# 法本信息

1. 集群大小是多少，hdfs的磁盘空间是多大？

10台节点，128G，10T磁盘空间，32core

1. 公司每日的数据量是多少？
2. 数仓是什么样的一个架构？数仓的模型是如何进行设计的？

数仓是四层架构：

ods层

dws层

dwd层

ads层

1. 数仓架构中你主要负责了哪些工作？

离线数据仓库pipeline的设计架构，数据的etl以及入库，

对ods层数据的etl以及进行扁平化处理、数据中特殊字段进行解析、业务库数据进行降维处理。数仓中的数据部分冗余，避免大量的join操作。

1. 数仓使用的是那种压缩方式，文件以什么样的格式进行存储？

文件格式使用的parquet：

列式存储格式，相对于行式存储，列式存储具有(1)可以跳过不符合条件的数据，只读取需要的数据，降低IO数据量(2)压缩编码可以降低磁盘存储空间。由于同一列的数据类型是一样的，可以使用更高效的压缩编码进一步节约存储空间。(3)只需要读取需要的列，支持向量运算，能够获取更好的扫描性能等的优点

压缩方式使用的是snappy压缩：

项目中选择了LZO和snappy两种的压缩格式，经过选择，为了能够达到尽可能快的压缩和解压速度，压缩格式选择了snappy，snappy具有合理的压缩率以及压缩速率和解压速率。一般在mapreduce作业的map输出的数据到reduce时，作为中间的数据的压缩格式来使用

1. spark中算子有哪几种(RDD的算子是lazy执行，只有触发Action算子后才会真正的执行)？

转换算子：Transformation，操作时延迟计算的，也就是从一个RDD转换生成另一个RDD的转换操作并不会马上得到执行，需要等到有Action操作的时候才会真正触发运算。

执行算子：Action，算子会触发Spark提交作业(Job)，并将数据输出到Spark系统。

1. spark的调优。

见整理资料，spark各种调优

1. 如何预防RDD的数据丢失。

缓存有可能丢失，或者存储存储于内存的数据由于内存不足而被删除。通过基于RDD的一系列转换，丢失的数据会被重算，由于RDD的各个Partition是相对独立的，因此只需要计算丢失的部分即可，并不需要重算全部Partition。

但是，多次迭代后数据丢失的重新计算，会影响这个效率。因此，RDD的缓存容错机制保证了即使缓存丢失也能保证快速的恢复，而不是重新计算。

checkpoint保存的目录是在HDFS目录中，保证了存储的可靠性。

|  |
| --- |
| sc.setCheckpointDir("hdfs://master:9000/..")//会在..目录创建一个文件夹  //对象面的rdd设置checkpoint  rdd.checkpoint  rdd.collect  checkpoint和cache一样，是transformation  当遇到action时，checkpoint会启动另一个任务，将数据切割拆分，保存到设置的checkpoint目录中。 |

1. 当使用了checkpoint后，数据被保存到HDFS，此RDD的依赖关系也会丢掉，因为数据已经持久化到硬盘，不需要重新计算。
2. 强烈推荐先将数据持久化到内存中（cache操作），否则直接使用checkpoint会开启一个计算，浪费资源。

详细资料：<https://www.cnblogs.com/wangrd/p/6232826.html>，

1. spark的检查点机制。

详细资料：<https://www.jianshu.com/p/a75d0439c2f9>，

1. 集群的单点故障问题。

集群中采用的时HA模式，避免了出现单点故障问题。

1. Hive分区中的大量小文件是如何进行合并的，如何进行合并，多久合并一次。

<https://www.jianshu.com/p/73b28337bf37>，对分区中存在大量的小文件采用通过sparksql程序进行合并，如果分区中频繁的产生大量的小文件，进行设置上的检查，是否为数据每日入库时参数的配置造成的的大量小文件的产生。

1. 项目中对Hive进行过怎么样的调优？
2. 开启Fetch抓取：在hive中某些情况下的查询可以不必使用MapReduce计算。例如select \* from stu；这种情况下，Hive可以简单读取stu目录下对应存储的文件，然后输出即可。在hive-default.xml文件中，设置参数hive.fetch.task.conversion为more即可，老版本默认为minimal，修改为more后，全局查找、字段查找、limit查找等都不会以mapreduce任务来执行。
3. 开启本地模式：在Hive的输入数据量特别非常小的时候，为查询触发任务所消耗的事件可能会比实际job执行的时间要多的多，因此Hive可以通过本地模式在单台机器上处理所有的任务，对于小数据集，执行时间可以明显被缩短。通过设置hive.exec.mode.local.auto的值为true，来在合适的时间自动开启优化。

|  |
| --- |
| set hive.exec.mode.local.auto=true; 开启本地mr  //设置local mr的最大输入数据量，当输入数据量小如这个值的时候采用local mr的方式，默认值为134217728，即128M  set hive.exec.mode.local.auto.inputbytes.max=50000000;  //设置local mr的最大输入文件个数，当输入文件个数小于这个值时采用local mr的方式，默认为4  set hive.exec.mode.local.auto.input.files.max=10; |

1. 开启JVM重用,Hadoop的默认配置是使用派生JVM来执行map和Reduce任务的,因此如果出现有大量的小文件或task特别多的场景，JVM的启动过程就会造成很大的开销,开启JVM的重用可以使得JVM在同一个job中重新使用多次。

通过在Hadoop的mapred-site.xml文件中设置参数mapreduce.job.jvm.numtasks来开启JVM的重用。

|  |
| --- |
| <property>  <name>mapreduce.job.jvm.numtasks</name>  <value>15</value>  <description>How many tasks to run per jvm. If set to -1, there is  no limit.  </description>  </property> |

1. 并行执行，Hive中会将一个查询操作转换成一个或者多个阶段。而在默认情况下，Hive一次只会执行一个阶段，某些job中的这些任务执行的阶段可能并非完全相互依赖的，可以通过并行执行的，可以通过设置参数hive.exec.parallel值为true开启并行执行，并行执行会增加集群的资源利用率，需要有足够的资源才会有优势。

|  |
| --- |
| set hive.exec.parallel=true //开启任务并行执行  set hive.exec.parallel.thread.number=16;//设置一个sql运行最大的并行度，调整为16,默认为8。 |

1. Spark中的序列化。

java的序列化机制

spark的Kryo序列化机制

set("spark.serializer", "org.apache.spark.serializer.KryoSerializer")

**Spark默认情况下**，Spark内部是使用Java的序列化机制，ObjectOutputStream / ObjectInputStream，对象输入输出流机制，来进行序列化

这种默认序列化机制的好处在于，处理起来比较方便；也不需要我们手动去做什么事情，只是，你在**算子里面使用的外部变量，必须是实现Serializable接口的**，可序列化即可。

但是**缺点**在于，**默认的序列化机制的效率不高，序列化的速度比较慢**；序列化以后的数据，**占用的内存空间相对还是比较大**。

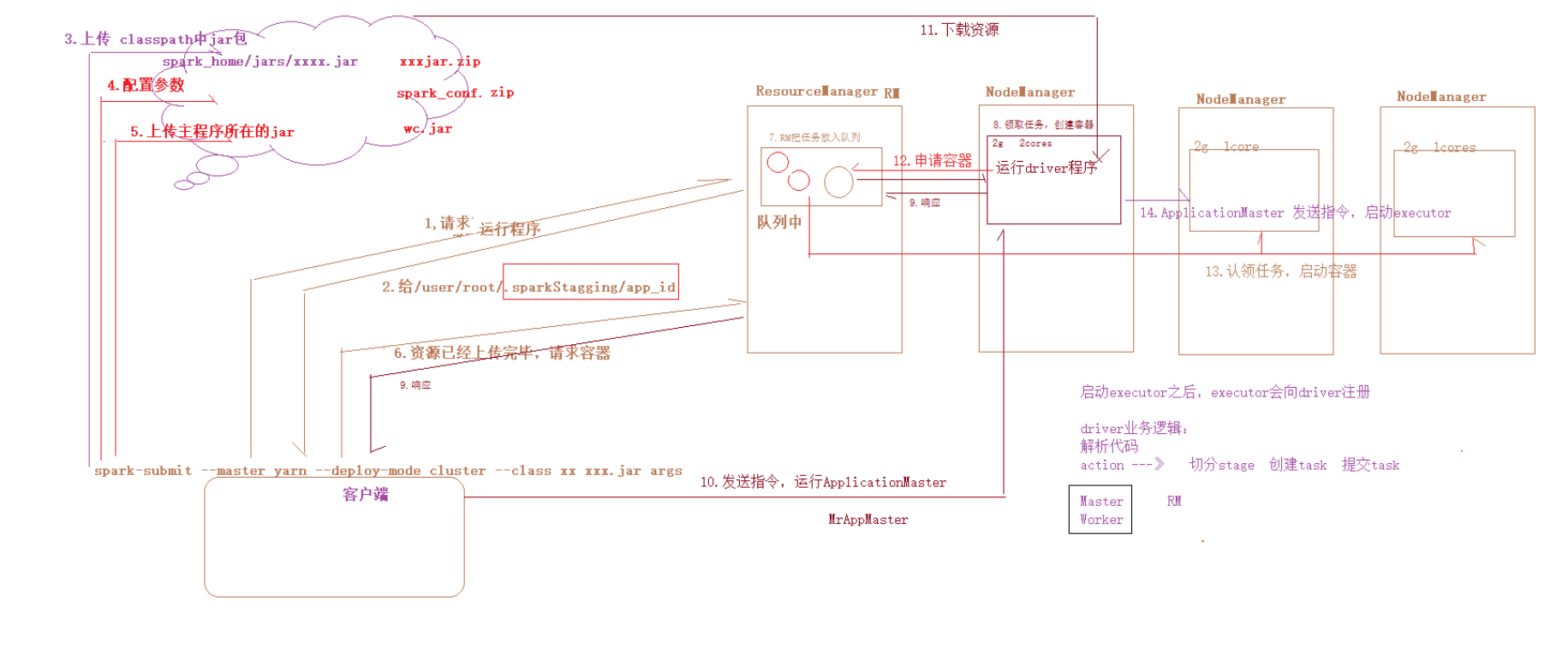
**可以手动进行序列化格式的优化。**

**Spark支持使用Kryo序列化机制**。Kryo序列化机制，比默认的Java序列化机制，速度要快，序列化后的**数据要更小，大概是Java序列化机制的1/10。**

所以Kryo序列化优化以后，可以让**网络传输的数据变少**；在集群中**耗费的内存资源大大减少**。

# 华为中企----电话面试

1. spark的资源管理模式
2. spark的RDD、DateSet、DateFrom的区别和联系
3. Spark on Yarn的运行原理



1. stom、SparkStreaming、Flink的对比
2. Hbase的compact概念
3. Hbase的compact的两种方式
4. Hbase中架构的角色
5. Hbase读数据流程
6. Hbase写数据流程

# RunX跑先体育面试

## 研发

用户画像分析平台项目

数仓项目

其他的消息中间件

HDFS读数据流程

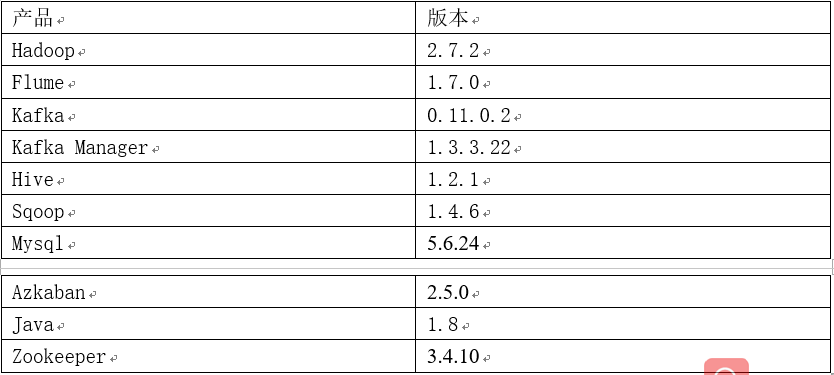
HDFS写数据流程

## 技术

1. Runx实时数据处理现状：

目前的处理方式是以批处理来进行的，自开发的程序来进行定期的去第三方来拉去数据然后进行数据的处理，其实时性要求不高，实时性在3秒。RunX的数据是实时去第三方进行拉取然后进行处理。如何强化提高这种数据处理的方式

1. RunX实时数据进行去做对比，历史数据来做分析，让你来选择，离线和分析这两快，你会选择什么样的技术框架来进行实现？
2. 你对Spark的工作原理的理解！！！(大图)。
3. spark-submit运行一个程序，通过反射调用程序的main方法，执行程序。
4. 首先会实例化sparkContext，实例sparkcontex时会初始化许多的组件，其主要的组件位TaskScheduler、DAGScheduler、SchedulerBackend
   1. TaskSchedule：任务调度器
   2. DAGScheduler：面向DAG的调度器
   3. SchedulerBackend：SchedulerBackend是TaskScheduler的调度后端接口。给Task分配资源后将与分配个Task的Executor通信，并要求Executor来运行Task。
5. SchedulerBackend的子类(StandaloneSchedulerBackend)创建内部类ClientEndpoint来进行与集群管理器进行通信。
6. ClientEndpoint向Master进行注册
   1. ClientEndpoint通过registerMasterThreadPool方法向Master注册Application的ThreadPoolExecutor
7. Master向Worker发送指令启动executor，Worker启动Executor。
8. Executor启动后向Driver进行注册(DriverEndpoint)。
   1. Executor向Driver进行注册主要是与DriverEndopint进行通信。
9. 初始化完成后，driver解析业务逻辑代码
   1. 记录描述信息
   2. 记录数据源
   3. RDD的依赖关系，每个rdd之间传递的具体函数
   4. 构建DAG
10. 但driver解析到Action算子时，DAG构建完成。
11. DAGScheduler开始切分Stage。
12. 提交准备好的Stage
13. 开始执行一个Stage
    1. 根据stage的最后一个分区的数量，创建同等数量的task
    2. 根据stage的类型创建具体的类型(ShuffleMapStage 🡪 shuffleMapTask)
    3. taskScheduler.submitTasks(new TaskSet()) DAGScheduler会把该stage中所有的task，用TaskSet封装，然后通过TaskScheduler来调度Task
14. TaskScheduler的调度
    1. 遍历taskset，将每一个task进行序列化，将序列化后的task发送个executor。
    2. 在进行task调度的时候，会进行优先位置的判断。
15. Executor执行task的业务逻辑
    1. 反序列化每一个task
    2. 使用TaskRunner来封装task，TaskRunner继承了Runnable
    3. 将TaskRunner添加到线程池中，等待执行调度。
    4. 根据Task的类型来进行Task的调度。
16. Hbase在项目中使用的什么版本？



1. 如何规划Hbase的存储。
2. Hbase的优点、缺点。

Hbase是一个基于HDFS做存储、分布式、非关系型(NoSQL)数据库系统！(海量存储能力、hailiang服务能力)

优点:

海量存储能力、海量服务能力：

Hbase为什么有海量存储和服务能力呢？

因为Hbase的数据存储在HDFS中，所以理论上可以存无限多的数据；因其本身就是分布式的，所以有海量服务能力，如果是普通的mysql类数据库，只是单机运行，其服务能力必然受到本机资源性能等的制约，所有core所有memory全占用能提供的服务也是有限的

缺点：

Hbase更新数据操作不易，而且查询较慢

因为Hbase的表实际是存储在hdfs上的一个个文件，hdfs上的文件是无法执行插入修改dengcaozuo 的，数据在更新时会首先写入wal日志和内存(MemStore)中，MemStore中的数据是排序的，是不会进行数据的更新或合并操作的，只增加数据，当MemStore累计到一定阈值时(128M),就会创建一个新的MemSotre，并且将老的MenSotre添加到flush队列，有单独的线程flush到磁盘上，成为一个SotreFile(只读)，这使得用户的写操作只要进入内存就可以立即返回；

因为新插入或更新的操作无法修改源文件，所以这些数据会先cache缓存，然后生成一个新的文件，所以在同一个region列族文件夹下有多个文件，要想查询到一条数据，则要遍历检索该列族下的所有文件。

单一的RowKey固有的局限性决定了Hbase不支持多条件查询

1. 为了让Hbase支持多条件查询，应当对Hbase进行什么样的设计。

方案一：使用Hbase API中提供的各种filter过滤器进行筛选，查询的效率相对会不是特别的理想！

方案二：建立二级索引

1. 使用Hbase协处理器Coprocessor在写入数据时，创建二级索引表，并将每条数据的索引写入二级索引表中，查询时现根据筛选条件查询二级索引表，获取相对应的一级索引rowkey，然后根据rowkey去数据表中查询结果。
2. 使用solr搜索应用服务器，再写入数据时，在solr中建立二级索引(也可以建立全文索引)，查询时先根据筛选条件获取一级索引rowkey的集合，然后根据rowkey去数据表中获取查询结果。

全文检索引擎Solr系列—–全文检索基本原理: <http://www.importnew.com/12707.html>

1. Hbase的RowKey的设计。

项目中对RowKey的设计：row的长度为16位，将ids进行base32编码取前八位，拼接时间。

1. Hbase的最新版本的特性。

2.1.3支持二级索引。

1. Hbase最新版本的搭建中的一些坑。
2. 如何保证Hbase的读写性能。
3. Hbase写入性能优化

是否需要写WAL？WAL是否需要同步写？

WAL机制可以确保即使写入缓存中的数据丢失了，也可以进行恢复，为了集群之间的异步复制。默认时WAL机制开启的，且使用同步机制写入WAL，企业中的业务大多数时开启WAL机制，但是在部分业务中可能对出现异常情况照成数据丢失的情况并不是特别关心，而更关心的数据的写入吞吐量，不能照成数据队列堵塞。这种场景可以选择关闭WAL写入；或者看能否接受异步写入，因此根据业务关注点在WAL机制和写入吞吐量之间进行一个抉择。

PUT同步批量提交

HBase分别提供了单条put以及批量put的API接口，使用批量put接口可以减少客户端到RegionServer之间的RPC连接数，提高写入性能。另外需要注意的是，批量put请求要么全部成功返回，要么抛出异常。

所以建议使用批量PUT写入请求

PUT异步批量提交

业务如果可以接受异常情况下少量数据丢失的话，还可以使用异步批量提交的方式提交请求。提交分为两阶段执行：用户提交写请求之后，数据会写入客户端缓存，并返回用户写入成功；当客户端缓存达到阈值（默认2M）之后批量提交给RegionServer。需要注意的是，在某些情况下客户端异常的情况下缓存数据有可能丢失。

所以在业务可以接受的情况下开启异步批量提交

Region数量少于RegionServer数量

当前集群中表的Region个数如果小于RegionServer个数,可以考虑切分Region并尽可能分布到不同RegionServer来提高系统请求并发度，如果Num(Region of Table) > Num(RegionServer)，再增加Region个数效果并不明显

所以在Num(Region of Table) <Num(RegionServer)的场景下切分部分请求负载高的Region并迁移到其他RegionServer

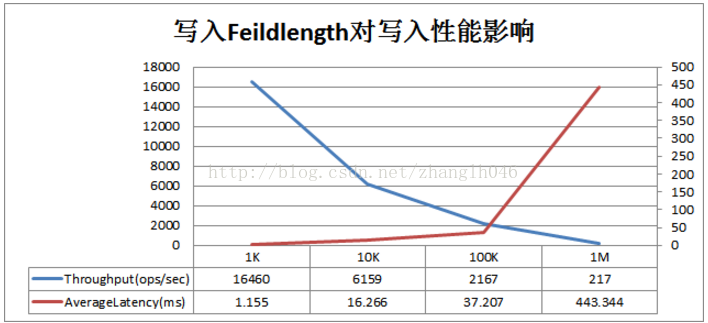
写入请求是否均衡

另一个需要考虑的问题是写入请求是否均衡，如果不均衡，一方面会导致系统并发度较低，另一方面也有可能造成部分节点负载很高，进而影响其他业务。分布式系统中特别害怕一个节点负载很高的情况，一个节点负载很高可能会拖慢整个集群，这是因为很多业务会使用Mutli批量提交读写请求，一旦其中一部分请求落到该节点无法得到及时响应，就会导致整个批量请求超时。

所以建议检查Rowkey 和预分区策略

写入的KeyValue数据是否太大

KeyValue大小对写入性能的影响巨大，一旦遇到写入性能比较差的情况，需要考虑是否由于写入KeyValue数据太大导致。KeyValue大小对写入性能影响曲线图如下：



KeyValue太大会导致HLog文件写入频繁切换、flush以及compaction频繁触发，写入性能急剧下降。

如果是大字段scan导致RegionServer宕机，那么客户端在访问的时候对返回结果大小做限制(scan.setMaxResultSize(2\*1024\*1024))，并且对列数量做限制(scan.setBatch(100))

1. Hbase的读数据性能优化

合理的Scan操作缓存设置：

因为通常来将一次scan会返回大量的数据，因此客户端发起一次sacn请求，实际并不会将所有的数据加载到本地，而是分成多次RPC请求进行加载，这种的设计方式避免了因为大量数据请求可能会因为大量的数据请求而导致网络带宽严重消耗而影响其他业务，另一方面也可能会因数据量过大而导致本地客户端发生OOM。在这种设计体系下用户会首先加载一部分数据到本地，然后在进行遍历处理，如此反复，直至所有数据都加载完成。默认数据加载本地就存放在scan缓存中，默认存放100条数据。

为了避免遇到一些比较大的scan请求，提高缓存的命中率，将hbase.client.scanner.caching的参数设置为1000条。

根据业务需求使用批量的Get请求来批量的获取数据

Hbase分别提供了单条get以及批量get的API接口，使用批量get接口可以减少客户端到HRegionServer之间的RPC连接数，提高读取性能。使用批量get请求要么成功返回请求数据，要么抛出异常。在项目中根据业务需求来使用批量get[table.get(List<Get>)]进行读取请求

保证读请求的均衡

极端情况下所有的读请求都落在一台RegionServer的某几个Region上，那么就会造成该RegionServer资源的严重消耗，读请求不均衡不仅会造成本身业务性能很差，还会严重影响其他业务。

通过观察所有RegionServer的读请求QPS曲线，来确定读写请是否存在不均衡现象，存在的化，则对RowKey进行散列化处理(比如MD5散列),同时在建表时进行预分区处理。

避免HFile文件过多

<https://www.jianshu.com/p/bf9eb0618933>，

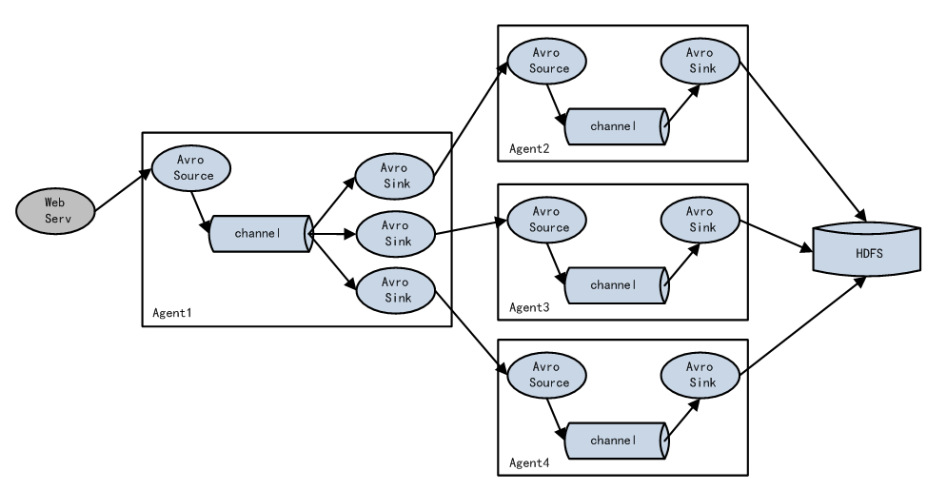
1. ELK技术栈。
2. 对kafka进行手动维护偏移量。

通过对kafka进行手动维护偏移量，实现进消费一次语意，实时处理后的数据会存入到redis中，通过使用redis的pipeline操作，将结果和偏移量一同进行提交，同时成功，或者同时失败，避免重复计算。

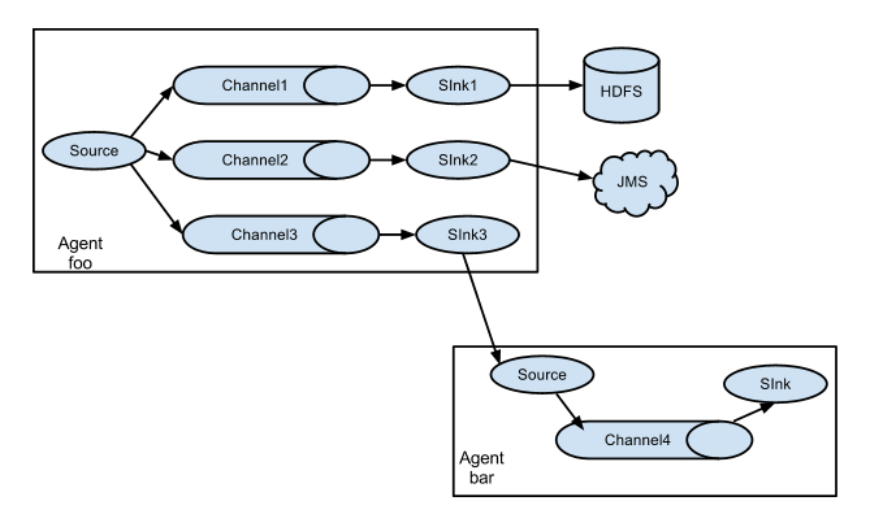
1. 数据的采集：Flume和logstash的grok插件。

Flume：1.6

Flume从原理上来讲，其主要分为三段式的结构：源(Source输入)--存储(Channel管道)--出口(Sink目标输出)



Flume可以支持一个Agent中有多个不同类型的channel和sink，可以讲Source中的数据进行复制，分发给不同目的地的端口：

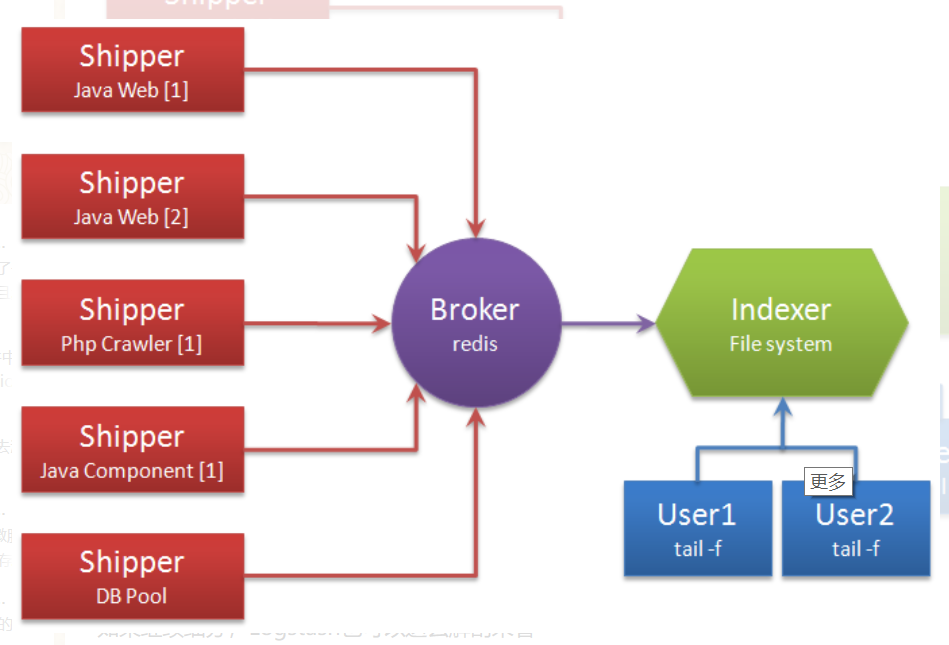


Flume还自带了分区和拦截器功能，Flume的过滤功能是比较弱的。

Flume在集群中最擅长的事情就是路由，每一个Flume Agent相连就构成了一条链路，这也是Flume比较出色的亮点，正也是如此，如果有一个Flume Agent出现了问题，那么整个链路就会出现问题，所以在集群中设计分层架构等来实现冗余备份。

|  |
| --- |
|  |

logstash：主要有三个组件其分工分别为：Shipper负责日志收集。职责时监控本地日志文件的变化，并输出到Redis缓存起来；Broker可以连接多个Shipper和多个Indexer；Indexer负责日志的存储。



Logstash在Filter plugin部分具备比较完备的功能，比如grok，能通过正则解析和结构化任何文本，Grok目前是Logstash中对非结构化日志数据解析成结构化和可查询的最好方式。logstash还可以重命名、删除、替换和修改事件字段，也可以完全丢弃事件。

1. 日志的两种采集状态。

主动探测

被动探测

1. 实时数据分析pipeline的架构。
2. 实时数据仓库pipeline的架构。

# 禧云国际

**Yarn的资源调度。**

**Yarn中的角色。**

**canal数据库消息变更订阅中心。**

**Spark的Checkpoint机制源码。**

**Spark的RDD的血缘关系。**

**Spark的RDD的解析。**

**Spark的RDD的迭代机制。**

**kafka中正在运行中，需要释放一部分的空间，会对kafka造成什么样的影响。**

**kafka如何保证语义的实现。**

**如何保证数据的质量。(时效性，和正确性)**

监控，

重试，

# 禧云国际

**Hive的分区和分桶**

**Hive的优化**

**CDH中是否还是用过其他的服务。**

**HDFS的写数据流程。**

**Spark的运行机制。**

**结合项目中的业务，描述Spark的运行机制。**

**Executor是如何去向Master进行注册Application的**

**Spark在运行过程中，如果一个Worker宕机，是如何解决的。**

在Spark中Worker与Master之间通过心跳来保持状态的更新，当一个Worker节点宕机后，Master通过心跳机制能够检查到超时的Worker，主要是通过调用timeOutDeadWorkers方法来检查超时的Worker，然后过滤处所有的超时Worker，即当前事件与WORKER\_TIMEOUT\_MS之差仍然大于WorkerInfo的lastHeartbeat的Worker节点。

过滤出所有的超时Worker后，根据WorkerInfo的状态进行处理。

1. 如果WorkerInfo的状态不是DEAD，则调用removeWorker方法，移除相关Worker的信息
2. 如果WorkerInfo的状态是DEAD，则等待足够长的时间后将它从wokers列表中移除：
   1. 足够长的时间的计算公式为：REAPER\_ITERATIONS与1的和再乘以WORKER\_TIMEOUT\_MS.

**哪些服务配置了高可用。**

**kafka增加节点时，如何进行动态的负载均衡**

通过Zookeeper达到负载均衡的实现。

* 每当一个Broker启动时，会首先完成Broker注册过程，在Zookeeper的节点列表中保存Broker。
* Kafka的生产者会对Zookeeper上的Broker的新增与减少、Topic的新增和减少和Broker和Topic关联关系的变化等事件注册Watcher监听
* 通过Zookeeper的Watcher通知能够让生产者动态的获取Broker和Topic的变化情况
* Kafka有消费者分组的概念，每个消费之分组包含了若干个消费者，每一条消息只会发送给分组内的一个消费者，不同消费者分组消费自己特定的Topic下面的消息，互不干扰
* Kafka会为每个消费者分配全局唯一的Consumer ID，采用Hostname：UUID形式来表示
* 每个消费者一旦确定了对一个消息分区的消费权力，Zookeeper会将其Consumer ID写入到对应消息分区的临时节点上
* 消费进度管理：kafka需要定时地将分区消息的消费进度，记录到Zookeeper上去。

**kafka的节点，主题，分区，每个分区的副本等数量。**

**kafka中topic的保留时长。**

**使用压缩格式的优点与缺点：Snappy**

Snappy的压缩率最低，但是编解码速率最高，对CPU的消耗也最小，目前一般建议使用Snappy。

**乂学教育**

Flume如何保证数据的安全性。

HDFS的存储原理

HDFS中blockSize大小的调整的好处

(1) 减少搜寻时间，一般硬盘传输速率比寻道时间要快，大的块可以减少寻道时间；

(2) 减少管理块的数据开销，每个块都需要在NameNode上有对应的记录；

(3) 对数据块进行读写，减少建立网络的连接成本

**类Hive的工具。**

impala

presto

Kylin

Spark on Yarn的任务运行流程

RDD，Spark是如何去进行加载数据的。遵循何种原则。

RDD之间的宽窄依赖

SparkStreaming的实时计算（新版本特性）

**Kafka中添加多个消费端，如何进行配置，提高kafka的并行度**

提高kafka的并行度可以通过调整kafka的partition数量来对kafka进行并行度的提高。

java的HashMap和Scala的Map有什么不同

scala中的val的Map变量，在底层的处理方式

java中的流式编程

<https://baymaxhuang.github.io/2016/12/04/Java-8%E6%96%B0%E7%89%B9%E6%80%A7%EF%BC%9A%E6%B5%81%E5%BC%8F%E5%A4%84%E7%90%86/>

<https://www.ibm.com/developerworks/cn/java/j-lo-java8streamapi/index.html>