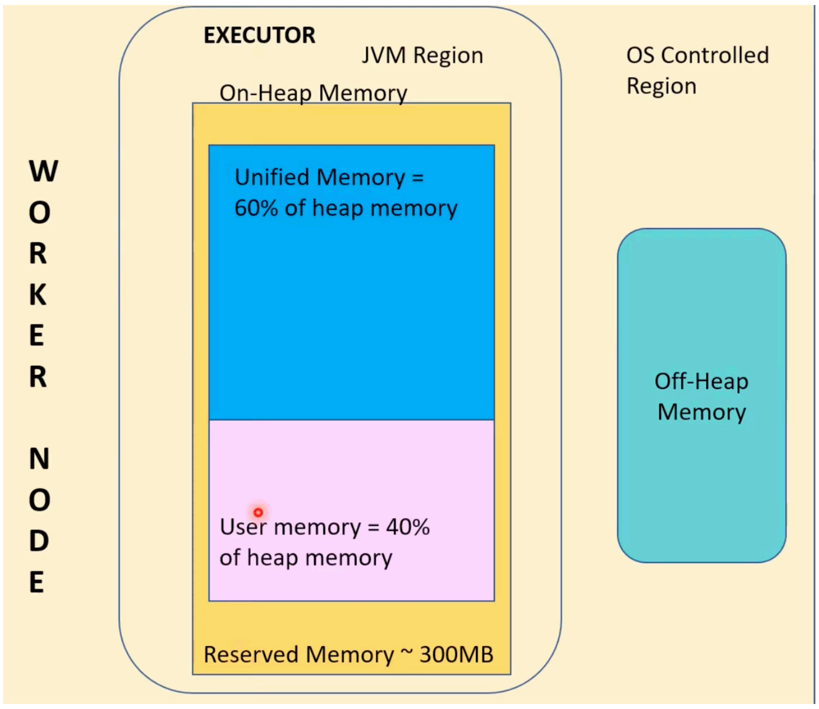
In this above example lets assume we have one worker that has 16gb memory

In this above example lets assume we have one worker that has 16gb memory.
Executor has 10gb of this 16gb memory
Remaining 6b will be directly controlled by OS - called off heap memory.
10gb will be controlled by JVM(Java virtual machine, a java process) - this is called on-heap memory.

In a spark application we can use any of on-heap or off-heap memory.
To design efficient spark applications we need to use both on-heap and off-heap memory.

What is On-heap memory - memory available in executor

What is Off-heap memory - memory directly controlled by OS



1.Reserved Memory: Reserved by Spark for internal purposes

2.User Memory: For storing the data-structures created and managed by the user's code

3.Execution Memory: JVM heap space used by data-structures during shuffle operations (join and aggregations)

4.Storage Memory: JVM heap space reserved for cached data

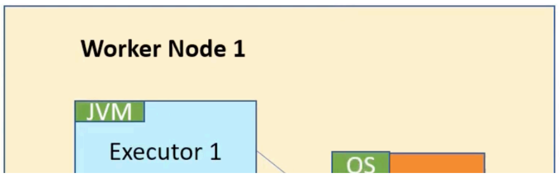
UMM = Execution memory (50% UMM) - Storage memory (50% of UMM

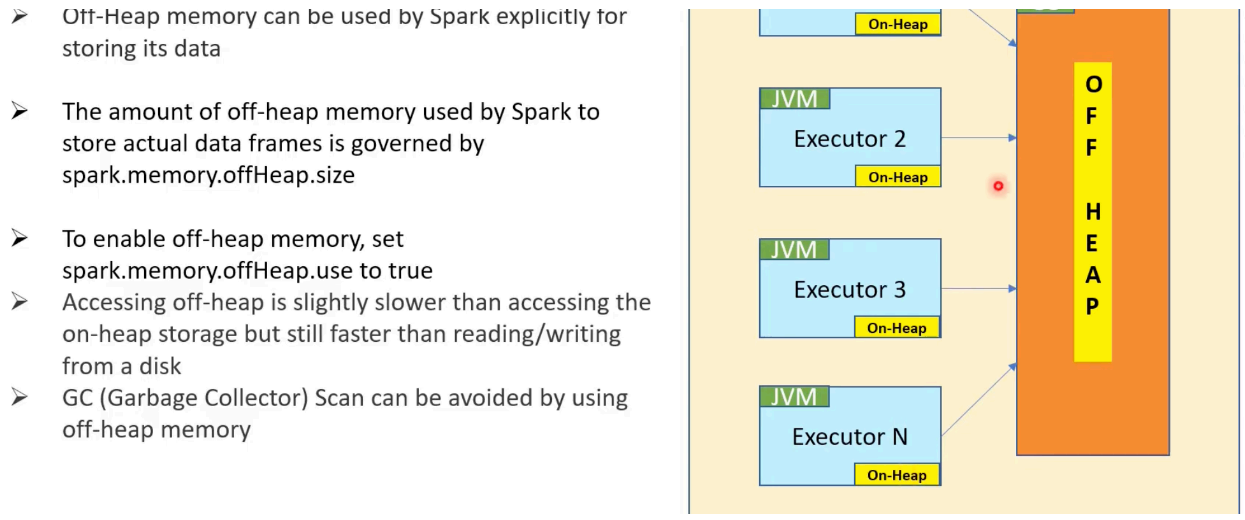
Reserved memory is used internally by spark and is used in case spark crashes, it's a small percentage of executor memory.

User memory - used for code execution from user, to store metadata etc
Unified memory is combination of execution memory and Storage memory

This percentage of unified memory can be configured by changing spark configuration settings.

- Each executor within worker node has access to Off-heap memory





- Off-Heap memory can be used by Spark explicitly for storing its data
- The amount of off-heap memory used by Spark to store actual data frames is governed by spark.memory.offHeap.size
- To enable off-heap memory, set spark.memory.offHeap.use to true
- Accessing off-heap is slightly slower than accessing the on-heap storage but still faster than reading/writing from a disk
- GC (Garbage Collector) Scan can be avoided by using off-heap memory

Data objects stored in off-heap memory are common to all executors.
In case for example one executor's on-heap memory is 1gb and it needs to process more than 1gb of data, it can take some memory from off-heap and store objects.

Each executor in worker node has access to off-heap memory.

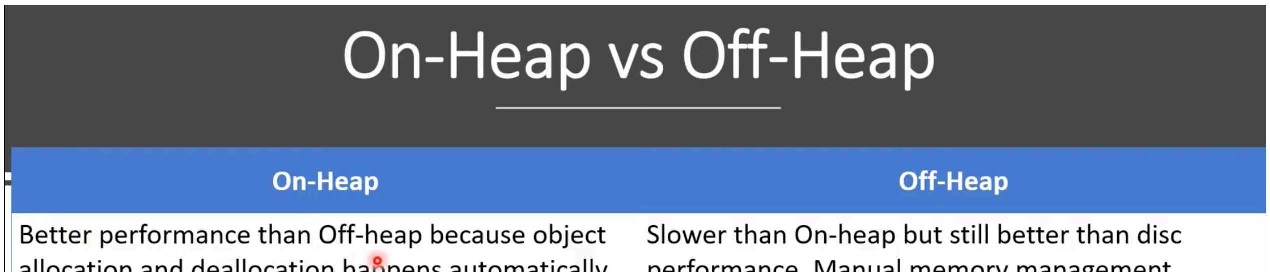
Spark can also use off-heap memory explicitly.

set
Spark.memory.offHeap.use to true to make use of off-heap memory and also assign size to this memory.

Using On-heap memory is fastest, using off-heap is slower than on-heap but still faster than using read/write on disk.

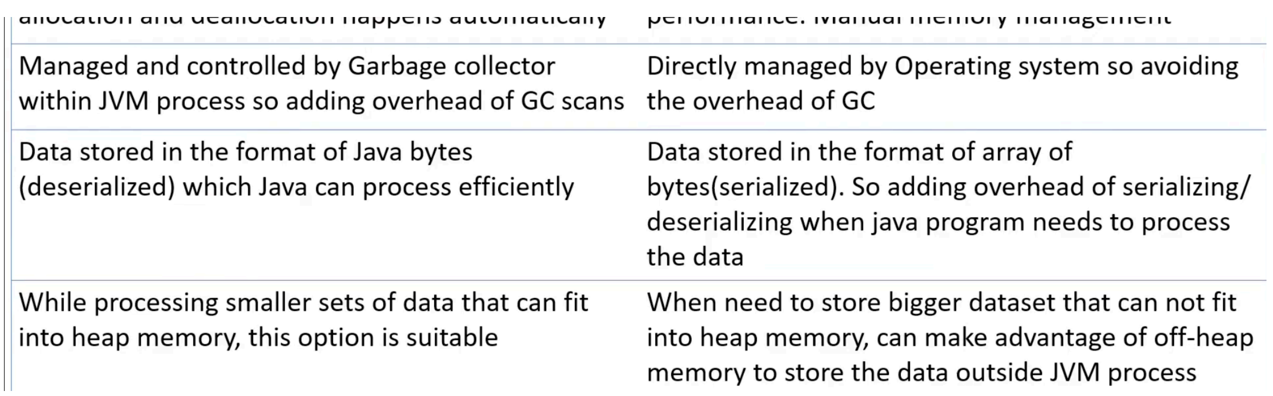
Garbage collector - part of JVM process, when on-heap of any executor is full ,garbage collector starts scanning the on-heap and start removing obsolete or old objects from on-heap to make space.
GC do not work on off-heap.
Having GC is both good and bad, in on-heap since GC will clean it automatically we do not have to worry about obsolete data objects.

When GC scan has to happen it will make programs wait before they can run on executor. This is disadvantage.



On-Heap vs Off-Heap

On-Heap	Off-Heap
Better performance than Off-heap because object allocation and deallocation happens automatically	Slower than On-heap but still better than disc performance. Manual memory management



allocation and deallocation happens automatically	performance. Manual memory management
Managed and controlled by Garbage collector within JVM process so adding overhead of GC scans	Directly managed by Operating system so avoiding the overhead of GC
Data stored in the format of Java bytes (deserialized) which Java can process efficiently	Data stored in the format of array of bytes(serialized). So adding overhead of serializing/ deserializing when java program needs to process the data
While processing smaller sets of data that can fit into heap memory, this option is suitable	When need to store bigger dataset that can not fit into heap memory, can make advantage of off-heap memory to store the data outside JVM process

On-Heap memory is managed by the JVM and is used to store Resilient Distributed Datasets (RDDs). When you first load data into Py Spark, it's stored in On-Heap memory as an RDD. Off-Heap memory, on the other hand, is managed by the operating system and is used for caching and storing serialized data.

> *“Of course! On-Heap memory refers to memory that is managed by the Java Virtual Machine (JVM), while Off-Heap memory refers to memory that is managed by the operating system.”*

Bob was relieved to hear the explanation, but he still didn’t understand why this was important for PySpark. The interviewer continued,

> *“In PySpark, On-Heap memory is used for storing RDDs, while Off-Heap memory is used for caching and storing serialized data.”*

Bob was starting to understand, but he was still a little confused. The interviewer noticed his confusion and decided to give him an example.

> *“Imagine you have a very large dataset of pet food orders. When you first load the data into PySpark, it’s stored in On-Heap memory as an RDD. If you want to cache that RDD for faster access later, PySpark will move it to Off-Heap memory. This can improve performance, but it can also use up more memory on your system.”*

Bob was starting to get it. “So, if I’m running out of memory, I should try to use Off-Heap memory more?” he asked.

“Exactly!” said the interviewer.

> *“Off-Heap memory can be more efficient for storing serialized data, but it can also be more difficult to manage. You’ll need to balance your usage of On-Heap and Off-Heap memory to get the best performance.”*

https://medium.com/@sathamn.n/spark-on-heap-and-off-heap-memory-in-pyspark-7016b48f8512