|  |  |
| --- | --- |
| 问题 | **答案** |
| Storm有哪些特点 | 低延迟。  高性能。  分布式。  可扩展  容错 |
| Storm的编程模型 | 数据模型：  Tuple：Storm内部中数据传输的基本单元，里面封装了一个List对象，用来保存数据。  Stream：一系列tuple组成了Stream，表示数据的流向  计算模型：  **Topology**：Storm中运行的一个实时应用程序的名称。将 Spout、 Bolt整合起来的拓扑图。定义了 Spout和 Bolt的结合关系、并发数量、配置等等。  **Spout：**接受外部数据源的组件，将外部数据源转化成Storm内部的数据，以Tuple为基本的传输单元下发给Bolt  **Bolt：**接受Spout发送的数据，或上游的bolt的发送的数据。根据业务逻辑进行处理。发送给下一个Bolt或者是存储到某种介质上。介质可以是mongodb或mysql，或者其他。 |
| Storm都有哪些常见的流分组策略 | **Shuffle Group**：随机分发元组到Bolt的任务，保证每个任务获得相等数量的元组  **Field Group**：按字段分组， 比如按userid来分组，具有同样userid的tuple会被分到相同的Bolts， 而不同的userid则会被分配到不同的Bolts。  **Global Group：**全局分组，这个tuple被分配到storm中的一个bolt的其中一个task。再具体一点就是分配给id值最低的那个task。  **All Group：**广播发送， 对于每一个tuple， 所有的Bolts都会收到。 |
| 描述Storm的架构 | Nimbus: 负责在集群里面发送代码，分配工作给机器，并且监控状态。全局只有一个。相当于master的角色。  Supervisor：监听分配给它那台机器的工作，根据需要启动/关闭工作进程Worker。每一个要运行Storm的机器上都要部署一个，并且，按照机器的配置设定上面分配的槽位数。  Zookeeper：Storm重点依赖的外部资源。Nimbus和Supervisor甚至实际运行的Worker都是把心跳保存在Zookeeper上的。Nimbus也是根据Zookeerper上的心跳和任务运行状况，进行调度和任务分配的。两者之间的调度器。  Worker：具体处理组建逻辑的进程，  Excutor：不再与物理进程对应，是处理任务的线程， |
| Storm整体工作流程 | Storm的整体工作流程是：  1.客户端新建Topology，定义spout和bolt的初始并发度（初始executor的并发数），并定义各个组件之间的流分组策略。  2.客户端重写spout的方法，确定spout发送信息的可靠性原则（通过调用acker和fail方法），并保证信息流的发送的畅通（调用nextTuple方法）。  3.客户端重写bolt的方法，确定数据处理的具体原则，和最终的结果存储原则。  4.客户端提交整体Topology到Nimbus，主节点查询Zookeeper中子节点的资源信息，确定规则（随机分配/轮循分配/hash值固定分配等）开始分配任务，并将任务与子节点的对应关系存放在Zookeeper中。  5.子节点Supervisor查询Zookeeper中的信息领取任务，分配具体的Worker以及executors执行具体的tasks。  6.每个子节点运行Topology的子集，且整个Storm集群初始化后不会停止，连续运行，直到受到kill相应的守护进程。 |
| Storm的并发机制 | 在集群运行中,topology主要有四个组成部分。  task(bolt/spout实例),Executor(线程),Workers(JVM虚拟机),Nodes(服务器)  **task**： task的nextTuple和execute方法会被executor线程调用  **executor**：是jvm进程中运行的一个java线程,多个task可以分配给同一个executor来执行。也就是说executor与task是一对多的关系。不过,除非明确指定,Storm会默认给每个executor分配一个task。默认是一对一  **Worker**：指的是node上独立的jvm进程。每个node可以配置运行一个或者多个worker。一个topology会分配到一个或者多个worker上运行。  **Node**：指配置在一个 Storm 集群中的服务器，会执行 topology 的一部分运算。一个 Storm 集群可以包括一个或者多个工作node。 |
| 怎么设置各级别的并发数 | **增加Node**  增加集群的服务器数量。  **增加worker**  可以通过API和修改配置两种方式修改分配给topology的woker数量。  Config config = new Config();  config.setNumWorkers(2);  **增加Executor:**  builder.setSpout(spout\_id,spout,2);  builder.setBolt(bolt\_id,bolt,executor\_num);  **增加Task**  builder.setSpout(...).setNumTasks(2);  builder.setBolt(...).setNumTasks(task\_num); |
| Storm**的通信机制** | （1）对于worker进程来说，为了管理流入和传出的消息，每个worker进程有一个独立的接收线程[一个worker进程运行一个专用的接收线程来负责将外部发送过来的消息移动到对应的executor线程的incoming-queue中](对配置的TCP端口supervisor.slots.ports进行监听);  　　（2）对应Worker接收线程，每个worker存在一个独立的发送线程[transfer-queue的大小由参数topology.transfer.buffer.size来设置。transfer-queue的每个元素实际上代表一个tuple的集合]，它负责从worker的transfer-queue中读取消息，并通过网络发送给其他worker  　　（3）每个executor有自己的incoming-queue[executor的incoming-queue的大小用户可以自定义配置。]和outgoing-queue[executor的outgoing-queue的大小用户可以自定义配置]。Worker接收线程将收到的消息通过task编号传递给对应的executor(一个或多个)的incoming-queues;  　　（4）每个executor有单独的线程分别来处理spout/bolt的业务逻辑，业务逻辑输出的中间数据会存放在outgoing-queue中，当executor的outgoing-queue中的tuple达到一定的阀值，executor的发送线程将批量获取outgoing-queue中的tuple,并发送到transfer-queue中。  　　（5）每个worker进程控制一个或多个executor线程，用户可在代码中进行配置。其实就是我们在代码中设置的并发度个数。 |
| **Spout如何获取数据** | 1、直接链接，Spout作为数据输入的源头，启动线程直接链接对应的数据源，拉取特定条件的数据；  2、通过队列过度，不是直接的方式，通过消息队列来进行过度；  3、外部系统通知，消息系统通知到Spout，然后转换为Tuple进行传输； |
| **Storm如何存储数据** | Storm中设计的组件，没有专门存储数据的，一般情况下，会借助第三方的存储，例如mysql、Nosql 等，Bolt的节点，可以用于存储计算的中间结果或者最终结果。 |
| Storm的架构容错机制 | 1. **集群节点宕机**   Nimbus服务器 单点故障时可以添加报警，但程序银镜加载到内存中运行了。 非Nimbus服务器 故障时，该节点上所有Task任务都会超时，**Nimbus会将这些Task任务重新分配到其他服务器上运行**   1. **进程挂掉**   Worker 挂掉时，Supervisor会重新启动这个进程。如果启动过程中仍然一直失败，并且无法向Nimbus发送心跳，**Nimbus会将该Worker重新分配到其他服务器上** Supervisor **无状态（所有的状态信息都存放在Zookeeper中来管理）** 快速失败（每当遇到任何异常情况，都会自动毁灭） Nimbus **无状态（所有的状态信息都存放在Zookeeper中来管理）** 快速失败（每当遇到任何异常情况，都会自动毁灭） |
| **Storm如何保证消息被最终处理** | Ack机制：  Storm中有个特殊的task名叫acker，他们负责跟踪spout发出的每一个Tuple的Tuple树（因为一个tuple通过spout发出了，经过每一个bolt处理后，会生成一个新的tuple发送出去）。当acker（框架自启动的task）发现一个Tuple树已经处理完成了，它会发送一个消息给产生这个Tuple的那个task（spout）。对任意大的一个Tuple树，storm只需要恒定的20字节就可以进行跟踪。acker对于每个spout-tuple保存一个ack-val的校验值，它的初始值是0，然后每发射一个Tuple或Ack一个Tuple时，这个Tuple的id就要跟这个校验值异或一下，并且把得到的值更新为ack-val的新值。那么假设每个发射出去的Tuple都被ack了，那么最后ack-val的值就一定是0。Acker就根据ack-val是否为0来判断是否完全处理，如果为0则认为已完全处理。 |
| **Spout和Bolt为啥需要实现序列化** | 一个topology程序提交到集群，是先提交到Nimbus的，然后由其进行分发，分发是跨进程的，到了另外一个进程中，是需要反序列化出来这个处理类的。  同时Storm 中的 tuple可以包含任何类型的对象。由于Storm 是一个分布式系统，所以在不同的任务之间传递消息时Storm必须知道怎样序列化、反序列化消息对象。  　Storm 使用 Kryo库对对象进行序列化。Kryo 是一个灵活、快速的序列化库。Storm 默认支持基础类型、string、byte arrays、ArrayList、HashMap、HashSet 以及 Clojure 的集合类型的序列化。如果需要在tuple中使用其他的对象类型，就需要注册一个自定义的序列化器。 |
| Storm常用的命令 |  activate ：激活指定任务   classpath：打印storm的classpath   deactivate ：暂停storm的任务   help ：命令解释及操作提示   jar ：运行你的storm job，topology是类中的主函数   kill：通过任务名称kill一个任务   list：列出正在运行的topolofies和状态   localconfvalue：打印出具体配置参数在本地storm配置文件中的值   nimbus：启动一个nimubs进程   supervisor：启动一个supervisor进程   ui：启动监控页面UI的后台进程   rebalance：节点扩展之后进行负载均衡 |
| Strom性能优化 | 编程优化   1. kryo序列化 2. Bolt批量提交 3. 不要在spout中处理耗时的操作 4. 使用组件的并行度代替线程池 5. 注意fieldsGrouping 的数据均衡性 6. 优先使用localOrShuffleGrouping 7. 合理设置Worker、Spout、Bolt的并行度 8. 确认是否需要开启ACK，以及Acker数量 9. 存储层方面，需要考虑连接数据库的时候是否用了连接池来减少程序连接数据库的资源开销，是否合理使用批处理以减少程序和数据库的通信   配置优化   1. **最大 Spout 挂起时间**：在启用Ack 的情况下，Spout 中有个RotatingMap 用来保存Spout 已经发送出去，但还没有等到Ack 结果的消息。RotatingMap 的最大个数是有限制的，为p\*num-tasks。其中p 是topology.max.spout.pending 值，也就是MaxSpoutPending。如果不设置MaxSpoutPending 的大小或者设置得太大，可能消耗掉过多的内存导致内存溢出，设置太小则会影响Spout 发射Tuple 的速度。 2. **worker内存调优**：设置合适的worker内存大小 3. **GC调优**：建议使用CMS GC 方式， 4. **开启反压机制** |
| **实时计算业务场景举例** | 1、日志分析  例如应用系统产生大量的业务日志，这些例如网关系统的API调用情况日志，这些日志，不太适合马上存入数据库，需要进行加工，日志文件的量又非常大，所以没法直接统计，这时候可以通过Storm来进行分析。  2、大数据实时统计  互联网的数据量是海量的时候，没有办法在数据库层面直接SQL来进行统计，需要对于产生的数据，进行二次加工，然后产出结果，正好把实时变化的数据流到storm中处理一遍。  3、管道传输  例如有数据需要从A系统流道B系统，这时候需要中间处理一下，场景是不是很切和。 |