优化思想： 减少网络输出量（计算靠近存储），减少计算量，尽可能使用内存

大数据四门语言R、Python、Scala、Java 比较 <http://www.sohu.com/a/72778066_355129> 。Maven，SBT，Gradle以及自动安装依赖关系都是针对Java和Scala的。Python一般只需要PiP。由Java和Scala应用程序构建一个JAR需要一段时间，对于许多Hadoop来说，JVM语言有一定的优势。您可以在JVM上运行JPython，但我真的没有看到用于大数据，Spark或机器学习。

Google于03至06年左右公布了三篇论文，描述了**GFS、BigTable、MapReduce**三种技术以解决这些问题。由于Google并没有公布算法细节，因此由雅虎牵头，在06年左右建立了开源项目Hadoop，目的是根据Google的三篇论文，实现一个大规模的管理计算系统。但直到08年，Hadoop同Google公布的一些关键指标仍有几倍的差距。百度曾经由王选院士的一个博士带领，想基于Google论文独立实现(金字塔计划)一个自己的系统，但开发难度过大项目夭折，最终也转向了Hadoop。如今，Amazon、Facebook、Yahoo包括百度都在大规模应用Hadoop，而Google已经从2010年开始迁移到新的三驾马车**Caffeine、Pregel、Dremel**上了。单就搜索技术而言，Google不是领先百度，而是领先全世界。2009-2012年，Google公布了世界上第一个全球化的数据库系统[**Spanner**](https://www.cnblogs.com/linbingdong/p/6388621.html)，这套系统将分布在全球各地的数据中心连接到一起，利用原子钟和GPS，打破了地理间隔，实现了全球规模具有一致性和实时性的数据库。

**BigTable**: GFS也需要Bigtable来存储结构化数据，每个Table都是一个多维的稀疏图，为了管理巨大的Table，把Table根据行分割，最终分布式存储

在Google采用Caffeine之前，Google使用MapReduce和分布式文件系统（如GFS）来构建搜索索引（从已知的Web页面索引中）。**Caffeine**使Google能够更迅速的添加新的链接（包括新闻报道以及博客文章等）到自身大规模的网站索引系统中，相比于以往的系统，新系统可提供“50%新生”的搜索结果。

**Pregel**主要绘制大量网上信息之间关系的“图形数据库”

于Pig和Hive，**Dremel**的数据查询几乎是瞬时的

**MegaStore** 文件存储（半结构化），在谷歌内部至少有 300 个应用使用 Megastore(尽 管它具有相对低的性能)，因为它的数据模型要比 BigTable 简单，更易于管理，并且支持在跨数据中心层面进行同步复制

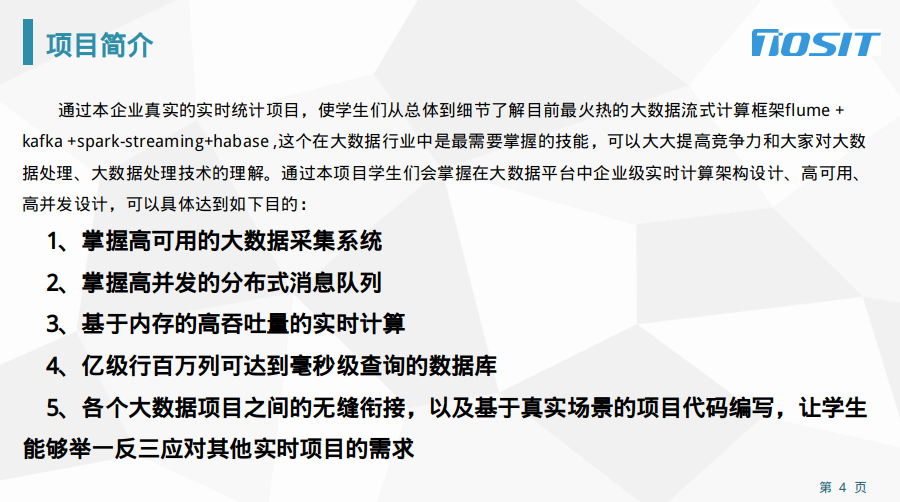
**Dapper** 分布式监控框架

大数据需要考虑数据的多样性（我觉得这应该作为一个历史遗留问题看待），数据的大量存储，数据产生的高速率（搜索引擎每天产生的日志有TB级别，这很明显不应该，大部分日志用于记录与方便程序panic的时候判断），数据的低质量（需要挖掘出高价值信息）

数据来源主要分三种情况，一是，自有平台产生，像腾讯、淘宝、华西医院。二是，靠网络采集系统，像百度、谷歌。三是，像第三方数据供应商提供，如运营商数据、开源大数据共享集合等。大数据即对大量数据中隐藏信息进行挖掘，比如zz做过通过某个身体部位的骨骼图片来识别年龄。

数据分片通用模型：key-partition/bucket（多到一，不是一对一估计是为了顺序读取的速度吧）、partition/bucket-machine（多到一）。中间加了一层说是为了key-machine解耦，但是我没有觉得。头条的abase对小数据直接key-machine，大数据会拆分（估计是为了增加读取大数据时候的并行性）。

hash映射支持点查询；范围映射支持范围查询。非一致性hash必须考虑机器个数才能够将让负载均衡起来，一致性hash允许了新节点找不到时向旧节点的渗透。

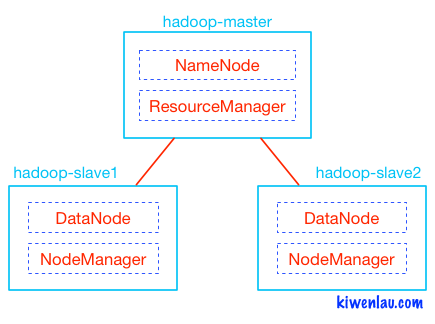


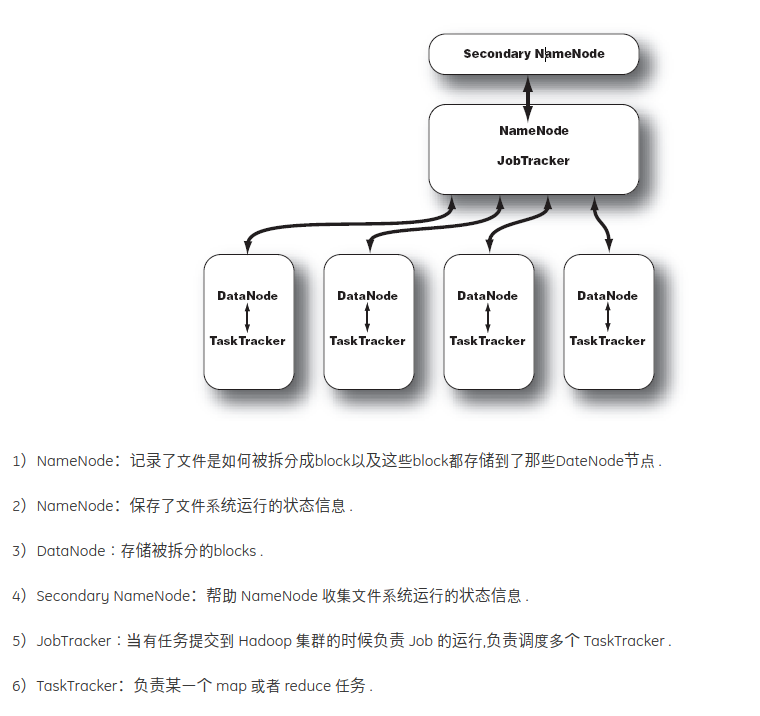
Google使用分布式数据库，却能够这么快返回检索结果，虽然有可能是缓存了所有相关的检索？

分布式系统中主服务器基本不会存储数据(hadoop等例外)，指引从服务器的数据位置

docker搭建Hadoop集群：https://github.com/kiwenlau/hadoop-cluster-docker

Hadoop的框架最核心的设计就是：HDFS和MapReduce（现在是一个很落后的并发处理框架）。HDFS为海量的数据提供了存储，则MapReduce为海量的数据提供了计算。Hadoop原本来自于谷歌一款名为MapReduce的编程模型包。谷歌的MapReduce框架可以把一个应用程序分解为许多并行计算指令，跨大量的计算节点运行非常巨大的数据集。Hadoop带有用[Java](https://baike.baidu.com/item/Java)语言编写的框架，因此运行在 Linux 生产平台上是非常理想的。Hadoop 上的[应用程序](https://baike.baidu.com/item/%E5%BA%94%E7%94%A8%E7%A8%8B%E5%BA%8F)也可以使用其他语言编写，比如 [C++](https://baike.baidu.com/item/C++)。通常有较高的延迟并且在作业提交与调度的时候需要大量开销。





只有可以将数据独立分离计算与合并的任务才可以用hadoop来搞定，即可分割。但大量的实战证明，绝大多数的计算任务都可以通过合理的设计化分成可以划分成可以hadoop处理的任务。但像gzip压缩包解压缩这样的操作，一般认为是无法用hadoop来并行计算的，因为gzip是不可分割的，但像lzo,bz2等可分割的压缩格式，均可以被hadoop处理。

HDFS Hadpoop Distribute File System :

被设计成通用硬件上的分布式文件系统（java写的，基于jvm）。HDFS是一个主从结构，一个HDFS集群是由一个名字节点，它是一个管理文件命名空间和调节客户端访问文件的主服务器，当然还有一些数据节点，通常是一个节点一个机器，它来管理对应节点的存储。HDFS对外开放文件命名空间并允许用户数据以文件形式存储。一个文件被分块存储在多个数据节点上。负责海量数据的存储，集群中的角色主要有 NameNode / DataNode。

DistributedFileSystem通过RPC（Remote Procedure Call）调用询问NameNode。

三种node：namenode、datanode、clientnode（这个node与数据存储无关，但可以发出mapreduce job等人为操作）

HDFS也是采用块管理的，但是比较大，在Hadoop1.x中默认大小是64M，Hadoop2.x中大小默认为128M，官方的解释是减少寻址开销，也就是让文件传输时间明显大于块的寻址时间。

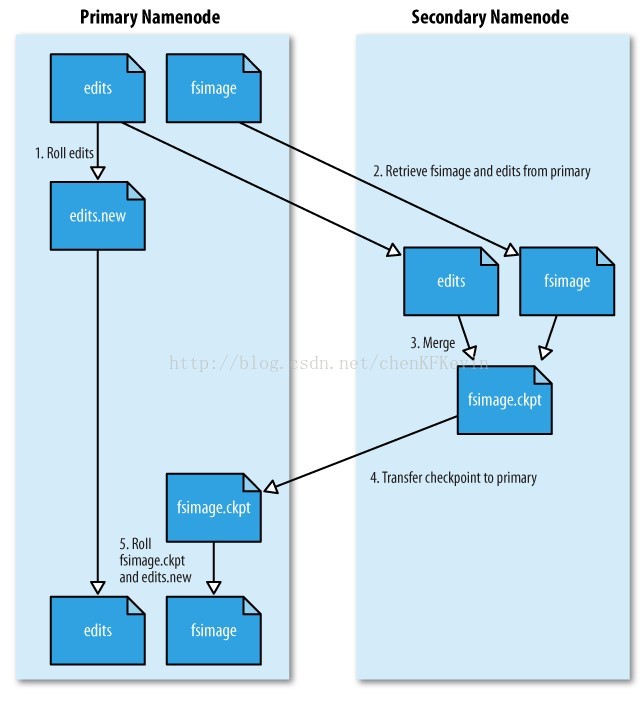
一个假定就是迁移计算到离数据更近的位置比将数据移动到程序运行更近的位置要更好。HDFS提供了接口，来让程序将自己移动到离[数据存储](https://baike.baidu.com/item/%E6%95%B0%E6%8D%AE%E5%AD%98%E5%82%A8)更近的位置。

机器之间的的通讯基于TCP/IP，运行上的数据量大小GB、TB级别。

分布式文件系统的容错只能通过对文件块的复制。datanode直接进行复制就好了，没有必要经过namenode。

hdfs 有snapshot，这个功能出来的比较迟，简单的备份即便是增量备份也会有不小的存储开销，所以备份也会是分布式的，估计就和存储块的那个备份功能（有个函数可以设定备份几个块）相关。snapshot 的存储开销与hadoop的备份相比，其实也不是很大。

hdfs有checkpoint，用于持久化结果方便之后快速使用。edits是在每次修改HDFS时都会插入记录，那么fsimage则在整个HDFS运行期间不会产生变化，只有在每次启动Namenode时(结束的时间是不知道的)，当然肯定还有显示执行合并的命令，才会把edits中的操作增加到fsimage中，并且把edits清空。所以fsimage总是记录启动Namenode时的状态，而edits在每次启动时也是空的，它只记录本次启动后的操作日志。按照fsimage和edits的工作机制(除非指定了合并周期)，在一次启动后，edits的文件可能会增长到很大，这样在下次启动Namenode时需要花费很长时间；另一方面，如果在HDFS运行过程中发生Namenode的故障，那么edits中的记录就会丢失。所以，我们需要利用Checkpoint即使将修改操作持久化。Checkpoint是将没有永久化的数据进行一次永久化存储，因为hdfs自带容灾机制。



第一步：将hdfs更新记录写入一个新的文件——edits.new。

第二步：将fsimage和editlog通过http协议发送至secondary namenode。

第三步：将fsimage与editlog合并，生成一个新的文件——fsimage.ckpt。这步之所以要在secondary namenode中进行，是因为比较耗时，如果在namenode中进行，或导致整个系统卡顿。

第四步：将生成的fsimage.ckpt通过http协议发送至namenode。

第五步：重命名fsimage.ckpt为fsimage，edits.new为edits。

放到secondary namenode上合并：（1） 减少了主namenode的资源占用（2）因为是复制合并，所以一旦合并失败，对于原数据也不会有影响。

运维：

UI查看工具： hadoop explorer

fsimage和edits等文件中的内容都是经过序列化，所以不能直接查看。不过hadoop默认提供了查看工具，对应两种文件分别是oiv、oev（offline edits viewer），执行命令：默认输出都是xml格式

[hadoop@hadoop current]$ hdfs oiv -i fsimage\_0000000000000000115 -o fsimage.ls

输出ls -l 命令查看一样的文本格式

[hadoop@hadoop current]$ cat fsimage.ls

drwxr-xr-x - hadoop supergroup 1412832662162 0 /

drwxr-xr-x - hadoop supergroup 1413795010372 0 /user

drwxr-xr-x - hadoop supergroup 1414032848858 0 /user/hadoop

drwxr-xr-x - hadoop supergroup 1411626881217 0 /user/hadoop/input

drwxr-xr-x - hadoop supergroup 1413770138964 0 /user/hadoop/output

hdfs oev [OPTIONS] -i INPUT\_FILE -o OUTPUT\_FILE

hdfs oev -i edits\_0000000000000000081-0000000000000000089 -o edits.xml

输出xml格式

GFS ： Google File System

YARN：

负责海量数据运算时的资源调度，集群中的角色主要有 ResourceManager /NodeManager

jps命令可以在datanode上看到yarnchild。

MRAppMaster是MapReduce的ApplicationMaster实现，它使得MapReduce可以直接运行在YARN上，老师在datanode上看到的不是yarnchild而是MRAppMaster。

运行一个任务之后（比如mapreduce），就会在HDFS的/tmp中产生一些记录文件。

MapReduce： map -->shuffle-->reduce

mapreduce只是hadoop的一种Application Type，最后结果是按照key的从小到大顺序输出，虽然两次排序都是可以自己定制的，但是如果有需要最后按照value的排序来输出，最好不要用MapReduce，或者https://blog.csdn.net/ymybxx/article/details/78581235 。

hadoop的map/reduce都是以进程为单位进行计算的，reduce设置的过多会导致reduce的初始化与销毁浪费时间，从而影响整个任务的效率。要根据输出数据量来计算reduce的数量。

job：运行作业的客户端通过调用getSplits()计算分片，然后将他们发送到jobtracker。jobtracker使用其存储位置信息来调度map任务从而在tasktracker上处理这些分片数据。

map与combine以及reduce其实都是函数操作，map甚至可以用于实现filter、数据分片

map：map函数接受一个键值对（key-value pair），产生一组中间键值对（由输入的value得到许多value）。map任务把经过规则处理后的分片（有个默认规则是把每一行文本内容解析成键值对，这里的“键”是每一行的起始位置(单位是字节)，“值”是本行的文本内容）传给InputFormat的getRecordReader（）方法来获得这个分片的RecordReader。RecordReader就像是记录上的迭代器，map任务通过调用mapper的run（）方法用一个RecordReader来生成记录的键/值对，进而将该键/值对传给mapper的map方法作为输入。

Combine：

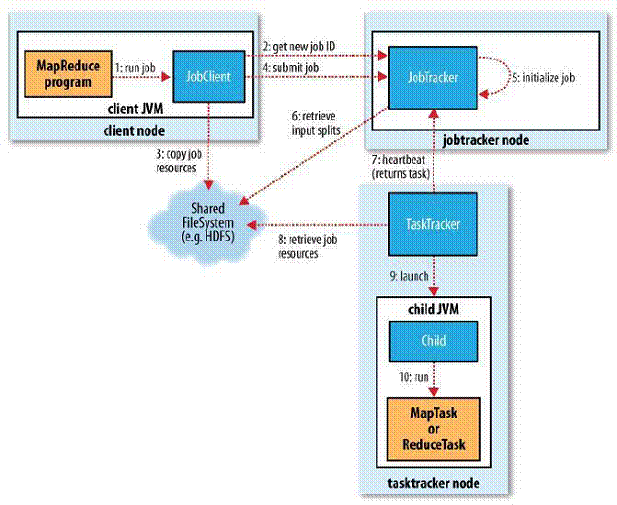
hadoop提供combiner来合并map输出的零散小数据，从而减小网络传输，提高整体的效率。还有减少对磁盘的IO。Combine在整个过程中会多次触发。

mapper方法的输出按照**partition**规则，将输出写入不同的环形缓冲区中（默认对key hash后再以reduce task数量取模。默认的此方式只是为了平均reduce的处理能力）（桶排序思想）。输出的过程中存在按照序列化的key排序与**combine**（按照reduce规则合并同key数据的value）。当内存缓冲区快满的时候就会溢写到磁盘，如果内存缓冲区已经满了就会阻塞map任务，持续对磁盘的写入。最后所有的输出都会写入磁盘，之前溢写是流写入，现在是块写入。

Task Tracker会不断询问Map Task是否完成，如果某台Task Tracker上的Map Task已经完成，就会把这多个临时文件合并，即做merge操作，注意，这里的merge操作只是简单的合并，如果没有在该处设置Combiner，是不会对相同key进行压缩的，所以可能会有相同的key出现。**merge**操作就是对于同样的key，其value变为list，把多个value放在list中。这种key/value的形式就是reduce的输入数据格式。

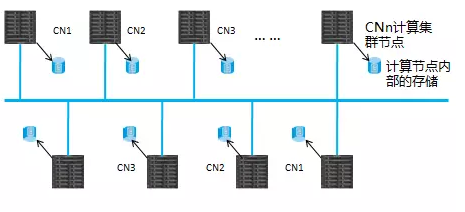
reduce函数（包含Shuffle，即pull方式合并拉取map的结果）：接受一个键，以及相关的一组值，将这组值进行合并产生一组规模更小的值（通常只有一个或零个值）。

在统计词频的例子里，map函数接受的键是文件名，值是文件的内容，map逐个遍历单词，每遇到一个单词w，就产生一个中间键值对<w, "1">，这表示单词w咱又找到了一个；MapReduce将键相同（都是单词w）的键值对传给reduce函数，这样reduce函数接受的键就是单词w，值是一串"1"（最基本的实现是这样，但可以优化），个数等于键为w的键值对的个数，然后将这些“1”累加就得到单词w的出现次数。最后这些单词的出现次数会被写到用户定义的位置，存储在底层的分布式存储系统（GFS或HDFS）。

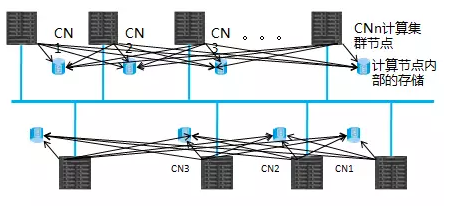


分布式文件系统历史发展：

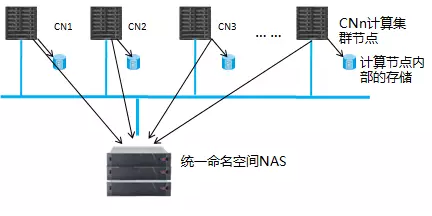
在计算集群形成初期，系统中并没有复杂的存储系统存在，甚至没有统一存储的概念，那时每台计算节点把自己所计算出的数据先暂时写到自己的本地硬盘上(Cache)，最终由主节点进行所有数据的回收，至主节点所挂载的相对大的空间中(其实这就是后来HPC存储的雏形)，再进行后续处理(分析、共享、可视化、备份与恢复)：



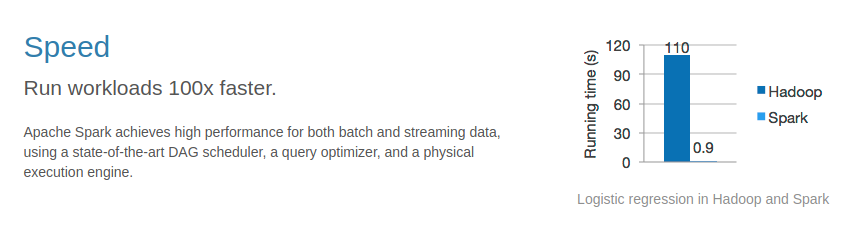
后来因为计算任务的划分导致节点上计算的数据可能不一定就要存储在本节点。spark采用TCP交换节点之间的信息：



因此后续的程序编写中，会让所有的节点都认到一块区域，将所有的临时性数据和最终数据都写到这里，即统一命名空间，因为所有计算节点都可以看到这块同一个名字的数据存储区域，表现在计算节点的OS中即为看到同一个名字的目录。即Hadoop的NameNode工作：



Spark:



Spark，拥有Hadoop MapReduce所具有的优点；但不同于MapReduce的是——Job中间输出结果可以保存在内存中，从而不再需要读写HDFS，因此Spark能更好地适用于数据挖掘与机器学习等需要迭代的MapReduce的算法。与Hadoop的MapReduce相比，Spark基于内存的运算要快100倍以上，基于硬盘的运算也要快10倍以上（明显夸张了，本质上主要是靠减少了磁盘IO次数）。Spark实现了高效的DAG执行引擎，可以通过基于内存来高效处理数据流。Spark可以使用Hadoop的YARN和Apache Mesos作为它的资源管理和调度器。

和hadoop的多进程相比，spark采用fork线程

Shark = Spark+HiveQL 被SparkSQL取代

SparkR = Spark + R

SparkMaster：

本地：local[4]，local[\*]（这个表示有几个逻辑内核就是数字几），数字指定了CPU资源个数，估计是将每一个CPU逻辑内核作为一个运算节点

WebUI: 8080

URL:          spark://supermap:7077  
 REST URL: spark://supermap:6066(cluster mode)

Spark-shell会自动创建一个 [SparkContext](http://spark.apachecn.org/docs/cn/2.2.0/api/scala/index.html#org.apache.spark.SparkContext) 对象，名字叫做sc。”:quit” 退出spark-shell

弹性分布式数据集：RDDs are created by starting with a file in the Hadoop file system (or any other Hadoop-supported file system), or an existing Scala collection in the driver program, and transforming it.它代表一个不可变、可分区、里面的元素可并行计算的集合。RDD允许用户在执行多个查询时显式地将工作集缓存在内存中，后续的查询能够重用工作集，这极大地提升了查询速度。

 Spark中类似于RDD的抽象还有可分享变量，可分享变量分为两种：1.广播变量 2.累加器（比如counters 、 sums）。广播变量和RDD都是只能读取不能改动，累加器是可以修改的。

当前Spark中实现了两种类型的分片函数，一个是基于哈希的HashPartitioner，另外一个是基于范围的RangePartitioner

窄依赖指的是每一个父RDD的Partition最多被子RDD的一个Partition使用，宽依赖指的是多个子RDD的Partition会依赖同一个父RDD的Partition。宽依赖意味着最后一个父RDD需要被传送到多个机器上进行多个子RDD的运算。

缓存：RDD的Lineage会记录RDD的元数据信息和转换行为，当该RDD的部分分区数据丢失时，它可以根据这些信息来重新运算和恢复丢失的数据分区。持久化（不同于checkpoint），RDD通过persist方法或cache方法可以将前面的计算结果缓存，但是并不是这两个方法被调用时立即缓存，而是触发后面的action时，该RDD将会被缓存在计算节点的内存中，并供后面重用。



默认的存储级别都是仅在内存存储一份，Spark的存储级别还有好多种，存储级别在object StorageLevel中定义的，作为参数传入persist():



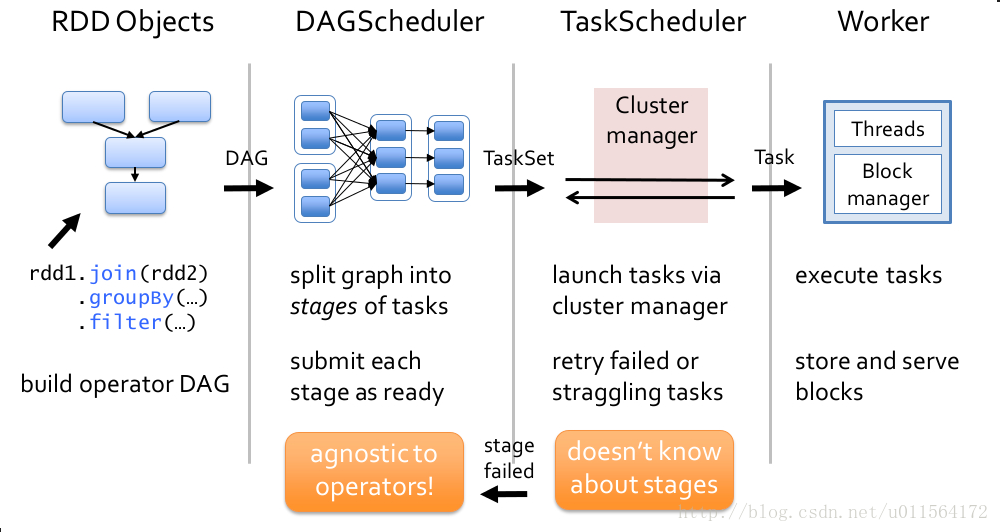
缓存有可能丢失，或者存储存储于内存的数据由于内存不足而被删除，RDD的缓存容错机制保证了即使缓存丢失也能保证计算的正确执行。通过基于RDD的一系列转换，丢失的数据会被重算，由于RDD的各个Partition是相对独立的，因此只需要计算丢失的部分即可，并不需要重算全部Partition。

专门介绍rdd的文档：Spark RDD.docx

Stage的划分：DAG(Directed Acyclic Graph)叫做有向无环图，原始的RDD通过一系列的转换就就形成了DAG，根据RDD之间的依赖关系的不同将DAG划分成不同的Stage，对于窄依赖，partition的转换处理在Stage中完成计算。对于宽依赖，由于有Shuffle的存在，只能在parent RDD全部处理完成后，才能开始接下来的计算，因此宽依赖是划分Stage的依据。



运行流程：



spark-core中有RDD、scheduler文件夹下DAGScheduler、TaskScheduler类

spark-core中还有Partitioner、Shuffle相关的

Spark SQL:与RDD类似，DataFrame也是一个分布式数据容器。然而DataFrame更像传统数据库的二维表格（想到Google的BigTable），除了数据以外，还记录数据的结构信息，即schema。同时，与Hive类似，DataFrame也支持嵌套数据类型（struct、array和map）。从API易用性的角度上 看，DataFrame API提供的是一套高层的关系操作，比函数式的RDD API要更加友好，门槛更低。

SparkSQL类已经过时，改用SparkSession类，SparkSession 合并了 SQLContext 和 HiveContext

DataSet简称DS ，DataFrame简称DF，type DataFrame = Dataset[Row]

case class Person(name: String, age: Long)

val caseClassDS = Seq(Person("Andy", 32)).toDS()

val peopleDF = spark.sparkContext

.textFile("D:\\scdx\\spark.bigdata.mr\\src\\main\\resources\\people.txt")

.map(\_.split(","))

.map(attributes => Person(attributes(0), attributes(1).trim.toInt))

.toDF()

Spark Streaming是在Spark上建立的可扩展的高吞吐量实时处理流数据的框架，数据可以是来自多种不同的源，例如kafka，Flume，Twitter，ZeroMQ或者TCP Socket等。根据batchInterval时间片不断生成Job，并将Job提交集群处理。

将sparkStringContext的batchInterval时间设置长一些，通过web管理界面能看到很多job：负载均衡、接受数据、我们自己编写的业务逻辑：<https://blog.csdn.net/erfucun/article/details/52291761>



之所以是nc -lkp xxx，是因为streaming程序如果还没有建立连接，每隔一段时间就会来连接目标端口，而nc -lkp 会将stdin作为对指定端口的输出，所以内容就这样传过去了。

nc -nv 与 nc -lkp是可以相互传递信息的

val rdd = sc.parallelize(List("1","2","3","4","5","1","5","3"),3).map((\_,1)).reduceByKey(\_+\_)

val rdd2 = sc.parallelize(List("a","b","c","d","e","f"),2) 第二个参数指定了分区数量，否则会是一个默认的值，这个值似乎不同环境不同

使用reduceByKey比使用groupByKey在效率上要好一些，这是因为reduceByKey比groupByKey在shuffle之前多了一步merge操作。

map与mapPartitions的区别：<https://blog.csdn.net/xingzhiqing/article/details/56304155> 设计到的资源初始化数量不同

//build.gradle

plugins {  
 id 'scala'  
}  
  
group 'cn.hellovega'  
version '1.0-SNAPSHOT'  
  
sourceCompatibility = 1.8  
  
repositories {  
 mavenCentral()  
}  
  
dependencies {  
 implementation 'org.apache.spark:spark-core\_2.11:2.3.1'  
 implementation 'org.apache.spark:spark-sql\_2.11:2.3.1'  
}//2.11是scala版本 2.3.1是spark版本

需要的库的名字：<http://mvnrepository.com/artifact/org.apache.spark>

gradle是Maven的简化版

数据仓库Data Warehouse， 研究与解决从数据库中获取信息的问题，帮助决策者从大量数据中分析出有用的信息，所以说是面向主题的。

Hive ： hadoop的一个数据仓库工具，将结构化的数据文件映射为一张数据表，并提供简单的sql查询功能，可以将sql语句转换为MapReduce任务运行。Hive 不支持对数据的改写和添加，所有的数据都是在加载的时候确定的，数据仓库的作用就是分析数据。支持web UI、命令行操作。

Zookeeper是一个高效的分布式协调服务，可以提供配置信息管理、命名、分布式同步、集群管理、数据库切换等服务。它不适合用来存储大量信息，可以用来存储一些配置、发布与订阅等少量信息。

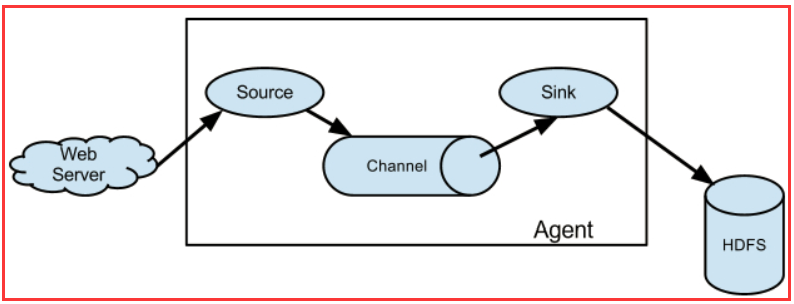


Kafka  是一种高吞吐量的分布式发布订阅消息系统，基于内存的消息队列，负责缓冲数据，供实时分析程序提取。

broker id取值范围大于等于-1

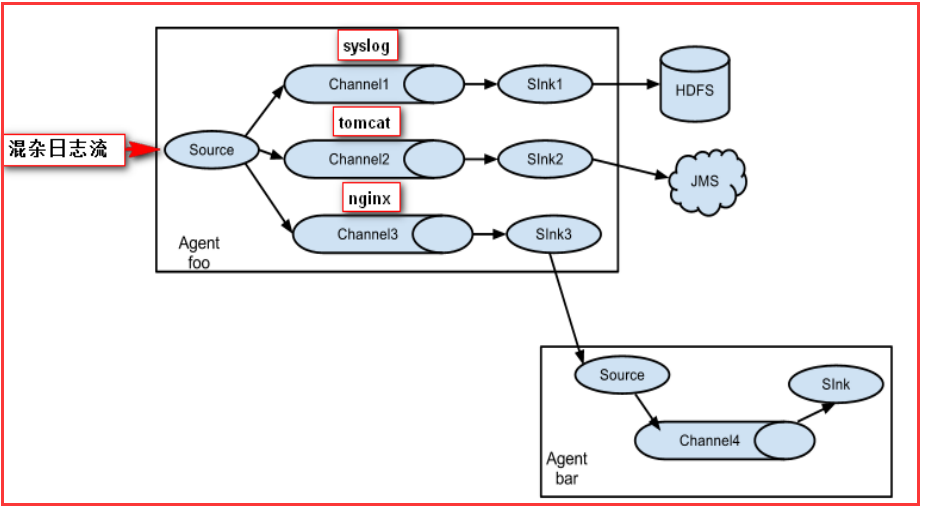
kafkaUtils.createDirectStream 不与zookeeper交互，直接去kafka中读取数据，自己维护offset，于是速度比KafkaUtils.createStream要快上很多。kafkaUtils.createDirectStream

Flume是一款管道工具，常见用于分布式日志采集系统(海量日志采集、聚合和传输)，提供了从console（控制台）、RPC（Thrift-RPC）、text（文件）、tail（UNIX tail）、syslog（syslog日志系统，支持TCP和UDP等2种模式），exec（命令执行）等数据源上收集数据的能力。



source后面可以设置filter

一个source可以连接多个channel，不同的channel可以要不同的filter，每个channel到一个sink，然后不同sink之间可以指定不同的文件存储系统。甚至sink后的内容重新作为一个source。



Source：

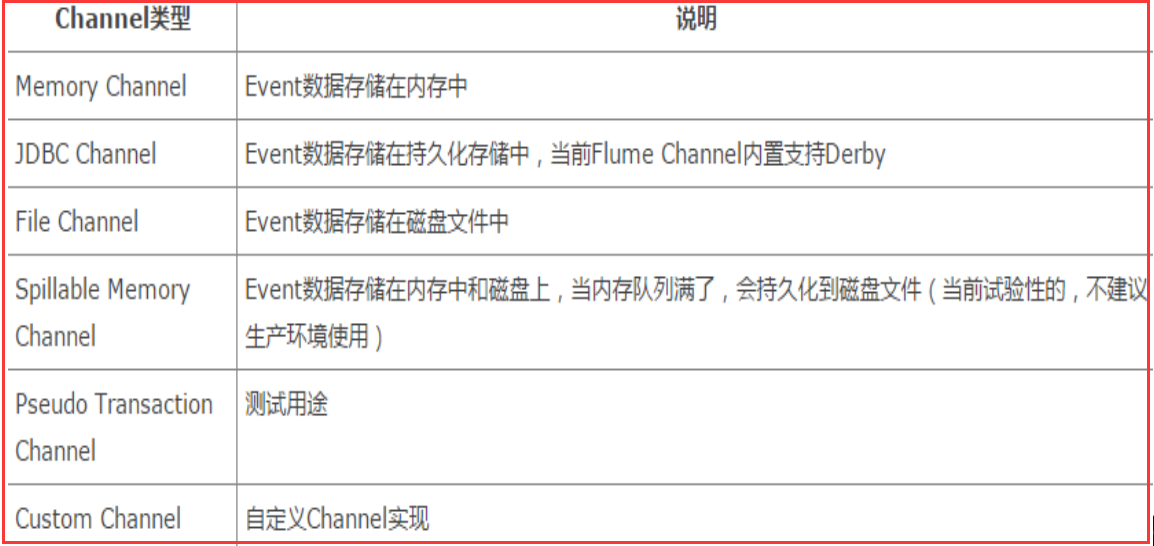
log4j有一个TimeRolling的插件，可以把log4j分割的文件到spool目录



channel通过event实现了“通知性缓存”：

MemoryChannel可以实现高速的吞吐， 但是无法保证数据完整性

FileChannel保证数据的完整性与一致性。建议FileChannel设置的目录和程序日志文件保存的目录设成不同的磁盘，以便提高效率。



sink实现了数据转发

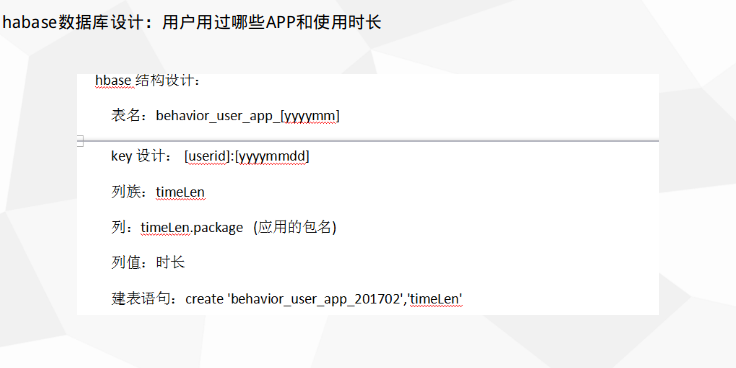
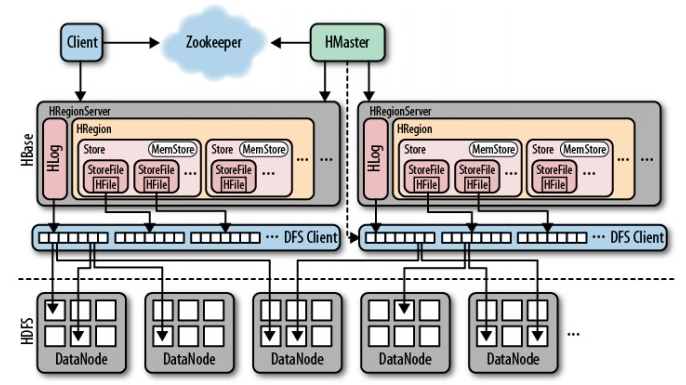
org.apache.flume.sink.kafka.KafkaSink 配合kafka使用的sink（或者自定义并包含Kafka Producer）https://www.cnblogs.com/mengyao/p/4526058.html



按照用户在网站上停留的时间的长度来划分用户的行为组。要及时得知用户已经下线，这就需要每隔一段时间就刷一下。比如DWR长连接实现。

数据上的分片存储（负载均衡），还需要均衡器进行数据迁移

HBase 分布式非关系型数据库，面向列存储，列族是表的chema的一部分(而列不是)，必须在使用表之前定义。列名都以列族作为前缀。每个 RegionServer 包含多个 Region，每个 Region 包含多个Store（每一个列族对应一个Store），每个 Store 包含一个 MemStore 和多个 StoreFile。



举例，列族里面可能会有这样的列名：timeLen.org.deeipn.wine.qq，值就是使用过的时长

对表还有disable、enable操作

hbase自带zookeeper，分布式Apache HBase安装依赖于正在运行的ZooKeeper集群。Apache HBase默认情况下为您管理ZooKeeper“集群”。它将启动和停止ZooKeeper集合作为HBase启动/停止过程的一部分。你还可以独立于HBase管理ZooKeeper集群，只需要在Hbase的配置文件hbase-env.sh中做一些设置即可：export HBASE\_MANAGES\_ZK=false。

hbase shell操作 <https://www.cnblogs.com/ityouknow/p/7344001.html>

在hbase shell中使用list指令是一个很慢的操作，容易导致timeout

Hbase的优点   
 1 列的可以动态增加，并且列为空就不存储数据,节省存储空间.  
 2 Hbase自动切分数据，使得数据存储自动具有水平scalability.  
 3 Hbase可以提供高并发读写操作的支持  
 Hbase的缺点：  
 1 不能支持条件查询，只支持按照Row key来查询.

2 暂时不能支持Master server的故障切换,当Master宕机后,整个存储系统就会挂掉.

Hbase 热点倾斜问题：

治标不治本 <https://blog.csdn.net/Spark2creat_palains/article/details/73611122> 还是应该一致性hash

hadoop官网示例代码：<https://wiki.apache.org/hadoop/Hbase/Scala>

HBaseAdmin admin = new HBaseAdmin(conf);

In place this, You should use:

Connection conn =ConnectionFactory.createConnection(conf);

Admin admin = conn.getAdmin();

hbase操作Demo：

**import** org.apache.hadoop.conf.Configuration  
**import** org.apache.hadoop.hbase.{HBaseConfiguration, TableName}  
**import** org.apache.hadoop.hbase.client.ConnectionFactory  
  
**object** HbaseDemo {  
 **def** main(args: Array[String]): Unit = {  
 **val** conf: Configuration = HBaseConfiguration.*create* conf.set("hbase.zookeeper.quorum","192.168.27.101,192.168.27.102,192.168.27.103,192.168.27.104")  
 conf.set("hbase.rootdir", "hdfs://192.168.27.101:9000/hbase")  
 **val** conn =ConnectionFactory.*createConnection*(conf);  
 //连接创建之后有两种函数 getAdmin从而建表、删表、删除列 getTable获取表对象，进行delete、put、get、scan等操作  
 **val** admin = conn.getAdmin  
 *print*(admin.listTableNames().length)  
 **val** t = conn.getTable(TableName.*valueOf*("T\_TEST"))  
   
 }  
}

build.gradle

plugins {  
 id 'scala'  
}  
  
group 'cn.hellovega'  
version '1.0-SNAPSHOT'  
  
repositories {  
 mavenCentral()  
}  
  
dependencies {  
 implementation 'org.apache.hbase:hbase-client:2.0.1'  
 implementation 'org.apache.hbase:hbase-common:2.0.1'  
}

添加记录是下面这样，先用rowkey创建put对象，然后调用addColumn完善列信息与put上传记录：

val put = new Put(Bytes.toBytes(rowKey))

put.addColumn(Bytes.toBytes("timeLen"),Bytes.toBytes(hour), Bytes.toBytes(secondOfHour)) //函数名字很能体现是列存储

table.put(put)

ElasticSearch是一个基于Lucene的搜索服务器。它提供了一个分布式多用户能力的全文搜索引擎，基于RESTful web接口。通过curl配合参数就将json格式的数据以HTTP格式传送到端口进行数据操作。