数据采集面试题

Flume

1. Flume 使用场景

线上数据一般主要是落地（存储到磁盘）或者通过 socket 传输给另外一个系统，这种情况下，你很难推动线上应用或服务去修改接口，实现直接向 kafka 里写数据，这时候你可能就需要 flume 这样的系统帮你去做传输。

2. Flume 丢包问题

单机 upd 的 flume source 的配置，100+M/s 数据量，10w qps flume 就开始大量丢包，因此很多公司在搭建系统时，抛弃了 Flume，自己研发传输系统，但是往往会参考 Flume 的 Source-Channel-Sink 模式。

一些公司在 Flume 工作过程中，会对业务日志进行监控，例如 Flume agent 中有多少条日志，Flume 到 Kafka 后有多少条日志等等，如果数据丢失保持在 1%左右是没有问题的，当数据丢失达到 5%左右时就必须采取相应措施。

3. Flume 与 Kafka 的选取

采集层主要可以使用 Flume、Kafka 两种技术。

Flume：Flume 是管道流方式，提供了很多的默认实现，让用户通过参数部署，及扩展 API。

Kafka：Kafka 是一个可持久化的分布式的消息队列。

Kafka 是一个非常通用的系统。你可以有许多生产者和很多的消费者共享多个主题 Topics。相比之下，Flume 是一个专用工具被设计为旨在往 HDFS，HBase 发送数据。它对 HDFS 有特殊的优化，并且集成了 Hadoop 的安全特性。所以，Cloudera 建议如果数据被多个系统消费的话，使用 kafka；如果数据被设计给 Hadoop 使用，使用 Flume。

正如你们所知 Flume 内置很多的 source 和 sink 组件。然而，Kafka 明显有一个更小的生产消费者生态系统，并且 Kafka 的社区支持不好。希望将来这种情况会得到改善，但是目前：使用 Kafka 意味着你准备好了编写你自己的生产者和消费者代码。如果已经存在的 Flume Sources 和 Sinks 满足你的需求，并且你更喜欢不需要任何开发的系统，请使用 Flume。

Flume 可以使用拦截器实时处理数据。这些对数据屏蔽或者过量是很有用的。Kafka 需要外部的流处理系统才能做到。

Kafka 和 Flume 都是可靠的系统，通过适当的配置能保证零数据丢失。然而，Flume 不支持副本事件。于是，如果 Flume 代理的一个节点奔溃了，即使使用了可靠的文件管道方式，你也将丢失这些事件直到你恢复这些磁盘。如果你需要一个高可靠性的管道，那么使用 Kafka 是个更好的选择。

Flume 和 Kafka 可以很好地结合起来使用。如果你的设计需要从 Kafka 到 Hadoop 的流数据，使用 Flume 代理并配置 Kafka 的 Source 读取数据也是可行的：你没有必要实现自己的消费者。你可以直接利用 Flume 与 HDFS 及

HBase 的结合的所有好处。你可以使用 Cloudera Manager 对消费者的监控，并且你甚至可以添加拦截器进行一些流处理。

4. 数据怎么采集到 Kafka，实现方式

使用官方提供的 flumeKafka 插件，插件的实现方式是自定义了 flume 的 sink，将数据从 channle 中取出，通过 kafka 的 producer 写入到 kafka 中，可以自定义分区等。

5. flume 管道内存，flume 宕机了数据丢失怎么解决

1）Flume 的 channel 分为很多种，可以将数据写入到文件。

2） 防止非首个 agent 宕机的方法数可以做集群或者主备。

6. flume 配置方式，flume 集群（详细讲解下）

Flume 的配置围绕着 source、channel、sink 叙述，flume 的集群是做在 agent 上的，而非机器上。

7. flume 不采集 Nginx 日志，通过 Logger4j 采集日志，优缺点是什么？

优点：Nginx 的日志格式是固定的，但是缺少 sessionid，通过 logger4j 采集的日志是带有 sessionid 的，而 session 可以通过 redis 共享，保证了集群日志中的同一 session 落到不同的 tomcat 时，sessionId 还是一样的，而且 logger4j 的方式比较稳定，不会宕机。

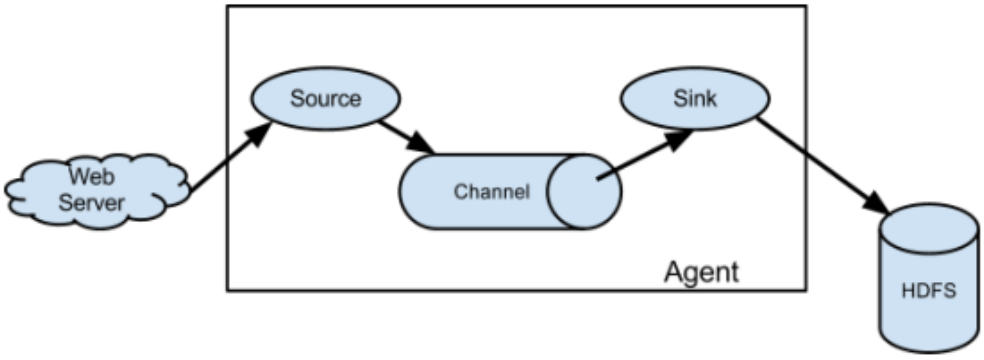
缺点：不够灵活，logger4j 的方式和项目结合过于紧密，而 flume 的方式比较灵活，拔插式比较好，不会影响项目性能。

8. flume 和 kafka 采集日志区别，采集日志时中间停了，怎么记录之前的日志？

Flume 采集日志是通过流的方式直接将日志收集到存储层，而 kafka 是将缓存在 kafka 集群，待后期可以采集到存储层。

Flume 采集中间停了，可以采用文件的方式记录之前的日志，而 kafka 是采用 offset 的方式记录之前的日志。

9. flume 有哪些组件，flume 的 source、channel、sink 具体是做什么的



1）source：用于采集数据，Source 是产生数据流的地方，同时 Source 会将产生的数据流传输到 Channel，这个有点类似于 Java IO 部分的 Channel。

2）channel：用于桥接 Sources 和 Sinks，类似于一个队列。

3）sink：从 Channel 收集数据，将数据写到目标源(可以是下一个 Source，也可以是 HDFS 或者 HBase)。

注意：要熟悉 source、channel、sink 的类型

数据存储面试题

HDFS

1. 请说下 HDFS 读写流程

HDFS 写流程

1）client 客户端发送上传请求，通过 RPC 与 namenode 建立通信，namenode 检查该用户是否有上传权限，以及上传的文件是否在 hdfs 对应的目录下重名，如果这两者有任意一个不满足，则直接报错，如果两者都满足，则返回给客户端一个可以上传的信息

2）client 根据文件的大小进行切分，默认 128M 一块，切分完成之后给 namenode 发送请求第一个 block 块上传到哪些服务器上

3）namenode 收到请求之后，根据网络拓扑和机架感知以及副本机制进行文件分配，返回可用的 DataNode 的地址

4）客户端收到地址之后与服务器地址列表中的一个节点如 A 进行通信，本质上就是 RPC 调用，建立 pipeline，A 收到请求后会继续调用 B，B 在调用 C，将整个 pipeline 建立完成，逐级返回 client

5）client 开始向 A 上发送第一个 block（先从磁盘读取数据然后放到本地内存缓存），以 packet（数据包，64kb）为单位，A 收到一个 packet 就会发送给 B，然后 B 发送给 C，A 每传完一个 packet 就会放入一个应答队列等待应答

6）数据被分割成一个个的 packet 数据包在 pipeline 上依次传输，在 pipeline 反向传输中，逐个发送 ack（命令正确应答），最终由 pipeline 中第一个 DataNode 节点 A 将 pipelineack 发送给 Client

7）当一个 block 传输完成之后, Client 再次请求 NameNode 上传第二个 block ，namenode 重新选择三台 DataNode 给 client

HDFS 读流程

1）client 向 namenode 发送 RPC 请求。请求文件 block 的位置

2）namenode 收到请求之后会检查用户权限以及是否有这个文件，如果都符合，则会视情况返回部分或全部的 block 列表，对于每个 block，NameNode 都会返回含有该 block 副本的 DataNode 地址； 这些返回的 DN 地址，会按照集群拓扑结构得出 DataNode 与客户端的距离，然后进行排序，排序两个规则：网络拓扑结构中距离 Client 近的排靠前；心跳机制中超时汇报的 DN 状态为 STALE，这样的排靠后

3）Client 选取排序靠前的 DataNode 来读取 block，如果客户端本身就是 DataNode,那么将从本地直接获取数据(短路读取特性)

4）底层上本质是建立 Socket Stream（FSDataInputStream），重复的调用父类 DataInputStream 的 read 方法，直到这个块上的数据读取完毕

5）当读完列表的 block 后，若文件读取还没有结束，客户端会继续向 NameNode 获取下一批的 block 列表

6）读取完一个 block 都会进行 checksum 验证，如果读取 DataNode 时出现错误，客户端会通知 NameNode，然后再从下一个拥有该 block 副本的 DataNode 继续读

7）read 方法是并行的读取 block 信息，不是一块一块的读取；NameNode 只是返回 Client 请求包含块的 DataNode 地址，并不是返回请求块的数据

8） 最终读取来所有的 block 会合并成一个完整的最终文件

2. HDFS 在读取文件的时候,如果其中一个块突然损坏了怎么办

客户端读取完 DataNode 上的块之后会进行 checksum 验证，也就是把客户端读取到本地的块与 HDFS 上的原始块进行校验，如果发现校验结果不一致，客户端会通知 NameNode，然后再从下一个拥有该 block 副本的 DataNode 继续读。

3. HDFS 在上传文件的时候,如果其中一个 DataNode 突然挂掉了怎么办

客户端上传文件时与 DataNode 建立 pipeline 管道，管道正向是客户端向 DataNode 发送的数据包，管道反向是 DataNode 向客户端发送 ack 确认，也就是正确接收到数据包之后发送一个已确认接收到的应答，当 DataNode 突然挂掉了，客户端接收不到这个 DataNode 发送的 ack 确认

，客户端会通知 NameNode，NameNode 检查该块的副本与规定的不符，NameNode 会通知 DataNode 去复制副本，并将挂掉的 DataNode 作下线处理，不再让它参与文件上传与下载。

4. 请说下 HDFS 的组织架构

1）Client：客户端

（1）切分文件。文件上传 HDFS 的时候，Client 将文件切分成一个一个的 Block，然后进行存储

（2）与 NameNode 交互，获取文件的位置信息

（3）与 DataNode 交互，读取或者写入数据

（4）Client 提供一些命令来管理 HDFS，比如启动关闭 HDFS、访问 HDFS 目录及内容等

2）NameNode：名称节点，也称主节点，存储数据的元数据信息，不存储具体的数据

（1）管理 HDFS 的名称空间

（2）管理数据块（Block）映射信息

（3）配置副本策略

（4）处理客户端读写请求

3）DataNode：数据节点，也称从节点。NameNode 下达命令，DataNode 执行实际的操作

（1）存储实际的数据块

（2）执行数据块的读/写操作

4）Secondary NameNode：并非 NameNode 的热备。当 NameNode 挂掉的时候，它并不能马上替换 NameNode 并提供服务

（1）辅助 NameNode，分担其工作量

（2）定期合并 Fsimage 和 Edits，并推送给 NameNode

（3）在紧急情况下，可辅助恢复 NameNode

Kafka

5. 为什么要使用 kafka？

缓冲和削峰：上游数据时有突发流量，下游可能扛不住，或者下游没有足够多的机器来保证冗余，kafka 在中间可以起到一个缓冲的作用，把消息暂存在 kafka 中，下游服务就可以按照自己的节奏进行慢慢处理。解耦和扩展性：项目开始的时候，并不能确定具体需求。消息队列可以作为一个接口层，解耦重要的业务流程。只需要遵守约定，针对数据编程即可获取扩展能力。冗余：可以采用一对多的方式，一个生产者发布消息，可以被多个订阅 topic 的服务消费到，供多个毫无关联的业务使用。健壮性：消息队列可以堆积请求，所以消费端业务即使短时间死掉，也不会影响主要业务的正常进行。异步通信：很多时候，用户不想也不需要立即处理消息。消息队列提供了异步处理机制，允许用户把一个消息放入队列，但并不立即处理它。想向队列中放入多少消息就放多少，然后在需要的时候再去处理它们。

6. Kafka 消费过的消息如何再消费？

kafka 消费消息的 offset 是定义在 zookeeper 中的， 如果想重复消费 kafka 的消息，可以在 redis 中自己记录 offset 的 checkpoint 点（n 个），当想重复消费消息时，通过读取 redis 中的 checkpoint 点进行 zookeeper 的 offset 重设，这样就可以达到重复消费消息的目的了

7. kafka 的数据是放在磁盘上还是内存上，为什么速度会快？

kafka 使用的是磁盘存储。

速度快是因为：

顺序写入：因为硬盘是机械结构，每次读写都会寻址->写入，其中寻址是一个“机械动作”，它是耗时的。所以硬盘 “讨厌”随机 I/O， 喜欢顺序 I/O。为了提高读写硬盘的速度，Kafka 就是使用顺序 I/O。Memory Mapped Files（内存映射文件）：64 位操作系统中一般可以表示 20G 的数据文件，它的工作原理是直接利用操作系统的 Page 来实现文件到物理内存的直接映射。完成映射之后你对物理内存的操作会被同步到硬盘上。Kafka 高效文件存储设计： Kafka 把 topic 中一个 parition 大文件分成多个小文件段，通过多个小文件段，就容易定期清除或删除已经消费完文件，减少磁盘占用。通过索引信息可以快速定位

message 和确定 response 的 大 小。通过 index 元数据全部映射到 memory（内存

映射文件），

可以避免 segment file 的 IO 磁盘操作。通过索引文件稀疏存储，可以大幅降低 index 文件元数据占用空间大小。

8. Kafka 数据怎么保障不丢失？

分三个点说，一个是生产者端，一个消费者端，一个 broker 端。

生产者数据的不丢失

kafka 的 ack 机制：在 kafka 发送数据的时候，每次发送消息都会有一个确认反馈机制，确保消息正常的能够被收到，其中状态有 0，1，-1。

如果是同步模式：

ack 设置为 0，风险很大，一般不建议设置为 0。即使设置为 1，也会随着 leader 宕机丢失数据。所以如果要严格保证生产端数据不丢失，可设置为-1。

如果是异步模式：

也会考虑 ack 的状态，除此之外，异步模式下的有个 buffer，通过 buffer 来进行控制数据的发送，有两个值来进行控制，时间阈值与消息的数量阈值，如果 buffer 满了数据还没有发送出去，有个选项是配置是否立即清空 buffer。可以设置为-1，永久阻塞，也就数据不再生产。异步模式下，即使设置为-1。也可能因为程序员的不科学操作，操作数据丢失，比如 kill -9，但这是特别的例外情况。

> 注：
>
> ack=0：producer 不等待 broker 同步完成的确认，继续发送下一条(批)信息。

> ack=1（默认）：producer 要等待 leader 成功收到数据并得到确认，才发送下一条 message。
>
> ack=-1：producer 得到 follwer 确认，才发送下一条数据。

消费者数据的不丢失

通过 offset commit 来保证数据的不丢失，kafka 自己记录了每次消费的 offset 数值，下次继续消费的时候，会接着上次的 offset 进行消费。

而 offset 的信息在 kafka0.8 版本之前保存在 zookeeper 中，在 0.8 版本之后保存到 topic 中，即使消费者在运行过程中挂掉了，再次启动的时候会找到 offset 的值，找到之前消费消息的位置，接着消费，由于 offset 的信息写入的时候并不是每条消息消费完成后都写入的，所以这种情况有可能会造成重复消费，但是不会丢失消息。

唯一例外的情况是，我们在程序中给原本做不同功能的两个 consumer 组设置 KafkaSpoutConfig.bulider.setGroupid 的时候设置成了一样的 groupid，这种情况会导致这两个组共享同一份数据，就会产生组 A 消费 partition1，partition2 中的消息，组 B 消费 partition3 的消息，这样每个组消费的消息都会丢失，都是不完整的。为了保证每个组都独享一份消息数据，groupid 一定不要重复才行。

kafka 集群中的 broker 的数据不丢失

每个 broker 中的 partition 我们一般都会设置有 replication（副本）的个数，生产者写入的时候首先根据分发策略（有 partition 按 partition，有 key 按 key，都没有轮询）写入到 leader 中，follower（副本）再跟 leader 同步数据，这样有了备份，也可以保证消息数据的不丢失。

9. 采集数据为什么选择 kafka？

采集层 主要可以使用 Flume, Kafka 等技术。

Flume：Flume 是管道流方式，提供了很多的默认实现，让用户通过参数部署，及扩展 API.

Kafka：Kafka 是一个可持久化的分布式的消息队列。 Kafka 是一个非常通用的系统。你可以有许多生产者和很多的消费者共享多个主题 Topics。

相比之下,Flume 是一个专用工具被设计为旨在往 HDFS，HBase 发送数据。它对 HDFS 有特殊的优化，并且集成了 Hadoop 的安全特性。

所以，Cloudera 建议如果数据被多个系统消费的话，使用 kafka；如果数据被设计给 Hadoop 使用，使用 Flume。

10. kafka 重启是否会导致数据丢失？

kafka 是将数据写到磁盘的，一般数据不会丢失。但是在重启 kafka 过程中，如果有消费者消费消息，那么 kafka 如果来不及提交 offset，可能会造成数据的不准确（丢失或者重复消费）。

11. kafka 宕机了如何解决？

先考虑业务是否受到影响

kafka 宕机了，首先我们考虑的问题应该是所提供的服务是否因为宕机的机器而受到影响，如果服务提供没问题，如果实现做好了集群的容灾机制，那么这块就不用担心了。

节点排错与恢复

想要恢复集群的节点，主要的步骤就是通过日志分析来查看节点宕机的原因，从而解决，重新恢复节点。

12. 为什么 Kafka 不支持读写分离？

在 Kafka 中，生产者写入消息、消费者读取消息的操作都是与 leader 副本进行交互的，从 而实现的是一种**主写主读**的生产消费模型。

Kafka 并不支持**主写从读**，因为主写从读有 2 个很明显的缺点:

数据一致性问题：数据从主节点转到从节点必然会有一个延时的时间窗口，这个时间窗口会导致主从节点之间的数据不一致。某一时刻，在主节点和从节点中 A 数据的值都为 X， 之后将主节点中 A 的值修改为 Y，那么在这个变更通知到从节点之前，应用读取从节点中的 A 数据的值并不为最新的 Y，由此便产生了数据不一致的问题。延时问题：类似 Redis 这种组件，数据从写入主节点到同步至从节点中的过程需要经历 网络→主节点内存→网络→从节点内存 这几个阶段，整个过程会耗费一定的时间。而在 Kafka 中，主从同步会比 Redis 更加耗时，它需要经历 网络→主节点内存→主节点磁盘→网络→从节 点内存→从节点磁盘 这几个阶段。对延时敏感的应用而言，主写从读的功能并不太适用。

而 kafka 的**主写主读**的优点就很多了：

可以简化代码的实现逻辑，减少出错的可能; 将负载粒度细化均摊，与主写从读相比，不仅负载效能更好，而且对用户可控;没有延时的影响;在副本稳定的情况下，不会出现数据不一致的情况。

13. kafka 数据分区和消费者的关系？

每个分区只能由同一个消费组内的一个消费者(consumer)来消费，可以由不同的消费组的消费者来消费，同组的消费者则起到并发的效果。

14. kafka 的数据 offset 读取流程

连接 ZK 集群，从 ZK 中拿到对应 topic 的 partition 信息和 partition 的 Leader 的相关信息连接到对应 Leader 对应的 brokerconsumer 将自自己己保存的 offset 发送给 LeaderLeader 根据 offset 等信息定位到 segment（索引文文件和日日志文文件）根据索引文文件中的内容，定位到日日志文文件中该偏移量量对应的开始位置读取相应长长度的数据并返回给 consumer

15. kafka 内部如何保证顺序，结合外部组件如何保证消费者的顺序？

kafka 只能保证 partition 内是有序的，但是 partition 间的有序是没办法的。爱奇艺的搜索架构，是从业务上把需要有序的打到同一个 partition。

16. Kafka 消息数据积压，Kafka 消费能力不足怎么处理？

如果是 Kafka 消费能力不足，则可以考虑增加 Topic 的分区数，并且同时提升消费组的消费者数量，消费者数=分区数。（两者缺一不可）如果是下游的数据处理不及时：提高每批次拉取的数量。批次拉取数据过少（拉取数据/处理时间<生产速度），使处理的数据小于生产的数据，也会造成数据积压。

17. Kafka 单条日志传输大小

kafka 对于消息体的大小默认为单条最大值是 1M 但是在我们应用场景中, 常常会出现一条消息大于 1M，如果不对 kafka 进行配置。则会出现生产者无法将消息推送到 kafka 或消费者无法去消费 kafka 里面的数据, 这时我们就要对 kafka 进行以下配置：

server.properties

1replica.fetch.max.bytes: 1048576 broker 可复制的消息的最大字节数, 默认为 1M

2message.max.bytes: 1000012 kafka 会接收单个消息 size 的最大限制， 默认为 1M 左右

注意：message.max.bytes 必须小于等于 replica.fetch.max.bytes，否则就会导致 replica 之间数据同步失败。

数据处理面试题

Flink

1. Flink checkpoint 与 Spark Flink 有什么区别或优势吗

spark streaming 的 checkpoint 仅仅是针对 driver 的故障恢复做了数据和元数据的 checkpoint。而 flink 的 checkpoint 机制 要复杂了很多，它采用的是轻量级的分布式快照，实现了每个算子的快照，及流动中的数据的快照。

2. Flink 中的 Time 有哪几种

在 flink 中被划分为事件时间，提取时间，处理时间三种。

如果以 EventTime 为基准来定义时间窗口那将形成 EventTimeWindow,要求消息本身就应该携带 EventTime。如果以 IngesingtTime 为基准来定义时间窗口那将形成 IngestingTimeWindow,以 source 的 systemTime 为准。如果以 ProcessingTime 基准来定义时间窗口那将形成 ProcessingTimeWindow，以 operator 的 systemTime 为准。

3. 对于迟到数据是怎么处理的

Flink 中 WaterMark 和 Window 机制解决了流式数据的乱序问题，对于因为延迟而顺序有误的数据，可以根据 eventTime 进行业务处理，对于延迟的数据 Flink 也有自己的解决办法，主要的办法是给定一个允许延迟的时间，在该时间范围内仍可以接受处理延迟数据

设置允许延迟的时间是通过 allowedLateness(lateness: Time)设置

保存延迟数据则是通过 sideOutputLateData(outputTag: OutputTag[T])保存

获取延迟数据是通过 DataStream.getSideOutput(tag: OutputTag[X])获取

4. Flink 的运行必须依赖 Hadoop 组件吗

Flink 可以完全独立于 Hadoop，在不依赖 Hadoop 组件下运行。但是做为大数据的基础设施，Hadoop 体系是任何大数据框架都绕不过去的。Flink 可以集成众多 Hadooop 组件，例如 Yarn、Hbase、HDFS 等等。例如，Flink 可以和 Yarn 集成做资源调度，也可以读写 HDFS，或者利用 HDFS 做检查点。

5. Flink 集群有哪些角色？各自有什么作用

有以下三个角色：

JobManager 处理器：

也称之为 Master，用于协调分布式执行，它们用来调度 task，协调检查点，协调失败时恢复等。Flink 运行时至少存在一个 master 处理器，如果配置高可用模式则会存在多个 master 处理器，它们其中有一个是 leader，而其他的都是 standby。

TaskManager 处理器：

也称之为 Worker，用于执行一个 dataflow 的 task(或者特殊的 subtask)、数据缓冲和 data stream 的交换，Flink 运行时至少会存在一个 worker 处理器。

Clint 客户端：

Client 是 Flink 程序提交的客户端，当用户提交一个 Flink 程序时，会首先创建一个 Client，该 Client 首先会对用户提交的 Flink 程序进行预处理，并提交到 Flink 集群中处理，所以 Client 需要从用户提交的 Flink 程序配置中获取 JobManager 的地址，并建立到 JobManager 的连接，将 Flink Job 提交给 JobManager

6. Flink 资源管理中 Task Slot 的概念

在 Flink 中每个 TaskManager 是一个 JVM 的进程, 可以在不同的线程中执行一个或多个子任务。

为了控制一个 worker 能接收多少个 task。worker 通过 task slot（任务槽）来进行控制（一个 worker 至少有一个 task slot）。

7. Flink 的重启策略了解吗

Flink 支持不同的重启策略，这些重启策略控制着 job 失败后如何重启：

固定延迟重启策略

固定延迟重启策略会尝试一个给定的次数来重启 Job，如果超过了最大的重启次数，Job 最终将失败。在连续的两次重启尝试之间，重启策略会等待一个固定的时间。

失败率重启策略

失败率重启策略在 Job 失败后会重启，但是超过失败率后，Job 会最终被认定失败。

在两个连续的重启尝试之间，重启策略会等待一个固定的时间。

无重启策略

Job 直接失败，不会尝试进行重启。

8. Flink 是如何保证 Exactly-once 语义的

Flink 通过实现**两阶段提交**和状态保存来实现端到端的一致性语义。分为以下几个步骤：

开始事务（beginTransaction）创建一个临时文件夹，来写把数据写入到这个文件夹里面

预提交（preCommit）将内存中缓存的数据写入文件并关闭

正式提交（commit）将之前写完的临时文件放入目标目录下。这代表着最终的数据会有一些延迟

丢弃（abort）丢弃临时文件

若失败发生在预提交成功后，正式提交前。可以根据状态来提交预提交的数据，也可删除预提交的数据。

9. 如果下级存储不支持事务，Flink 怎么保证 exactly-once

端到端的 exactly-once 对 sink 要求比较高，具体实现主要有幂等写入和事务性写入两种方式。

幂等写入的场景依赖于业务逻辑，更常见的是用事务性写入。而事务性写入又有预写日志（WAL）和两阶段提交（2PC）两种方式。

如果外部系统不支持事务，那么可以用预写日志的方式，把结果数据先当成状态保存，然后在收到 checkpoint 完成的通知时，一次性写入 sink 系统。

10. Flink 是如何处理反压的

Flink 内部是基于 producer-consumer 模型来进行消息传递的，Flink 的反压设计也是基于这个模型。Flink 使用了高效有界的分布式阻塞队列，就像 Java 通用的阻塞队列（BlockingQueue）一样。下游消费者消费变慢，上游就会受到阻塞。

11. Flink 中的状态存储

Flink 在做计算的过程中经常需要存储中间状态，来避免数据丢失和状态恢复。选择的状态存储策略不同，会影响状态持久化如何和 checkpoint 交互。Flink 提供了三种状态存储方式：**MemoryStateBackend、FsStateBackend、RocksDBStateBackend。**

12. Flink 是如何支持批流一体的

这道题问的比较开阔，如果知道 Flink 底层原理，可以详细说说，如果不是很了解，就直接简单一句话：**Flink 的开发者认为批处理是流处理的一种特殊情况。批处理是有限的流处理。Flink 使用一个引擎支持了 DataSet API 和 DataStream API。**

13. Flink 的内存管理是如何做的

Flink 并不是将大量对象存在堆上，而是将对象都序列化到一个预分配的内存块上。此外，Flink 大量的使用了堆外内存。如果需要处理的数据超出了内存限制，则会将部分数据存储到硬盘上。Flink 为了直接操作二进制数据实现了自己的序列化框架。

14. Flink CEP 编程中当状态没有到达的时候会将数据保存在哪里

在流式处理中，CEP 当然是要支持 EventTime 的，那么相对应的也要支持数据的迟到现象，也就是 watermark 的处理逻辑。CEP 对未匹配成功的事件序列的处理，和迟到数据是类似的。在 Flink CEP 的处理逻辑中，状态没有满足的和迟到的数据，都会存储在一个 Map 数据结构中，也就是说，如果我们限定判断事件序列的时长为 5 分钟，那么内存中就会存储 5 分钟的数据，这在我看来，也是对内存的极大损伤之一。

15.讲一下 flink 的运行架构

当 Flink 集群启动后，首先会启动一个 JobManger 和一个或多个的 TaskManager。由 Client 提交任务给 JobManager，JobManager 再调度任务到各个 TaskManager 去执行，然后 TaskManager 将心跳和统计信息汇报给 JobManager。TaskManager 之间以流的形式进行数据的传输。上述三者均为独立的 JVM 进程。

- **Client** 为提交 Job 的客户端，可以是运行在任何机器上（与 JobManager 环境连通即可）。提交 Job 后，Client 可以结束进程（Streaming 的任务），也可以不结束并等待结果返回。

- **JobManager** 主要负责调度 Job 并协调 Task 做 checkpoint，职责上很像 Storm 的 Nimbus。从 Client 处接收到 Job 和 JAR 包等资源后，会生成优化后的执行计划，并以 Task 的单元调度到各个 TaskManager 去执行。
- **TaskManager** 在启动的时候就设置好了槽位数（Slot），每个 slot 能启动一个 Task，Task 为线程。从 JobManager 处接收需要部署的 Task，部署启动后，与自己的上游建立 Netty 连接，接收数据并处理。

16.讲一下 flink 的作业执行流程

以 yarn 模式 Per-job 方式为例概述作业提交执行流程

当执行 executor() 之后,会首先在本地 client 中将代码转化为可以提交的 JobGraph

1. 如果提交为 Per-Job 模式,则首先需要启动 AM, client 会首先向资源系统申请资源, 在 yarn 下即为申请 container 开启 AM, 如果是 Session 模式的话则不需要这个步骤
2. Yarn 分配资源, 开启 AM
3. Client 将 Job 提交给 Dispatcher
4. Dispatcher 会开启一个新的 JobManager 线程
5. JM 向 Flink 自己的 Resourcemanager 申请 slot 资源来执行任务
6. RM 向 Yarn 申请资源来启动 TaskManger (Session 模式跳过此步)
7. Yarn 分配 Container 来启动 taskManger (Session 模式跳过此步)

8. Flink 的 RM 向 TM 申请 slot 资源来启动 task
9. TM 将待分配的 slot 提供给 JM
10. JM 提交 task, TM 会启动新的线程来执行任务,开始启动后就可以通过 shuffle 模块进行 task 之间的数据交换

17．flink 中的时间概念，eventTime 和 processTime 的区别

Flink 中有三种时间概念,分别是 Processing Time、Event Time 和 Ingestion Time

- **Processing Time**

Processing Time 是指事件被处理时机器的系统时间。

当流程序在 Processing Time 上运行时，所有基于时间的操作(如时间窗口)将使用当时机器的系统时间。每小时 Processing Time 窗口将包括在系统时钟指示整个小时之间到达特定操作的所有事件

- **Event Time**

Event Time 是事件发生的时间，一般就是数据本身携带的时间。这个时间通常是在事件到达 Flink 之前就确定的，并且可以从每个事件中获取到事件时间戳。在 Event Time 中，时间取决于数据，而跟其他没什么关系。Event Time 程序必须指定如何生成 Event Time 水印，这是表示 Event Time 进度的机制

- **Ingestion Time**

Ingestion Time 是事件进入 Flink 的时间。 在源操作处，每个事件将源的当前时间作为时间戳，并且基于时间的操作（如时间窗口）会利用这个时间戳

Ingestion Time 在概念上位于 Event Time 和 Processing Time 之间。 与 Processing Time 相比，它稍微贵一些，但结果更可预测。因为 Ingestion Time 使用稳定的时间戳（在源处分配一次），所以对事件的不同窗口操作将引用相同的时间戳，而在 Processing Time 中，每个窗口操作符可以将事件分配给不同的窗口（基于机器系统时间和到达延迟）

与 Event Time 相比，Ingestion Time 程序无法处理任何无序事件或延迟数据，但程序不必指定如何生成水印

Spark

1. spark 如何保证宕机迅速恢复?

适当增加 spark standby master

编写 shell 脚本，定期检测 master 状态，出现宕机后对 master 进行重启操作

2. Spark streaming 以及基本工作原理？

Spark streaming 是 spark core API 的一种扩展，可以用于进行大规模、高吞吐量、容错的实时数据流的处理。

它支持从多种数据源读取数据，比如 Kafka、Flume、Twitter 和 TCP Socket，并且能够使用算子比如 map、reduce、join 和 window 等来处理数据，处理后的数据可以保存到文件系统、数据库等存储中。

Spark streaming 内部的基本工作原理是：接受实时输入数据流，然后将数据拆分成 batch，比如每收集一秒的数据封装成一个 batch，然后将每个 batch 交给 spark 的计算引擎进行处理，最后会生产处一个结果数据流，其中的数据也是一个一个的 batch 组成的。

3. spark 有哪些组件？

master：管理集群和节点，不参与计算。worker：计算节点，进程本身不参与计算，和 master 汇报。Driver：运行程序的 main 方法，创建 spark context 对象。spark context：控制整个 application 的生命周期，包括 dagsheduler 和 task scheduler 等组件。client：用户提交程序的入口。

4. spark 工作机制？

用户在 client 端提交作业后，会由 Driver 运行 main 方法并创建 spark context 上下文。执行 add 算子，形成 dag 图输入 dagscheduler，按照 add 之间的依赖关系划分

stage 输入 task scheduler。task scheduler 会将 stage 划分为 task set 分发到各个节点的 executor 中执行。

5. Spark 主备切换机制原理知道吗？

Master 实际上可以配置两个，Spark 原生的 standalone 模式是支持 Master 主备切换的。当 Active Master 节点挂掉以后，我们可以将 Standby Master 切换为 Active Master。

Spark Master 主备切换可以基于两种机制，一种是基于文件系统的，一种是基于 ZooKeeper 的。

基于文件系统的主备切换机制，需要在 Active Master 挂掉之后手动切换到 Standby Master 上；

而基于 Zookeeper 的主备切换机制，可以实现自动切换 Master。

6. spark 的有几种部署模式，每种模式特点？

1）本地模式

Spark 不一定非要跑在 hadoop 集群，可以在本地，起多个线程的方式来指定。将 Spark 应用以多线程的方式直接运行在本地，一般都是为了方便调试，本地模式分三类

local：只启动一个 executor

local[k]:启动 k 个 executor

local[*]：启动跟 cpu 数目相同的 executor

2）standalone 模式

分布式部署集群，自带完整的服务，资源管理和任务监控是 Spark 自己监控，这个模式也是其他模式的基础。

3）Spark on yarn 模式

分布式部署集群，资源和任务监控交给 yarn 管理，但是目前仅支持粗粒度资源分配方式，包含 cluster 和 client 运行模式，cluster 适合生产，driver 运行在集群子节点，具有容错功能，client 适合调试，dirver 运行在客户端。

4）Spark On Mesos 模式。

官方推荐这种模式（当然，原因之一是血缘关系）。正是由于 Spark 开发之初就考虑到支持 Mesos，因此，目前而言，Spark 运行在 Mesos 上会比运行在 YARN 上更加灵活，更加自然。用户可选择两种调度模式之一运行自己的应用程序：

（1）粗粒度模式（Coarse-grained Mode）：每个应用程序的运行环境由一个 Dirver 和若干个 Executor 组成，其中，每个 Executor 占用若干资源，内部可运行多个 Task（对应多少个“slot”）。应用程序的各个任务正式运行之前，需要将运行环境中的资源全部申请好，且运行过程中要一直占用这些资源，即使不用，最后程序运行结束后，回收这些资源。

（2）细粒度模式（Fine-grained Mode）：鉴于粗粒度模式会造成大量资源浪费，Spark On Mesos 还提供了另外一种调度模式：细粒度模式，这种模式类似于现在的云计算，思想是按需分配。

7. Spark 为什么比 mapreduce 快？

1）基于内存计算，减少低效的磁盘交互；

2）高效的调度算法，基于 DAG；

3）容错机制 Linage，精华部分就是 DAG 和 Lingae

8. 简单说一下 hadoop 和 spark 的 shuffle 相同和差异？

1）从 high-level 的角度来看，两者并没有大的差别。 都是将 mapper（Spark 里是 ShuffleMapTask）的输出进行 partition，不同的 partition 送到不同的 reducer（Spark 里 reducer 可能是下一个 stage 里的 ShuffleMapTask，也可能是 ResultTask）。Reducer 以内存作缓冲区，边 shuffle 边 aggregate 数据，等到数据 aggregate 好以后进行 reduce()（Spark 里可能是后续的一系列操作）。

2）从 low-level 的角度来看，两者差别不小。 Hadoop MapReduce 是 sort-based，进入 combine() 和 reduce() 的 records 必须先 sort。这样的好处在于 combine/reduce() 可以处理大规模的数据，因为其输入数据可以通过外排得到（mapper 对每段数据先做排序，reducer 的 shuffle 对排好序的每段数据做归并）。目前的 Spark 默认选择的是 hash-based，通常使用 HashMap 来对 shuffle 来的数据进行 aggregate，不会对数据进行提前排序。

如果用户需要经过排序的数据，那么需要自己调用类似 sortByKey() 的操作；如果你是 Spark 1.1 的用户，可以将 spark.shuffle.manager 设置为 sort，则会对数据进行排序。在 Spark 1.2 中，sort 将作为默认的 Shuffle 实现。

3）从实现角度来看，两者也有不少差别。 Hadoop MapReduce 将处理流程划分出明显的几个阶段：map(), spill, merge, shuffle, sort, reduce() 等。每个阶段各司其职，可以按照过程式的编程思想来逐一实现每个阶段的功能。在 Spark 中，没有这样功能明确的阶段，只有不同的 stage 和一系列的 transformation()，所以 spill, merge, aggregate 等操作需要蕴含在 transformation() 中。

如果我们将 map 端划分数据、持久化数据的过程称为 shuffle write，而将 reducer 读入数据、aggregate 数据的过程称为 shuffle read。那么在 Spark 中，问题就变为怎么在 job 的逻辑或者物理执行图中加入 shuffle write 和 shuffle read 的处理逻辑？以及两个处理逻辑应该怎么高效实现？

Shuffle write 由于不要求数据有序，shuffle write 的任务很简单：将数据 partition 好，并持久化。之所以要持久化，一方面是要减少内存存储空间压力，另一方面也是为了 fault-tolerance。

9. spark 工作机制

① 构建 Application 的运行环境，Driver 创建一个 SparkContext

② SparkContext 向资源管理器（Standalone、Mesos、Yarn）申请 Executor 资源，资源管理器启动 StandaloneExecutorbackend（Executor）③ Executor 向 SparkContext 申请 Task ④ SparkContext 将应用程序分发给 Executor ⑤ SparkContext 就建成 DAG 图，DAGScheduler 将 DAG 图解析成 Stage，每个 Stage 有多个 task，形成 taskset 发送给 task Scheduler，由 task Scheduler 将 Task 发送给 Executor 运行 ⑥ Task 在 Executor 上运行，运行完释放所有资源

10. 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

1）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

2）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

3）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

11.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

12.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。

13.RDD 有哪些缺陷？

1）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。

2）不支持增量迭代计算，Flink 支持

14. 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

15. 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。

16. 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。

17.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。

18.描述 Yarn 执行一个任务的过程？

1）客户端 client 向 ResouceManager 提交 Application，ResouceManager 接受 Application 并根据集群资源状况选取一个 node 来启动 Application 的任务调度器 driver（ApplicationMaster）。

2）ResouceManager 找到那个 node，命令其该 node 上的 nodeManager 来启动一个新的 JVM 进程运行程序的 driver（ApplicationMaster）部分，driver（ApplicationMaster）启动时会首先向 ResourceManager 注册，说明由自己来负责当前程序的运行。

3）driver（ApplicationMaster）开始下载相关 jar 包等各种资源，基于下载的 jar 等信息决定向 ResourceManager 申请具体的资源内容。

4）ResouceManager 接受到 driver（ApplicationMaster）提出的申请后，会最大化的满足 资源分配请求，并发送资源的元数据信息给 driver（ApplicationMaster）。

5）driver（ApplicationMaster）收到发过来的资源元数据信息后会根据元数据信息发指令给具体机器上的 NodeManager，让其启动具体的 container。

6）NodeManager 收到 driver 发来的指令，启动 container，container 启动后必须向 driver（ApplicationMaster）注册。

7）driver（ApplicationMaster）收到 container 的注册，开始进行任务的调度和计算，直到 任务完成。

注意：如果 ResourceManager 第一次没有能够满足 driver（ApplicationMaster）的资源请求 ，后续发现有空闲的资源，会主动向 driver（ApplicationMaster）发送可用资源的元数据信息以提供更多的资源用于当前程序的运行。

19. Spark on Yarn 模式有哪些优点？

1）与其他计算框架共享集群资源（Spark 框架与 MapReduce 框架同时运行，如果不用 Yarn 进行资源分配，MapReduce 分到的内存资源会很少，效率低下）；资源按需分配，进而提高集群资源利用等。

2）相较于 Spark 自带的 Standalone 模式，Yarn 的资源分配更加细致。

3）Application 部署简化，例如 Spark，Storm 等多种框架的应用由客户端提交后，由 Yarn 负责资源的管理和调度，利用 Container 作为资源隔离的单位，以它为单位去使用内存,cpu 等。

4）Yarn 通过队列的方式，管理同时运行在 Yarn 集群中的多个服务，可根据不同类型的应用程序负载情况，调整对应的资源使用量，实现资源弹性管理。

20.谈谈你对 container 的理解？

1）Container 作为资源分配和调度的基本单位，其中封装了的资源如内存，CPU，磁盘，网络带宽等。 目前 yarn 仅仅封装内存和 CPU

2）Container 由 ApplicationMaster 向 ResourceManager 申请的，由 ResouceManager 中的资源调度器异步分配给 ApplicationMaster

3）Container 的运行是由 ApplicationMaster 向资源所在的 NodeManager 发起的，Container 运行时需提供内部执行的任务命令

21.Spark 使用 parquet 文件存储格式能带来哪些好处？

1）如果说 HDFS 是大数据时代分布式文件系统首选标准，那么 parquet 则是整个大数据时代文件存储格式实时首选标准。

2）速度更快：从使用 spark sql 操作普通文件 CSV 和 parquet 文件速度对比上看，绝大多数情况会比使用 csv 等普通文件速度提升 10 倍左右，在一些普通文件系统无法在 spark 上成功运行的情况下，使用 parquet 很多时候可以成功运行。

3）parquet 的压缩技术非常稳定出色，在 spark sql 中对压缩技术的处理可能无法正常的完成工作（例如会导致 lost task，lost executor）但是此时如果使用 parquet 就可以正常的完成。

4）极大的减少磁盘 I/o,通常情况下能够减少 75%的存储空间，由此可以极大的减少 spark sql 处理数据的时候的数据输入内容，尤其是在 spark1.6x 中有个下推过滤器在一些情况下可以极大的减少磁盘的 IO 和内存的占用，（下推过滤器）。

5）spark 1.6x parquet 方式极大的提升了扫描的吞吐量，极大提高了数据的查找速度 spark1.6 和 spark1.5x 相比而言，提升了大约 1 倍的速度，在 spark1.6X 中，操作 parquet 时候 cpu 也进行了极大的优化，有效的降低了 cpu 消耗。

6）采用 parquet 可以极大的优化 spark 的调度和执行。我们测试 spark 如果用 parquet 可以有效的减少 stage 的执行消耗，同时可以优化执行路径。

22.介绍 parition 和 block 有什么关联关系？

1）hdfs 中的 block 是分布式存储的最小单元，等分，可设置冗余，这样设计有一部分磁盘空间的浪费，但是整齐的 block 大小，便于快速找到、读取对应的内容；

2）Spark 中的 partion 是弹性分布式数据集 RDD 的最小单元，RDD 是由分布在各个节点上的 partion 组成的。partion 是指的 spark 在计算过程中，生成的数据在计算空间内最小单元，同一份数据（RDD）的 partion 大小不一，数量不定，是根据 application 里的算子和最初读入的数据分块数量决定；

3）block 位于存储空间、partion 位于计算空间，block 的大小是固定的、partion 大小是不固定的，是从 2 个不同的角度去看数据。

23. Spark 应用程序的执行过程是什么？

1）构建 Spark Application 的运行环境（启动 SparkContext），SparkContext 向资源管理器（可以是 Standalone、Mesos 或 YARN）注册并申请运行 Executor 资源；

2）资源管理器分配 Executor 资源并启动 StandaloneExecutorBackend，Executor 运行情况将随着心跳发送到资源管理器上；

3）SparkContext 构建成 DAG 图，将 DAG 图分解成 Stage，并把 Taskset 发送给 Task Scheduler。Executor 向 SparkContext 申请 Task，Task Scheduler 将 Task 发放给 Executor 运行同时 SparkContext 将应用程序代码发放给 Executor；

4）Task 在 Executor 上运行，运行完毕释放所有资源。

24. 不需要排序的 hash shuffle 是否一定比需要排序的 sort shuffle 速度快？

不一定，当数据规模小，Hash shuffle 快于 Sorted Shuffle 数据规模大的时候；当数据量大，sorted Shuffle 会比 Hash shuffle 快很多，因为数量大的有很多小文件，不均匀，甚至出现数据倾斜，消耗内存大，1.x 之前 spark 使用 hash，适合处理中小规模，1.x 之后，增加了 Sorted shuffle，Spark 更能胜任大规模处理了。

25. Sort-based shuffle 的缺陷?

1）如果 mapper 中 task 的数量过大，依旧会产生很多小文件，此时在 shuffle 传递数据的过程中 reducer 段，reduce 会需要同时大量的记录进行反序列化，导致大量的内存消耗和 GC 的巨大负担，造成系统缓慢甚至崩溃。

2）如果需要在分片内也进行排序，此时需要进行 mapper 段和 reducer 段的两次排序。

26. spark.storage.memoryFraction 参数的含义,实际生产中如何调优？

1）用于设置 RDD 持久化数据在 Executor 内存中能占的比例，默认是 0.6,，默认 Executor 60%的内存，可以用来保存持久化的 RDD 数据。根据你选择的不同的持久化策略，如果内存不够时，可能数据就不会持久化，或者数据会写入磁盘；

2）如果持久化操作比较多，可以提高 spark.storage.memoryFraction 参数，使得更多的持久化数据保存在内存中，提高数据的读取性能，如果 shuffle 的操作比较多，有很多的数据读写操作到 JVM 中，那么应该调小一点，节约出更多的内存给 JVM，避免过多的 JVM gc 发生。在 web ui 中观察如果发现 gc 时间很长，可以设置 spark.storage.memoryFraction 更小一点。

Hive

1. Hive 表关联查询，如何解决数据倾斜的问题？

1）倾斜原因：map 输出数据按 key Hash 的分配到 reduce 中，由于 key 分布不均匀、业务数据本身的特、建表时考虑不周、等原因造成的 reduce 上的数据量差异过大。

（1）key 分布不均匀;

（2）业务数据本身的特性;

（3）建表时考虑不周;

（4）某些 SQL 语句本身就有数据倾斜;

如何避免：对于 key 为空产生的数据倾斜，可以对其赋予一个随机值。

2）解决方案

（1）参数调节：

hive.map.aggr = true

hive.groupby.skewindata=true

有数据倾斜的时候进行负载均衡，当选项设定位 true,生成的查询计划会有两个 MR Job。第一个 MR Job 中，Map 的输出结果集合会随机分布到 Reduce 中，每个 Reduce 做部分聚合操作，并输出结果，这样处理的结果是相同的 Group By Key 有可能被分发到不同的 Reduce 中，从而达到负载均衡的目的；第二个 MR Job 再根据预处理的数据结果按照 Group By Key 分布到 Reduce 中（这个过程可以保证相同的 Group By Key 被分布到同一个 Reduce 中），最后完成最终的聚合操作。

（2）SQL 语句调节：

① 选用 join key 分布最均匀的表作为驱动表。做好列裁剪和 filter 操作，以达到两表做 join 的时候，数据量相对变小的效果。

② 大小表 Join：

使用 map join 让小的维度表（1000 条以下的记录条数）先进内存。在 map 端完成 reduce。

③ 大表 Join 大表：

把空值的 key 变成一个字符串加上随机数，把倾斜的数据分到不同的 reduce 上，由于 null 值关联不上，处理后并不影响最终结果。

④ count distinct 大量相同特殊值:

count distinct 时，将值为空的情况单独处理，如果是计算 count distinct，可以不用处理，直接过滤，在最后结果中加 1。如果还有其他计算，需要进行 group by，可以先将值为空的记录单独处理，再和其他计算结果进行 union。

2. Hive 的 HSQL 转换为 MapReduce 的过程？

HiveSQL ->AST(抽象语法树) -> QB(查询块) ->OperatorTree（操作树）->优化后的操作树->mapreduce 任务树->优化后的 mapreduce 任务树

过程描述如下：

SQL Parser：Antlr 定义 SQL 的语法规则，完成 SQL 词法，语法解析，将 SQL 转化为抽象语法树 AST Tree；

Semantic Analyzer：遍历 AST Tree，抽象出查询的基本组成单元 QueryBlock；

Logical plan：遍历 QueryBlock，翻译为执行操作树 OperatorTree；

Logical plan optimizer: 逻辑层优化器进行 OperatorTree 变换，合并不必要的 ReduceSinkOperator，减少 shuffle 数据量；

Physical plan：遍历 OperatorTree，翻译为 MapReduce 任务；

Logical plan optimizer：物理层优化器进行 MapReduce 任务的变换，生成最终的执行计划。

3. Hive 底层与数据库交互原理？

由于 Hive 的元数据可能要面临不断地更新、修改和读取操作，所以它显然不适合使用 Hadoop 文件系统进行存储。目前 Hive 将元数据存储在 RDBMS 中，比如存储在 MySQL、Derby 中。元数据信息包括：存在的表、表的列、权限和更多的其他信息。

4. Hive 的两张表关联，使用 MapReduce 怎么实现？

如果其中有一张表为小表，直接使用 map 端 join 的方式（map 端加载小表）进行聚合。

如果两张都是大表，那么采用联合 key，联合 key 的第一个组成部分是 join on 中的公共字段，第二部分是一个 flag，0 代表表 A，1 代表表 B，由此让 Reduce 区分客户信息和订单信息；在 Mapper 中同时处理两张表的信息，将 join on 公共字段相同的数据划分到同一个分区中，进而传递到一个 Reduce 中，然后在 Reduce 中实现聚合。

5. 请谈一下 Hive 的特点？

hive 是基于 Hadoop 的一个数据仓库工具，可以将结构化的数据文件映射为一张数据库表，并提供完整的 sql 查询功能，可以将 sql 语句转换为 MapReduce 任务进行运行。其优点是学习成本低，可以通过类 SQL 语句快速实现简单的 MapReduce 统计，不必开发专门的 MapReduce 应用，十分适合数据仓库的统计分析，但是 Hive 不支持实时查询。

6. 请说明 hive 中 Sort By，Order By，Cluster By，Distrbute 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。

7. 写出 hive 中 split、coalesce 及 collect_list 函数的用法（可举例）？

split 将字符串转化为数组，即：split('a,b,c,d' , ',') ==> ["a","b","c","d"]。

coalesce(T v1, T v2, …) 返回参数中的第一个非空值；如果所有值都为 NULL，那么返回 NULL。

collect_list 列出该字段所有的值，不去重 => select collect_list(id) from table。

8. Hive 有哪些方式保存元数据，各有哪些特点？

Hive 支持三种不同的元存储服务器，分别为：内嵌式元存储服务器、本地元存储服务器、远程元存储服务器，每种存储方式使用不同的配置参数。

内嵌式元存储主要用于单元测试，在该模式下每次只有一个进程可以连接到元存储，Derby 是内嵌式元存储的默认数据库。

在本地模式下，每个 Hive 客户端都会打开到数据存储的连接并在该连接上请求 SQL 查询。

在远程模式下，所有的 Hive 客户端都将打开一个到元数据服务器的连接，该服务器依次查询元数据，元数据服务器和客户端之间使用 Thrift 协议通信。

9. Hive 内部表和外部表的区别？

创建表时：创建内部表时，会将数据移动到数据仓库指向的路径；若创建外部表，仅记录数据所在的路径，不对数据的位置做任何改变。

删除表时：在删除表的时候，内部表的元数据和数据会被一起删除， 而外部表只删除元数据，不删除数据。这样外部表相对来说更加安全些，数据组织也更加灵活，方便共享源数据。

10. Hive 中的压缩格式 TextFile、SequenceFile、RCfile 、ORCfile 各有什么区别？

1、TextFile

默认格式，**存储方式为行存储，数据不做压缩，磁盘开销大，数据解析开销大。**可结合 Gzip、Bzip2 使用(系统自动检查，执行查询时自动解压)，但使用这种方式，压缩后的文件不支持 split，Hive 不会对数据进行切分，从而无法对数据进行并行操作。并且在反序列化过程中，必须逐个字符判断是不是分隔符和行结束符，因此反序列化开销会比 SequenceFile 高几十倍。

2、SequenceFile

SequenceFile 是 Hadoop API 提供的一种二进制文件支持，**存储方式为行存储，其具有使用方便、可分割、可压缩的特点。**

SequenceFile 支持三种压缩选择：NONE，RECORD，BLOCK。Record 压缩率低，**一般建议使用 BLOCK 压缩。**

优势是文件和 hadoop api 中的 MapFile 是相互兼容的

3、RCFile

存储方式：**数据按行分块，每块按列存储。**结合了行存储和列存储的优点：

首先，RCFile 保证同一行的数据位于同一节点，因此元组重构的开销很低；

其次，像列存储一样，RCFile 能够利用列维度的数据压缩，并且能跳过不必要的列读取；

4、ORCFile

存储方式：数据按行分块 每块按照列存储。

压缩快、快速列存取。

效率比 rcfile 高，是 rcfile 的改良版本。

总结：**相比 TEXTFILE 和 SEQUENCEFILE，RCFILE 由于列式存储方式，数据加载时性能消耗较大，但是具有较好的压缩比和查询响应。**

数据仓库的特点是一次写入、多次读取，因此，整体来看，RCFILE 相比其余两种格式具有较明显的优势。

11.所有的 Hive 任务都会有 MapReduce 的执行吗？

不是，从 Hive0.10.0 版本开始，对于简单的不需要聚合的类似 SELECT from LIMIT n 语句，不需要起 MapReduce job，直接通过 Fetch task 获取数据。

12.Hive 的函数：UDF、UDAF、UDTF 的区别？

UDF：单行进入，单行输出

UDAF：多行进入，单行输出

UDTF：单行输入，多行输出

13.说说对 Hive 桶表的理解？

桶表是对数据进行哈希取值，然后放到不同文件中存储。

数据加载到桶表时，会对字段取 hash 值，然后与桶的数量取模。把数据放到对应的文件中。物理上，每个桶就是表(或分区）目录里的一个文件，一个作业产生的桶(输出文件)和 reduce 任务个数相同。

桶表专门用于抽样查询，是很专业性的，不是日常用来存储数据的表，需要抽样查询时，才创建和使用桶表。

数据查询面试题

HBase

1. HBase 的特点是什么？

1）大：一个表可以有数十亿行，上百万列；

2）无模式：每行都有一个可排序的主键和任意多的列，列可以根据需要动态的增加，同一张表中不同的行可以有截然不同的列；

3）面向列：面向列（族）的存储和权限控制，列（族）独立检索；

4）稀疏：空（null）列并不占用存储空间，表可以设计的非常稀疏；

5）数据多版本：每个单元中的数据可以有多个版本，默认情况下版本号自动分配，是单元格插入时的时间戳；

6）数据类型单一：Hbase 中的数据都是字符串，没有类型。

2. HBase 适用于怎样的情景？

① 半结构化或非结构化数据　对于数据结构字段不够确定或杂乱无章很难按一个概念去进行抽取的数据适合用 HBase。以上面的例子为例，当业务发展需要存储 author 的 email，phone， address 信息时 RDBMS 需要停机维护，而 HBase 支持动态增加。

② 记录非常稀疏

RDBMS 的行有多少列是固定的，为 null 的列浪费了存储空间。而如上文提到的，HBase 为 null 的 Column 不会被存储，这样既节省了空间又提高了读性能。

③ 多版本数据

如上文提到的根据 Row key 和 Column key 定位到的 Value 可以有任意数量的版本值，因此对于需要存储变动历史记录的数据，用 HBase 就非常方便了。比如上例中的 author 的 Address 是会变动的，业务上一般只需要最新的值，但有时可能需要查询到历史值。

④ 超大数据量

当数据量越来越大，RDBMS 数据库撑不住了，就出现了读写分离策略，通过一个 Master 专门负责写操作，多个 Slave 负责读操作，服务器成本倍增。 随着压力增加，Master 撑不住了，这时就要分库了，把关联不大的数据分开部署，一些 join 查询不能用了，需要借助中间层。随着数据量的进一步增加， 一个表的记录越来越大，查询就变得很慢，于是又得搞分表，比如按 ID 取模分成多个表以减少单个表的记录数。经历过这些事的人都知道过程是多么的折腾。 采用 HBase 就简单了，只需要加机器即可，HBase 会自动水平切分扩展，跟 Hadoop 的无缝集成保障了其数据可靠性（HDFS）和海量数据分析的高性能（MapReduce）。

3. 描述 HBase 的 rowKey 的设计原则？

（1）Rowkey 长度原则

Rowkey 是一个二进制码流，Rowkey 的长度被很多开发者建议说设计在 10~100 个字节，不过建议是越短越好，不要超过 16 个字节。

原因如下：

① 数据的持久化文件 HFile 中是按照 KeyValue 存储的，如果 Rowkey 过长比如 100 个字节，1000 万列数据光 Rowkey 就要占用 100*1000 万=10 亿个字节， 将近 1G 数据，这会极大影响 HFile 的存储效率；

② MemStore 将缓存部分数据到内存，如果 Rowkey 字段过长内存的有效利用率会降低，系统将无法缓存更多的数据，这会降低检索效率。 因此 Rowkey 的字节长度越短越好。

③ 目前操作系统是都是 64 位系统，内存 8 字节对齐。控制在 16 个字节，8 字节的整数倍利用操作系统的最佳特性。

（2）Rowkey 散列原则

如果 Rowkey 是按时间戳的方式递增，不要将时间放在二进制码的前面，建议将 Rowkey 的高位作为散列字段，由程序循环生成，低位放时间字段， 这样将提高数据均衡分布在每个 Regionserver 实现负载均衡的几率。如果没有散列字段，首字段直接是时间信息将产生所有新数据都在一个 RegionServer 上堆积的 热点现象，这样在做数据检索的时候负载将会集中在个别 RegionServer，降低查询效率。

（3）Rowkey 唯一原则

必须在设计上保证其唯一性。

4. 描述 HBase 中 scan 和 get 的功能以及实现的异同？

HBase 的查询实现只提供两种方式：

1）按指定 RowKey 获取唯一一条记录，get 方法（org.apache.hadoop.hbase.client.Get） Get 的方法处理分两种：设置了 ClosestRowBefore 和 没有设置 ClosestRowBefore 的 rowlock。主要是用来保证行的事务性，即每个 get 是以一个 row 来标记的。一个 row 中可以有很多 family 和 column。

2）按指定的条件获取一批记录，scan 方法(org.apache.Hadoop.hbase.client.Scan）实现条件查询功能使用的就是 scan 方式。

（1）scan 可以通过 setCaching 与 setBatch 方法提高速度(以空间换时间)；

（2）scan 可以通过 setStartRow 与 setEndRow 来限定范围([start，end)start 是闭区间，end 是开区间)。范围越小，性能越高。

（3）scan 可以通过 setFilter 方法添加过滤器，这也是分页、多条件查询的基础。

5. 请详细描述 HBase 中一个 cell 的结构？

HBase 中通过 row 和 columns 确定的为一个存贮单元称为 cell。

Cell：由{row key, column(= +), version}唯一确定的单元。cell 中的数据是没有类型的，全部是字节码形式存贮。

6. 简述 HBase 中 compact 用途是什么，什么时候触发，分为哪两种，有什么区别，有哪些相关配置参数？

在 hbase 中每当有 memstore 数据 flush 到磁盘之后，就形成一个 storefile，当 storeFile 的数量达到一定程度后，就需要将 storefile 文件来 进行 compaction 操作。

Compact 的作用：

① 合并文件

② 清除过期，多余版本的数据

③ 提高读写数据的效率

HBase 中实现了两种 compaction 的方式：minor and major. 这两种 compaction 方式的区别是：

1）Minor 操作只用来做部分文件的合并操作以及包括 minVersion=0 并且设置 ttl 的过期版本清理，不做任何删除数据、多版本数据的清理工作。

2）Major 操作是对 Region 下的 HStore 下的所有 StoreFile 执行合并操作，最终的结果是整理合并出一个文件。

7. 每天百亿数据存入 HBase，如何保证数据的存储正确和在规定的时间里全部录入完毕，不残留数据？

需求分析：

1）百亿数据：证明数据量非常大；

2）存入 HBase：证明是跟 HBase 的写入数据有关；

3）保证数据的正确：要设计正确的数据结构保证正确性；

4）在规定时间内完成：对存入速度是有要求的。

解决思路：

1）数据量百亿条，什么概念呢？假设一整天 60x60x24 = 86400 秒都在写入数据，那么每秒的写入条数高达 100 万条，HBase 当然是支持不了每秒百万条数据的， 所以这百亿条数据可能不是通过实时地写入，而是批量地导入。批量导入推荐使用 BulkLoad 方式（推荐阅读：Spark 之读写 HBase），性能是普通写入方式几倍以上；

2）存入 HBase：普通写入是用 JavaAPI put 来实现，批量导入推荐使用 BulkLoad；

3）保证数据的正确：这里需要考虑 RowKey 的设计、预建分区和列族设计等问题；

4）在规定时间内完成也就是存入速度不能过慢，并且当然是越快越好，使用 BulkLoad。

8. 请列举几个 HBase 优化方法？

1）减少调整

减少调整这个如何理解呢？HBase 中有几个内容会动态调整，如 region（分区）、HFile，所以通过一些方法来减少这些会带来 I/O 开销的调整。

① Region

如果没有预建分区的话，那么随着 region 中条数的增加，region 会进行分裂，这将增加 I/O 开销，所以解决方法就是根据你的 RowKey 设计来进行预建分区， 减少 region 的动态分裂。

② HFile

HFile 是数据底层存储文件，在每个 memstore 进行刷新时会生成一个 HFile，当 HFile 增加到一定程度时，会将属于一个 region 的 HFile 进行合并，这个步骤会带来开销但不可避免，但是合并后 HFile 大小如果大于设定的值，那么 HFile 会重新分裂。为了减少这样的无谓的 I/O 开销，建议估计项目数据量大小， 给 HFile 设定一个合适的值。

2）减少启停

数据库事务机制就是为了更好地实现批量写入，较少数据库的开启关闭带来的开销，那么 HBase 中也存在频繁开启关闭带来的问题。

① 关闭 Compaction，在闲时进行手动 Compaction。

因为 HBase 中存在 Minor Compaction 和 Major Compaction，也就是对 HFile 进行合并，所谓合并就是 I/O 读写，大量的 HFile 进行肯定会带来 I/O 开销， 甚至是 I/O 风暴，所以为了避免这种不受控制的意外发生，建议关闭自动 Compaction，在闲时进行 compaction。

② 批量数据写入时采用 BulkLoad。

如果通过 HBase-Shell 或者 JavaAPI 的 put 来实现大量数据的写入，那么性能差是肯定并且还可能带来一些意想不到的问题，所以当需要写入大量离线数据时 建议使用 BulkLoad。

3）减少数据量

虽然我们是在进行大数据开发，但是如果可以通过某些方式在保证数据准确性同时减少数据量，何乐而不为呢？

① 开启过滤，提高查询速度

开启 BloomFilter，BloomFilter 是列族级别的过滤，在生成一个 StoreFile 同时会生成一个 MetaBlock，用于查询时过滤数据

② 使用压缩

一般推荐使用 Snappy 和 LZO 压缩

4）合理设计

在一张 HBase 表格中 RowKey 和 ColumnFamily 的设计是非常重要，好的设计能够提高性能和保证数据的准确性

① RowKey 设计：应该具备以下几个属性

散列性：散列性能够保证相同相似的 rowkey 聚合，相异的 rowkey 分散，有利于查询。

简短性：rowkey 作为 key 的一部分存储在 HFile 中，如果为了可读性将 rowKey 设计得过长，那么将会增加存储压力。

唯一性：rowKey 必须具备明显的区别性。

业务性：举例来说：

假如我的查询条件比较多，而且不是针对列的条件，那么 rowKey 的设计就应该支持多条件查询。

如果我的查询要求是最近插入的数据优先，那么 rowKey 则可以采用叫上 Long.Max-时间戳的方式，这样 rowKey 就是递减排列。

② 列族的设计：列族的设计需要看应用场景

优势：HBase 中数据时按列进行存储的，那么查询某一列族的某一列时就不需要全盘扫描，只需要扫描某一列族，减少了读 I/O； 其实多列族设计对减少的作用不是很明显，适用于读多写少的场景

劣势：降低了写的 I/O 性能。原因如下：数据写到 store 以后是先缓存在 memstore 中，同一个 region 中存在多个列族则存在多个 store， 每个 store 都一个 memstore，当其实 memstore 进行 flush 时，属于同一个 region 的 store 中的 memstore 都会进行 flush，增加 I/O 开销。

9. Region 如何预建分区？

预分区的目的主要是在创建表的时候指定分区数，提前规划表有多个分区，以及每个分区的区间范围，这样在存储的时候 rowkey 按照分区的区间存储，可以避免 region 热点问题。

通常有两种方案：

方案 1：shell 方法

create 'tb_splits', {NAME => 'cf',VERSIONS=> 3},{SPLITS =>
['10','20','30']}

方案 2：JAVA 程序控制

① 取样，先随机生成一定数量的 rowkey,将取样数据按升序排序放到一个集合里；

② 根据预分区的 region 个数，对整个集合平均分割，即是相关的 splitKeys；

③ HBaseAdmin.createTable(HTableDescriptor tableDescriptor,byte[][]splitkeys)可以指定预分区的 splitKey， 即是指定 region 间的 rowkey 临界值。

10. HRegionServer 宕机如何处理？

1）ZooKeeper 会监控 HRegionServer 的上下线情况，当 ZK 发现某个 HRegionServer 宕机之后会通知 HMaster 进行失效备援；

2）该 HRegionServer 会停止对外提供服务，就是它所负责的 region 暂时停止对外提供服务；

3）HMaster 会将该 HRegionServer 所负责的 region 转移到其他 HRegionServer 上，并且会对 HRegionServer 上存在 memstore 中还未持久化到磁盘中的数据进行恢复；

4）这个恢复的工作是由 WAL 重播来完成，这个过程如下：

① wal 实际上就是一个文件，存在/hbase/WAL/对应 RegionServer 路径下。

② 宕机发生时，读取该 RegionServer 所对应的路径下的 wal 文件，然后根据不同的 region 切分成不同的临时文件 recover.edits。

③ 当 region 被分配到新的 RegionServer 中，RegionServer 读取 region 时会进行是否存在 recover.edits，如果有则进行恢复。

11. HBase 读写流程？

读：

① HRegionServer 保存着 meta 表以及表数据，要访问表数据，首先 Client 先去访问 zookeeper，从 zookeeper 里面获取 meta 表所在的位置信息， 即找到这个 meta 表在哪个 HRegionServer 上保存着。

② 接着 Client 通过刚才获取到的 HRegionServer 的 IP 来访问 Meta 表所在的 HRegionServer，从而读取到 Meta，进而获取到 Meta 表中存放的元数据。

③ Client 通过元数据中存储的信息，访问对应的 HRegionServer，然后扫描所在 HRegionServer 的 Memstore 和 Storefile 来查询数据。

④ 最后 HRegionServer 把查询到的数据响应给 Client。

写：

① Client 先访问 zookeeper，找到 Meta 表，并获取 Meta 表元数据。

② 确定当前将要写入的数据所对应的 HRegion 和 HRegionServer 服务器。

③ Client 向该 HRegionServer 服务器发起写入数据请求，然后 HRegionServer 收到请求并响应。

④ Client 先把数据写入到 HLog，以防止数据丢失。

⑤ 然后将数据写入到 Memstore。

⑥ 如果 HLog 和 Memstore 均写入成功，则这条数据写入成功。

⑦ 如果 Memstore 达到阈值，会把 Memstore 中的数据 flush 到 Storefile 中。

⑧ 当 Storefile 越来越多，会触发 Compact 合并操作，把过多的 Storefile 合并成一个大的 Storefile。

⑨ 当 Storefile 越来越大，Region 也会越来越大，达到阈值后，会触发 Split 操作，将 Region 一分为二。

12. HBase 内部机制是什么？

Hbase 是一个能适应联机业务的数据库系统

物理存储：hbase 的持久化数据是将数据存储在 HDFS 上。

存储管理：一个表是划分为很多 region 的，这些 region 分布式地存放在很多 regionserver 上 Region 内部还可以划分为 store， store 内部有 memstore 和 storefile。

版本管理：hbase 中的数据更新本质上是不断追加新的版本，通过 compact 操作来做版本间的文件合并 Region 的 split。

集群管理：ZooKeeper + HMaster + HRegionServer。

13. Hbase 中的 memstore 是用来做什么的？

hbase 为了保证随机读取的性能，所以 hfile 里面的 rowkey 是有序的。当客户端的请求在到达 regionserver 之后，为了保证写入 rowkey 的有序性， 所以不能将数据立刻写入到 hfile 中，而是将每个变更操作保存在内存中，也就是 memstore 中。memstore 能够很方便的支持操作的随机插入， 并保证所有的操作在内存中是有序的。当 memstore 达到一定的量之后，会将 memstore 里面的数据 flush 到 hfile 中，这样能充分利用 hadoop 写入大文件的性能优势， 提高写入性能。

由于 memstore 是存放在内存中，如果 regionserver 因为某种原因死了，会导致内存中数据丢失。所有为了保证数据不丢失， hbase 将更新操作在写入 memstore 之前会写入到一个 write ahead log(WAL)中。WAL 文件是追加、顺序写入的，WAL 每个 regionserver 只有一个， 同一个 regionserver 上所有 region 写入同一个的 WAL 文件。这样当某个 regionserver 失败时，可以通过 WAL 文件，将所有的操作顺序重新加载到 memstore 中。

14. HBase 在进行模型设计时重点在什么地方？一张表中定义多少个 Column Family 最合适？为什么？

Column Family 的个数具体看表的数据，一般来说划分标准是根据数据访问频度，如一张表里有些列访问相对频繁，而另一些列访问很少， 这时可以把这张表划分成两个列族，分开存储，提高访问效率。

15.如何提高 HBase 客户端的读写性能？请举例说明

① 开启 bloomfilter 过滤器，开启 bloomfilter 比没开启要快 3、4 倍

② Hbase 对于内存有特别的需求，在硬件允许的情况下配足够多的内存给它

③ 通过修改 hbase-env.sh 中的 export HBASE_HEAPSIZE=3000 #这里默认为 1000m

④ 增大 RPC 数量

通过修改 hbase-site.xml 中的 hbase.regionserver.handler.count 属性，可以适当的放大 RPC 数量，默认值为 10 有点小。

16.HBase 集群安装注意事项？

① HBase 需要 HDFS 的支持，因此安装 HBase 前确保 Hadoop 集群安装完成；

② HBase 需要 ZooKeeper 集群的支持，因此安装 HBase 前确保 ZooKeeper 集群安装完成；

③ 注意 HBase 与 Hadoop 的版本兼容性；

④ 注意 hbase-env.sh 配置文件和 hbase-site.xml 配置文件的正确配置；

⑤ 注意 regionservers 配置文件的修改；

⑥ 注意集群中的各个节点的时间必须同步，否则启动 HBase 集群将会报错。

17.直接将时间戳作为行健，在写入单个 region 时候会发生热点问题，为什么呢？

region 中的 rowkey 是有序存储，若时间比较集中。就会存储到一个 region 中，这样一个 region 的数据变多，其它的 region 数据很少，加载数据就会很慢， 直到 region 分裂，此问题才会得到缓解。

18.请描述如何解决 HBase 中 region 太小和 region 太大带来的冲突？

Region 过大会发生多次 compaction，将数据读一遍并重写一遍到 hdfs 上，占用 io，region 过小会造成多次 split，region 会下线，影响访问服务， 最佳的解决方法是调整 hbase.hregion. max.filesize 为 256m。