|  |  |
| --- | --- |
| **问题** | **答案** |
| Spark 运行模式 | * hadoop yarn   + yarn-cluster: 适合运行在生产环境。Spark Driver运行在由Yarn管理的AppMaster进程中，由AM负责向集群申请资源，并监督作业运行状态。Client在完成作业的提交后即可关掉，不影响作业在集群的运行。Client日志仅显示Job的运行状况，没有详细日志。   + yarn-client: 适合交互和调试。在此模式下，Spark Driver运行在Client端，负责与运行executor的container通信，而AppMaster仅负责向集群申请资源。Client端由Spark-submit开启的进程被kill，则运行在集群上的作业将被kill。 * 独立的spark集群 * mesos，Kubernetes等 |
| 如何理解Scala中体现的函数式编程 | 数据映射、变量不可变、没有副作用以及一等功能这四个方面回答。 |
| 什么是RDD | 1. 弹性式数据分布集  2. 逻辑上集中但是物理上是分散存储的  3. 具有只读属性，只能通过施加Transfromation生成新的RDD，无法改变原有的RDD  4. RDD之间保留着如何从其他RDD衍生出来的信息，可以做到失败恢复。 |
| 什么是宽依赖窄依赖 | 窄依赖：所有父RDD中的Partition均会一一映射到子RDD的Partition中，数据的分布没有发生变化。  宽依赖：父RDD中的Partition被切分，根据key不同划分到子RDD的不同分区中。 |
| Spark和MapReduce优缺点 | MapReduce  优点：模型简单已于理解，对资源使用量也少  缺点：   * 仅支持Map和Reduce两种操作 * Map中间结果需要写磁盘 * 任务调度和启动开销大 * 无法充分利用内存 * Map和Reduce都需要排序 * 不适合迭代计算   Spark  优点：   * 丰富的API（Java、Scala、Python、R四种语言，sort、join等高效算子） * DAG执行引擎，中间结果不落盘 * 线程池模型减少task启动开销 * 充分利用内存，减少磁盘IO * 避免不必要的排序操作   缺点：入门门槛较高，对资源使用量大 |
| 使用Spark实现数组平均数计算思路 | 使用aggregateByKey  **val** rdd = sc.parallelize(*Seq*(  (**"A"**,110),(**"A"**,130),(**"A"**,120),  (**"B"**,200),(**"B"**,206),(**"B"**,206),  (**"C"**,150),(**"C"**,160),(**"C"**,170))) **val** agg\_rdd = rdd.aggregateByKey((0,0))((acc, value) => (acc.\_1 + value, acc.\_2 + 1),(acc1, acc2) => (acc1.\_1 + acc2.\_1, acc1.\_2 + acc2.\_2)) **val** avg = agg\_rdd.mapValues(x => (x.\_1/x.\_2)) avg.collect |
| Spark内存 | * driver内存：由driver-memory参数指定，默认是512m，如果使用client模式运行且代码中有collect，take之类的调用，则可能需要调高此内存设置。 * nodemanager内存：   + yarn.nodemanager.resource.memory-mb：nm最多可以分配的内存量，默认是8g   + yarn.scheduler.maximum-allocation-mb：nm中每个任务（executor）可申请的最多的内存量，默认是1g * executor内存：分成三块，reserve memory（300m），user memory（除reserver外的25%），运行时memory（除reserver外的75%）。总的内存量=min(yarn.scheduler.maximum-allocation-mb, executor-memory)，其中运行时memory分为：   + storage内存：加载数据的内存占用，默认占50%   + execution内存：运行时内存占用，默认占50% * task内存：task运行时使用executor的内存 |
| Spark性能调优手段 | 1. 多使用mapPartitions  2. 结构化内容可以使用DataFrame或者DataSet  3. 数据结构上优化，使用numberic ID代替string作为key，尽量避免使用HashMap  4. 对反复用到的数据调用cache算子进行缓存，避免重复计算  5. 如果用到的堆小于32GB，可以 -XX:+UseCompressedOops 使得指针设置为4个字节，而不是默认的8个字节。  6. 观察GC，-verbose:gc -XX:+PrintGCDetails -XX:+PrintGCTimeStamps 打印GC信息，或者通过Spark的界面观察GC执行情况。  如果Full GC执行频繁，说明内存空间不足以执行任务，需要增大内存空间。  如果minor gc执行频繁，可以适当增大Eden空间。  再大堆上使用G1GC垃圾回收器。  7. 准确评估HDFS加载的文件大小，如果block是默认128mb，那么加载进入内存往往大小会是2到3倍。  8. sortByKey，reduceByKey以及join类似的操作时，可能会发生oom的情况，简单的方法是增加并行度。可以使用spark.default.parallism spark.sql.shuffle.partitions |