第三版：大数据 40 道

HDFS 写文件的步骤：

1、 Client 向 NameNode 提交上传某文件的申请。

2、 NameNode 返回响应给 Client，同意上传文件的申请。

3、 Client 向 NameNode 申请子节点 DataNode。

4、 NameNode 响应给 Client 它的子节点 DataNode。

5、 Client 向 DataNode 提交申请建立传输通道。

6、 DataNode 依次响应连接

7、 Client 向 DataNode 上传一个 Block，DataNode1 向其他子节点冗余文件。

HDFS 读取文件的步骤

1、 Client 向 NameNode 请求下载某文件。

2、 NameNode 向 Client 返回文件的元数据。

3、 Client 向 DataNode1 请求访问读数据 Block_1.

4、 DataNode1 向 Client 传输数据。

5、 Client 向 DataNode2 请求访问读数据 Block_2.

6、 DataNode2 向 Client 传输数据。

Hadoop 的 shuffle 过程

Map 端的 shuffle

Map 端会处理输入数据并产生中间结果，这个中间结果会写到本地磁盘，而不是 HDFS。每个 Map 的输出会先写到内存缓冲区中，当写入的数据达到设定的阈值时，系统将会启动一个线程将缓冲区的数据写到磁盘，这个过程叫做 spill。

在 spill 写入之前，会先进行二次排序，首先根据数据所属的 partition 进行排序，然后每个 partition 中的数据再按 key 来排序。partition 的目是将记录划分到不同的 Reducer 上去，以期望能够达到负载均衡，以后的 Reducer 就会根据 partition 来读取自己对应的数据。接着运行 combiner(如果设置了的话)，combiner 的本质也是一个 Reducer，其目的是对将要写入到磁盘上的文件先进行一次处理，这样，写入到磁盘的数据量就会减少。最后将数据写到本地磁盘产生 spill 文件(spill 文件保存在{mapred.local.dir}指定的目录中，Map 任务结束后就会被删除)。

最后，每个 Map 任务可能产生多个 spill 文件，在每个 Map 任务完成前，会通过多路归并算法将这些 spill 文件归并成一个文件。至此，Map 的 shuffle 过程就结束了。

Reduce 端的 shuffle

Reduce 端的 shuffle 主要包括三个阶段，copy、sort(merge)和 reduce。

首先要将 Map 端产生的输出文件拷贝到 Reduce 端，但每个 Reducer 如何知道自己应该处理哪些数据呢？

因为 Map 端进行 partition 的时候，实际上就相当于指定了每个 Reducer 要处理的数据(partition 就对应了 Reducer)，所以 Reducer 在拷贝数据的时候只需拷贝与自己对应的 partition 中的数据即可。

每个 Reducer 会处理一个或者多个 partition，但需要先将自己对应的 partition 中的数据从每个 Map 的输出结果中拷贝过来。

接下来就是 sort 阶段，也成为 merge 阶段，因为这个阶段的主要工作是执行了归并排序。

从 Map 端拷贝到 Reduce 端的数据都是有序的，所以很适合归并排序。

最终在 Reduce 端生成一个较大的文件作为 Reduce 的输入。

最后就是Reduce过程了，在这个过程中产生了最终的输出结果，并将其写到 HDFS 上。

fsimage 和 edit 的区别?

当 NN,SN 要进行数据同步时叫做 checkpoint 时就用到了 fsimage 与 edit，fsimage 是保存最新的元数据的信息，当 fsimage 数据到一定的大小事会去生成一个新的文件来保存元数据的信息，这个新的文件就是 edit，edit 会回滚最新的数据。

简单说一下 hadoop 的 map-reduce 模型

首先 map task会从本地文件系统读取数据,转换成 key-value 形式的键值对集合，使用的是 hadoop 内置的数据类型，如 Text，Longwritable 等。

将键值对集合输入 mapper 进行业务处理过程，将其转化成需要的 key-value 再输出。

之后会进行一个 partition 分区操作，默认使用的是 hashpartitioner，可以通过重写 hashpartitioner 的 getPartition 方法来自定义分区规则。

之后会对 key 进行 sort 排序，grouping 分组操作将相同 key 的 value 合并分组输出，在这里可以使用自定义的数据类型，重写 WritableComparator 的 Comparator 方法来自定义排序规则，重写 RawComparator 的 compara 方法来自定义分组规则。

之后进行一个 combiner 归约操作，就是一个本地的 reduce 预处理，以减小 shuffle，reducer 的工作量。

Reduce task 会用过网络将各个数据收集进行 reduce 处理，最后将数据保存或者显示，结束整个 job。

运行 hadoop 集群需要哪些守护进程?

DataNode,NameNode,TaskTracker 和 JobTracker 都是运行 Hadoop 集群需要的守护进程。

hadoop 的 TextInputFormat 作用是什么,如何自定义实现?

InputFormat 会在 map 操作之前对数据进行两方面的预处理。

1.是 getSplits，返回的是 InputSplit 数组，对数据进行 Split 分片，每片交给 map 操作一次。

2.是 getRecordReader，返回的是 RecordReader 对象，对每个 Split 分片进行转换为 key-value 键值对格式传递给 map 常用的 InputFormat 是 TextInputFormat，使用的是 LineRecordReader 对每个分片进行键值对的转换，以行偏移量作为键，行内容作为值。

3.自定义类继承 InputFormat 接口，重写 createRecordReader 和 isSplitable 方法在 createRecordReader 中可以自定义分隔符。

hadoop 和 spark 都是并行计算，那么他们有什么相同和区别？

两者都使用 mr 模型来进行并行计算，hadoop 的一个作业称为 job，job 里面分为 map task 和 reduce task，每个 task 都是在自己的进程中运行的，当 task 结束时，进程也会结束。

Spark 用户提交的任务称为 application，一个 application 对应一个 SparkContext，app 中存在多个 job，没触发一个 action 操作就会产生一个 job。

这些 job 可以并行或者串行执行，每个 job 有多个 stage，stage 是 shuffle 过程中 DAGSchaduler 通过 RDD 之间的依赖关系划分 job 而来的，每个 stage 里面有多个 task，组成 taskset 有 TaskSchaduler 分发到各个 executor 中执行，executor 的生命周期是和 application 一样的，即使没有 job 运行也是存在的，所以 task 可以快速启动读取内存进行计算的。

Hadoop 的 job 只有 map 和 reduce 操作，表达能力比较欠缺而且在 mr 过程中会重复的读写 hdfs，造成大量的 io 操作，多个 job 需要自己管理关系。

Spark 的迭代计算都是在内存中进行的，API 中提供了大量的 RDD 操作 join，groupby 等，而且通过 DAG 图可以实现良好的容错。

为什么要用 flume 导入 hdfs，hdfs 的架构是怎样的?

Flume 可以实时的导入数据到 hdfs 中，当 hdfs 上的文件达到一个指定大小的时候会形成一个文件，或者超时所指定时间的话也形成一个文件。

文件都是存储在 datanode 上的，namenode 存储着 datanode 的元数据信息，而 namenode 的元数据信息是存在内存中的，所以当文件切片很小或者很多的时候会卡死。

MR 程序运行的时候会有什么比较常见的问题?

比如说作业中大部分都完成了，但是总有几个 reduce 一直在运行。

这是因为这几个 reduce 中的处理的数据要远远大于其他的 reduce，可能是对键值对任务划分的不均匀造成的数据倾斜。

解决的方法可以在分区的时候重新定义分区规则对于 value 数据很多的 key 可以进行拆分、均匀打散等处理，或者是在 map 端的 combiner 中进行数据预处理的操作。

简单说一下 hadoop 和 spark 的 shuffle 过程

Hadoop：map 端保存分片数据，通过网络收集到 reduce 端。

Spark：spark 的 shuffle 实在 DAGSchedular 划分 Stage 的时候产生的，TaskSchedular 要分发 Stage 到各个 worker 的 executor，减少 shuffle 可以提高性能。

hive 中存放的是什么?

表。

存的是和 hdfs 的映射关系，hive 是逻辑上的数据仓库，实际操作的都是 hdfs 上的文件，HQL 就是用 SQL 语法来写的 MR 程序。

Hive 与关系型数据库的关系?

没有关系，hive 是数据仓库，不能和数据库一样进行实时的 CRUD 操作。

是一次写入多次读取的操作，可以看成是 ETL 的工具。

Flume 的工作及时是什么?

核心概念是 agent，里面包括 source，channel 和 sink 三个组件。

Source 运行在日志收集节点进行日志采集，之后临时存储在 channel 中，sink 负责将 channel 中的数据发送到目的地。

只有发送成功 channel 中的数据才会被删除。

首先书写 flume 配置文件，定义 agent、source、channel 和 sink 然后将其组装，执行 flume-ng 命令。

Hbase 行键列族的概念，物理模型，表的设计原则?

行键：是 hbase 表自带的，每个行键对应一条数据。

列族：是创建表时指定的，为列的集合，每个列族作为一个文件单独存储，存储的数据都是字节数组，其中数据可以有很多，通过时间戳来区分。

物理模型：整个 hbase 表会拆分成多个 region，每个 region 记录着行键的起始点保存在不同的节点上，查询时就是对各个节点的并行查询，当 region 很大时使用.META 表存储各个 region 的起始点，-ROOT 又可以存储.META 的起始点。

Rowkey 的设计原则：各个列族数据平衡，长度原则、相邻原则，创建表的时候设置表放入 regionserver 缓存中，避免自动增长和时间，使用字节数组代替 string，最大长度 64kb，最好 16 字节以内，按天分表，两个字节散列，四个字节存储时分毫秒。

列族的设计原则：尽可能少(按照列族进行存储，按照 region 进行读取，不必要的 io 操作)，经常和不经常使用的两类数据放入不同列族中，列族名字尽可能短。

请列出正常的 hadoop 集群中 hadoop 都分别需要启动 哪些进程，他们的作用分别都是什么，请尽量列的详细一些。

namenode：负责管理 hdfs 中文件块的元数据，响应客户端请求，管理 datanode 上文件 block 的均衡，维持副本数量

Secondname:主要负责做 checkpoint 操作；也可以做冷备，对一定范围内数据做快照性备份。

Datanode:存储数据块，负责客户端对数据块的 io 请求

Jobtracker :管理任务，并将任务分配给 tasktracker。

Tasktracker: 执行 JobTracker 分配的任务。

Resourcemanager、Nodemanager、Journalnode、Zookeeper、Zkfc

请说明 hive 中 Sort By、Order By、Cluster By，Distribute By 各代表什么意思?

order by：会对输入做全局排序，因此只有一个 reducer（多个 reducer 无法保证全局有序）。只有一个 reducer，会导致当输入规模较大时，需要较长的计算时间。

sort by：不是全局排序，其在数据进入 reducer 前完成排序。

distribute by：按照指定的字段对数据进行划分输出到不同的 reduce 中。

cluster by：除了具有 distribute by 的功能外还兼具 sort by 的功能。

HBase 简单读写流程?

读：

找到要读数据的 region 所在的 RegionServer，然后按照以下顺序进行读取：先去 BlockCache 读取，若 BlockCache 没有，则到 Memstore 读取，若 Memstore 中没有，则到 HFile 中去读。

写：

找到要写数据的 region 所在的 RegionServer，然后先将数据写到 WAL(Write-Ahead Logging，预写日志系统)中，然后再将数据写到 Memstore 等待刷新，回复客户端写入完成。

HBase 的特点是什么?

1、 hbase 是一个分布式的基于列式存储的数据库，基于 hadoop 的 HDFS 存储，zookeeper 进行管理。

2、 hbase 适合存储半结构化或非结构化数据，对于数据结构字段不够确定或者杂乱无章很难按一个概念去抽取的数据。

3、 hbase 为 null 的记录不会被存储。

4、 基于的表包括 rowkey，时间戳和列族。新写入数据时，时间戳更新，同时可以查询到以前的版本。

5、 hbase 是主从结构。Hmaster 作为主节点，hregionserver 作为从节点。

请描述如何解决 Hbase 中 region 太小和 region 太大带来的结果。

Region 过大会发生多次 compaction，将数据读一遍并写一遍到 hdfs 上，占用 io，region 过小会造成多次 split，region 会下线，影响访问服务，调整 hbase.heregion.max.filesize 为 256m。

海量日志数据，提取出某日访问百度次数最多的那个 IP。

首先是这一天，并且是访问百度的日志中的 IP 取出来，逐个写入到一个大文件中。注意到 IP 是 32 位的，最多有个 2^32 个 IP。同样可以采用映射的方法，比如模 1000，把整个大文件映射为 1000 个小文件，再找出每个小文中出现频率最大的 IP（可以采用 hash_map 进行频率统计，然后再找出频率最大的几个）及相应的频率。然后再在这 1000 个最大的 IP 中，找出那个频率最大的 IP，即为所求。

或者如下阐述：

算法思想：分而治之+Hash

1、 IP 地址最多有 2^32=4G 种取值情况，所以不能完全加载到内存中处理；

2、 可以考虑采用“分而治之”的思想，按照 IP 地址的 Hash(IP)24 值，把海量 IP 日志分别存储到 1024 个小文件中。这样，每个小文件最多包含 4MB 个 IP 地址；

3、 对于每一个小文件，可以构建一个 IP 为 key，出现次数为 value 的 Hash map，同时记录当前出现次数最多的那个 IP 地址；

4、 可以得到 1024 个小文件中的出现次数最多的 IP，再依据常规的排序算法得到总体上出现次数最多的 IP；

搜索引擎会通过日志文件把用户每次检索使用的所有检索串都记录下来，每个查询串的长度为 1-255 字节。

假设目前有一千万个记录（这些查询串的重复度比较高，虽然总数是 1 千万，但如果除去重复后，不超过 3 百万个。一个查询串的重复度越高，说明查询它的用户越多，也就是越热门。），请你统计最热门的 10 个查询串，要求使用的内存不能超过 1G。

典型的 Top K 算法，还是在这篇文章里头有所阐述，

文中，给出的最终算法是：

第一步、先对这批海量数据预处理，在 O（N）的时间内用 Hash 表完成统计（之前写成了排序，特此订正。July、2011.04.27）；

第二步、借助堆这个数据结构，找出 Top K，时间复杂度为 N ‘logK。

即，借助堆结构，我们可以在 log 量级的时间内查找和调整/移动。因此，维护一个 K(该题目中是 10)大小的小根堆，然后遍历 300 万的 Query，分别和根元素进行对比所以，我们最终的时间复杂度是：O（N） + N’ *O（logK），（N 为 1000 万，N’ 为 300 万）。ok，更多，详情，请参考原文。

或者：采用 trie 树，关键字域存该查询串出现的次数，没有出现为 0。最后用 10 个元素的最小推来对出现频率进行排序。

有一个 1G 大小的一个文件，里面每一行是一个词，词的大小不超过 16 字节，内存限制大小是 1M。返回频数最高的 100 个词。

方案：顺序读文件中，对于每个词 x，取 hash(x)P00，然后按照该值存到 5000 个小文件（记为 x0,x1,...x4999）中。这样每个文件大概是 200k 左右。

如果其中的有的文件超过了 1M 大小，还可以按照类似的方法继续往下分，直到分解得到的小文件的大小都不超过 1M。

对每个小文件，统计每个文件中出现的词以及相应的频率（可以采用 trie 树/hash_map 等），并取出出现频率最大的 100 个词（可以用含 100 个结点的最小堆），并把 100 个词及相应的频率存入文件，这样又得到了 5000 个文件。下一步就是把这 5000 个文件进行归并（类似与归并排序）的过程了。

有 10 个文件，每个文件 1G，每个文件的每一行存放的都是用户的 query，每个文件的 query 都可能重复。要求你按照 query 的频度排序。

还是典型的 TOP K 算法，解决方案如下：

方案 1：

顺序读取 10 个文件，按照 hash(query)的结果将 query 写入到另外 10 个文件（记为）中。这样新生成的文件每个的大小大约也 1G（假设 hash 函数是随机的）。

找一台内存在 2G 左右的机器，依次对用 hash_map(query, query_count)来统计每个 query 出现的次数。利用快速/堆/归并排序按照出现次数进行排序。将排序好的 query 和对应的 query_cout 输出到文件中。这样得到了 10 个排好序的文件（记为）。

对这 10 个文件进行归并排序（内排序与外排序相结合）。

方案 2：

一般 query 的总量是有限的，只是重复的次数比较多而已，可能对于所有的 query，一次性就可以加入到内存了。这样，我们就可以采用 trie 树/hash_map 等直接来统计每个 query 出现的次数，然后按出现次数做快速/堆/归并排序就可以了。

方案 3：

与方案 1 类似，但在做完 hash，分成多个文件后，可以交给多个文件来处理，采用分布式的架构来处理（比如 MapReduce），最后再进行合并。

给定 a、b 两个文件，各存放 50 亿个 url，每个 url 各占 64 字节，内存限制是 4G，让你找出 a、b 文件共同的 url？

方案 1：可以估计每个文件安的大小为 5G×64=320G，远远大于内存限制的 4G。所以不可能将其完全加载到内存中处理。考虑采取分而治之的方法。

遍历文件 a，对每个 url 求取 hash(url)00，然后根据所取得的值将 url 分别存储到 1000 个小文件（记为 a0,a1,…,a999）中。这样每个小文件的大约为 300M。

遍历文件 b，采取和 a 相同的方式将 url 分别存储到 1000 小文件（记为 b0,b1,…,b999）。这样处理后，所有可能相同的 url 都在对应的小文件（a0vsb0,a1vsb1,…,a999vsb999）中，不对应的小文件不可能有相同的 url。然后我们只要求出 1000 对小文件中相同的 url 即可。

求每对小文件中相同的 url 时，可以把其中一个小文件的 url 存储到 hash_set 中。然后遍历另一个小文件的每个 url，看其是否在刚才构建的 hash_set 中，如果是，那么就是共同的 url，存到文件里面就可以了。

方案 2：如果允许有一定的错误率，可以使用 Bloom filter，4G 内存大概可以表示 340 亿 bit。将其中一个文件中的 url 使用 Bloom filter 映射为这 340 亿 bit，然后挨个读取另外一个文件的 url，检查是否与 Bloom filter，如果是，那么该 url 应该是共同的 url（注意会有一定的错误率）。

Bloom filter 日后会在本 BLOG 内详细阐述。

在 2.5 亿个整数中找出不重复的整数，注，内存不足以容纳这 2.5 亿个整数。

方案 1：采用 2-Bitmap（每个数分配 2bit，00 表示不存在，01 表示出现一次，10 表示多次，11 无意义）进行，共需内存 2^32 * 2 bit=1 GB 内存，还可以接受。然后扫描这 2.5 亿个整数，查看 Bitmap 中相对应位，如果是 00 变 01，01 变 10，10 保持不变。所描完事后，查看 bitmap，把对应位是 01 的整数输出即可。

方案 2：也可采用与第 1 题类似的方法，进行划分小文件的方法。然后在小文件中找出不重复的整数，并排序。然后再进行归并，注意去除重复的元素。

腾讯面试题：给 40 亿个不重复的 unsigned int 的整数，没排过序的，然后再给一个数，如何快速判断这个数是否在那 40 亿个数当中?

与上第 6 题类似，我的第一反应时快速排序+二分查找。以下是其它更好的方法：

方案 1：oo，申请 512M 的内存，一个 bit 位代表一个 unsigned int 值。读入 40 亿个数，设置相应的 bit 位，读入要查询的数，查看相应 bit 位是否为 1，为 1 表示存在，为 0 表示不存在。

方案 2：这个问题在《编程珠玑》里有很好的描述，大家可以参考下面的思路，探讨一下：

又因为 2^32 为 40 亿多，所以给定一个数可能在，也可能不在其中；
这里我们把 40 亿个数中的每一个用 32 位的二进制来表示
假设这 40 亿个数开始放在一个文件中。

然后将这 40 亿个数分成两类:

1.最高位为 0
2.最高位为 1
并将这两类分别写入到两个文件中，其中一个文件中数的个数<=20 亿，而另一个>=20 亿（这相当于折半了）；

与要查找的数的最高位比较并接着进入相应的文件再查找
再然后把这个文件为又分成两类:
1.次最高位为 0
2.次最高位为 1
并将这两类分别写入到两个文件中，其中一个文件中数的个数<=10 亿，而另一个>=10 亿（这相当于折半了）；

与要查找的数的次最高位比较并接着进入相应的文件再查找。
……

以此类推，就可以找到了,而且时间复杂度为 O(logn)，方案 2 完。

附：这里，再简单介绍下，位图方法：

使用位图法判断整形数组是否存在重复

判断集合中存在重复是常见编程任务之一，当集合中数据量比较大时我们通常希望少进行几次扫描，这时双重循环法就不可取了。

位图法比较适合于这种情况，它的做法是按照集合中最大元素 max 创建一个长度为 max+1 的新数组，然后再次扫描原数组，遇到几就给新数组的第几位置上 1，如遇到 5 就给新数组的第六个元素置 1，这样下次再遇到 5 想置位时发现新数组的第六个元素已经是 1 了，这说明这次的数据肯定和以前的数据存在着重复。这种给新数组初始化时置零其后置一的做法类似于位图的处理方法故称位图法。它的运算次数最坏的情况为 2N。如果已知数组的最大值即能事先给新数组定长的话效率还能提高一倍。

欢迎，有更好的思路，或方法，共同交流。

怎么在海量数据中找出重复次数最多的一个?

方案 1：先做 hash，然后求模映射为小文件，求出每个小文件中重复次数最多的一个，并记录重复次数。然后找出上一步求出的数据中重复次数最多的一个就是所求（具体参考前面的题）。

上千万或上亿数据（有重复），统计其中出现次数最多的前 N 个数据。

方案 1：上千万或上亿的数据，现在的机器的内存应该能存下。所以考虑采用 hash_map/搜索二叉树/红黑树等来进行统计次数。然后就是取出前 N 个出现次数最多的数据了，可以用第 2 题提到的堆机制完成。

一个文本文件，大约有一万行，每行一个词，要求统计出其中最频繁出现的前 10 个词，请给出思想，给出时间复杂度分析。

方案 1：这题是考虑时间效率。用 trie 树统计每个词出现的次数，时间复杂度是 O(n_le)（le 表示单词的平准长度）。然后是找出出现最频繁的前 10 个词，可以用堆来实现，前面的题中已经讲到了，时间复杂度是 O(n_lg10)。所以总的时间复杂度，是 O(n_le)与 O(n_lg10)中较大的哪一个。

附、100w 个数中找出最大的 100 个数。

方案 2：在前面的题中，我们已经提到了，用一个含 100 个元素的最小堆完成。复杂度为 O(100w*lg100)。

方案 3：采用快速排序的思想，每次分割之后只考虑比轴大的一部分，知道比轴大的一部分在比 100 多的时候，采用传统排序算法排序，取前 100 个。复杂度为 O(100w*100)。

方案 4：采用局部淘汰法。选取前 100 个元素，并排序，记为序列 L。然后一次扫描剩余的元素 x，与排好序的 100 个元素中最小的元素比，如果比这个最小的要大，那么把这个最小的元素删除，并把 x 利用插入排序的思想，插入到序列 L 中。依次循环，知道扫描了所有的元素。复杂度为 O(100w*100)。

spark 的优化怎么做?

spark 调优比较复杂，但是大体可以分为三个方面来进行

平台层面的调优：防止不必要的 jar 包分发，提高数据的本地性，选择高效的存储格式如 parquet

应用程序层面的调优：过滤操作符的优化降低过多小任务，降低单条记录的资源开销，处理数据倾斜，复用 RDD 进行缓存，作业并行化执行等等

JVM 层面的调优：设置合适的资源量，设置合理的 JVM，启用高效的序列化方法如 kyro，增大 off head 内存等等

数据本地性是在哪个环节确定的?

具体的 task 运行在那他机器上，dag 划分 stage 的时候确定的

RDD 的弹性表现在哪几点?

1、 自动的进行内存和磁盘的存储切换；

2、 基于 Lineage 的高效容错；

3、 task 如果失败会自动进行特定次数的重试；

4、 stage 如果失败会自动进行特定次数的重试，而且只会计算失败的分片；

5、 checkpoint 和 persist，数据计算之后持久化缓存；

6、 数据调度弹性，DAG TASK 调度和资源无关；

7、 数据分片的高度弹性。

RDD 有哪些缺陷?

不支持细粒度的写和更新操作（如网络爬虫），spark 写数据是粗粒度的。所谓粗粒度，就是批量写入数据，为了提高效率。但是读数据是细粒度的也就是说可以一条条的读。

不支持增量迭代计算，Flink 支持

Spark 的 shuffle 过程?

从下面三点去展开

1、 shuffle 过程的划分

2、 shuffle 的中间结果如何存储

3、 shuffle 的数据如何拉取过来

Spark 的数据本地性有哪几种?

Spark 中的数据本地性有三种：

1、 PROCESS_LOCAL 是指读取缓存在本地节点的数据

2、 NODE_LOCAL 是指读取本地节点硬盘数据

3、 ANY 是指读取非本地节点数据

通常读取数据 PROCESS_LOCAL>NODE_LOCAL>ANY，尽量使数据以 PROCESS_LOCAL或 NODE_LOCAL方式读取。其中 PROCESS_LOCAL还和 cache 有关，如果 RDD 经常用的话将该 RDD cache 到内存中，注意，由于 cache 是 lazy 的，所以必须通过一个 action 的触发，才能真正的将该 RDD cache 到内存中。

Spark 为什么要持久化，一般什么场景下要进行 persist 操作?

为什么要进行持久化?

spark 所有复杂一点的算法都会有 persist 身影，spark 默认数据放在内存，spark 很多内容都是放在内存的，非常适合高速迭代，1000 个步骤只有第一个输入数据，中间不产生临时数据，但分布式系统风险很高，所以容易出错，就要容错，rdd 出错或者分片可以根据血统算出来，如果没有对父 rdd 进行 persist 或者 cache 的化，就需要重头做。

以下场景会使用 persist

1、 某个步骤计算非常耗时，需要进行 persist 持久化

2、 计算链条非常长，重新恢复要算很多步骤，很好使，persist

3、 checkpoint 所在的 rdd 要持久化 persist。checkpoint 前，要持久化，写个 rdd.cache 或者 rdd.persist，将结果保存起来，再写 checkpoint 操作，这样执行起来会非常快，不需要重新计算 rdd 链条了。checkpoint 之前一定会进行 persist。

4、 shuffle 之后要 persist，shuffle 要进性网络传输，风险很大，数据丢失重来，恢复代价很大

5、 shuffle 之前进行 persist，框架默认将数据持久化到磁盘，这个是框架自动做的。

介绍一下 join 操作优化经验?

join 其实常见的就分为两类： map-side join 和 reduce-side join。当大表和小表 join 时，用 map-side join 能显著提高效率。将多份数据进行关联是数据处理过程中非常普遍的用法，不过在分布式计算系统中，这个问题往往会变的非常麻烦，因为框架提供的 join 操作一般会将所有数据根据 key 发送到所有的 reduce 分区中去，也就是 shuffle 的过程。造成大量的网络以及磁盘 IO 消耗，运行效率极其低下，这个过程一般被称为 reduce-side-join。如果其中有张表较小的话，我们则可以自己实现在 map 端实现数据关联，跳过大量数据进行 shuffle 的过程，运行时间得到大量缩短，根据不同数据可能会有几倍到数十倍的性能提升。

备注：这个题目面试中非常非常大概率见到，务必搜索相关资料掌握，这里抛砖引玉。