特此声明

本视频由whoami博主（博客：itweet.cn） Hadoop十周年庆开始公益录制，《Enterprise_Hadoop_Solutions》系列，注重Hadoop企业级应用案例实践，特此声明此教程使用内容大部分来自网络,版权属于广大网友;任何个人或企业未经授权不得作商业用途。如想持续关注，可扫描如下微信二维码关注视频更新。

Enterprise_Hadoop_Solutions

SQL on Hadoop

作者：whoami

走向分布式?

一个系统走向分布式，一定有其不得不为的理由。 可扩展性是最常见的理由之一。我先简单的将“可伸缩”的需求分成两种：

- Data Scalability: 单台机器的容量不足以 (经济的) 承载所有资料，所以需要分散。如：NoSQL

- Computing Scalability: 单台机器的运算能力不足以 (经济的) 及时完成运算所以需要分散。如：科学运算。不管是哪一种需求，在决定采用分布式架构时，就几乎注定要接受一些牺牲：

  1.牺牲效率：网路延迟与节点间的协调，都会降低执行效率。

  2. 牺牲 AP 弹性：有些在单机上能执行的运算，无法轻易在分布式环境中完成。

  3.牺牲维护维运能力：分散式架构的问题常常很难重现，也很难追踪.另外，跟单机系统一样，也有一些系统设计上的 tradeoffs(权衡)

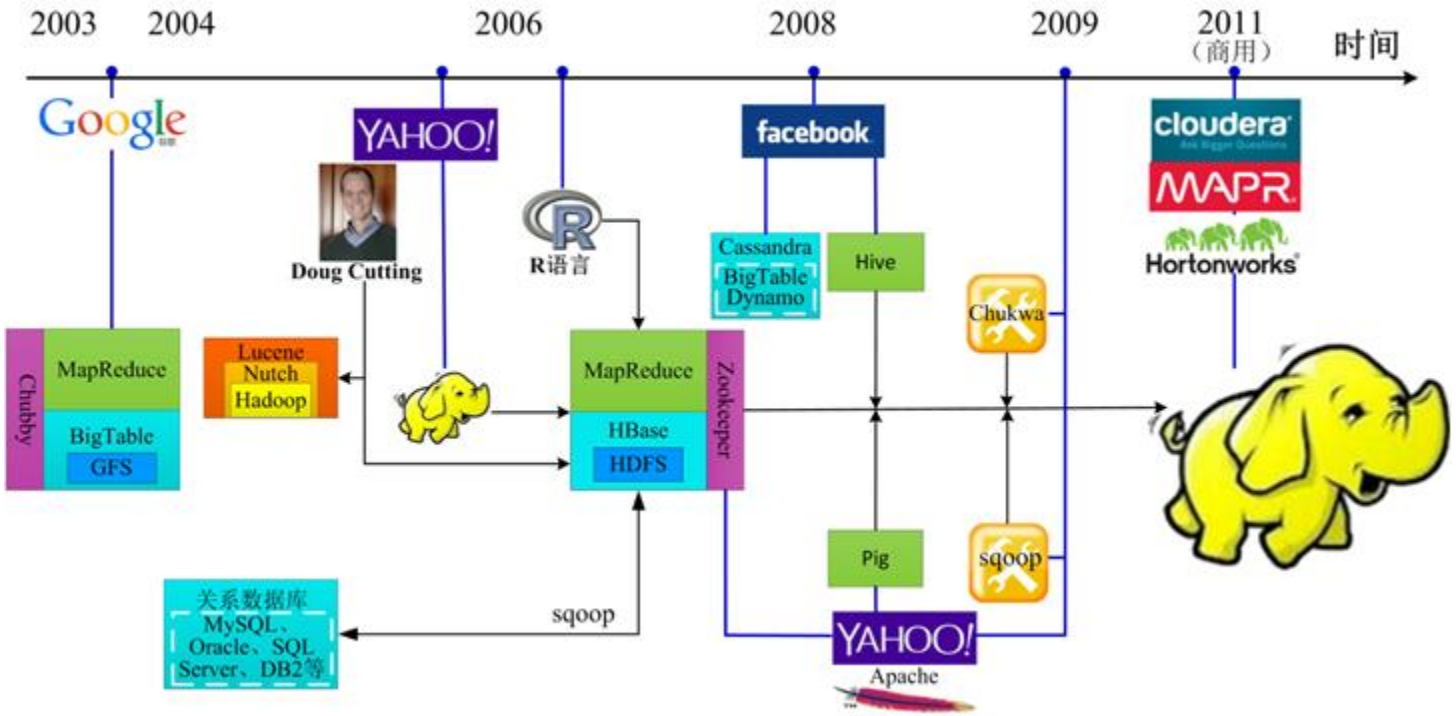
  4. CPU 使用效率优化或是 IO 效率优化

  5.读取优化或是写入优化

  6. 吞吐率优化或是 网络延迟优化

  7.资料一致性或是资料可得性,选择了不同的 tradeoff，就会有不同的系统架构。

Hadoop Version



Hadoop几大发行商？

目前大数据解决方案

其他方案

现有几大Hadoop平台比较

Hortonworks HDP 2

- 特点

纯开源
原装Hadoop(做了rpm包)
版本跟进快
Hive+tez(主推)
企业级安全(不开源)
监控集成方案：
ganglia+nagios
Spark合作(原装)
贡献Hadoop源代码为主？

- 缺点

升级(坑,各种版本升级方式还不同?)
配置管理?
添加删除节点?
维护升级成本？

Cloudera CDH 5

- 特点

CM小白式安装
监控,使用可视化程度高
CM修改配置方便
小白式升级
资源分配设置可视化
文档非常详细
Impala(主推),维护成本低
Spark合作(功能阉割)

- 缺点

CM不开源,企业级(Navigator付费)
版本跟进相对保守
社区版本不透明,
功能有些许小限制？

SQL on hadoop

- Hive [Tez,hive on spark]
  http://hive.apache.org/ 

- Spark-sql /shark(spark on hive)
  https://spark.apache.org/ 

- Impala 

  http://www.cloudera.com/

- phoenix+hbase 

  && Cassandra 

  http://phoenix.apache.org/

- Drill 

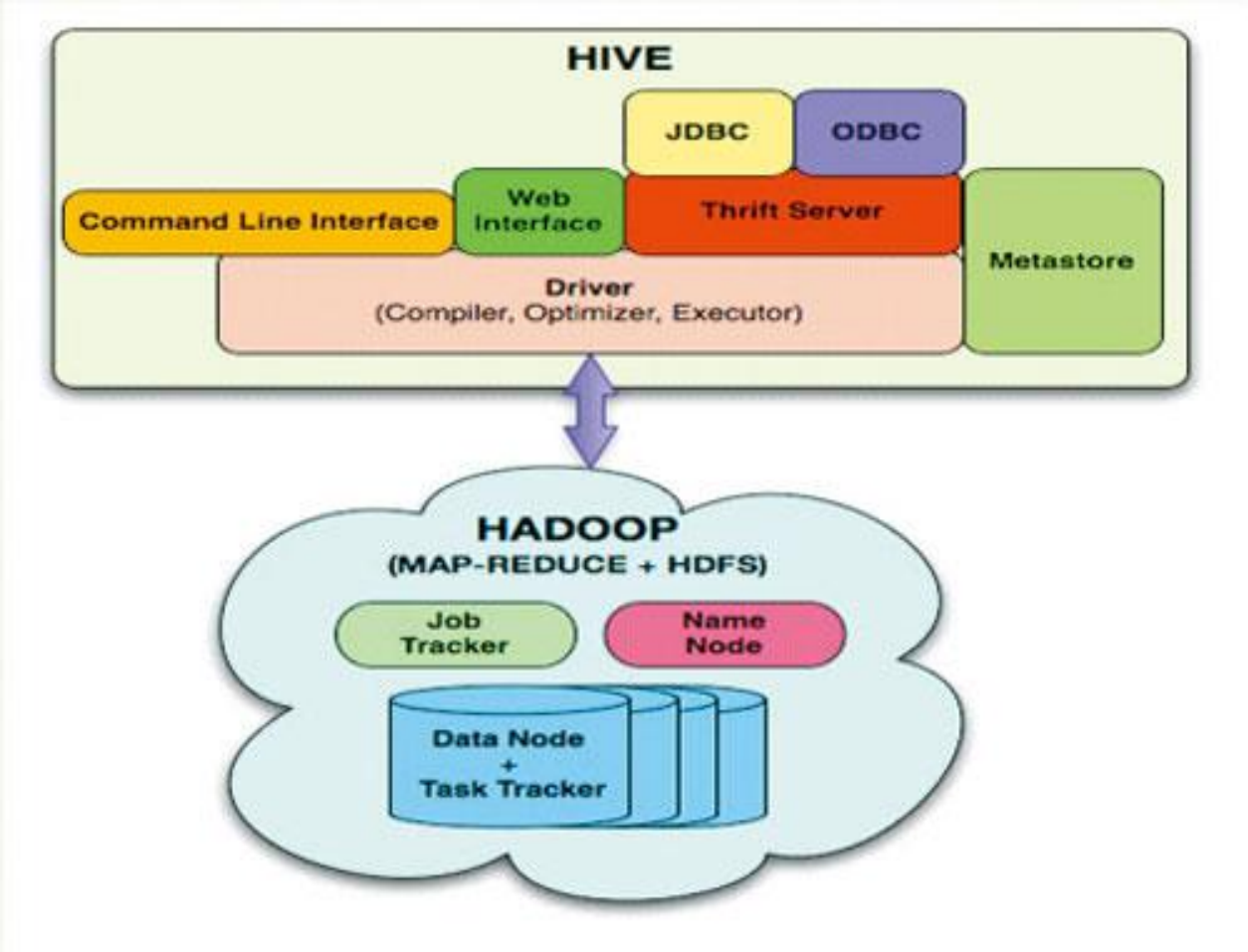
  http://drill.apache.org/

- Presto 

  http://prestodb.io/docs/current/

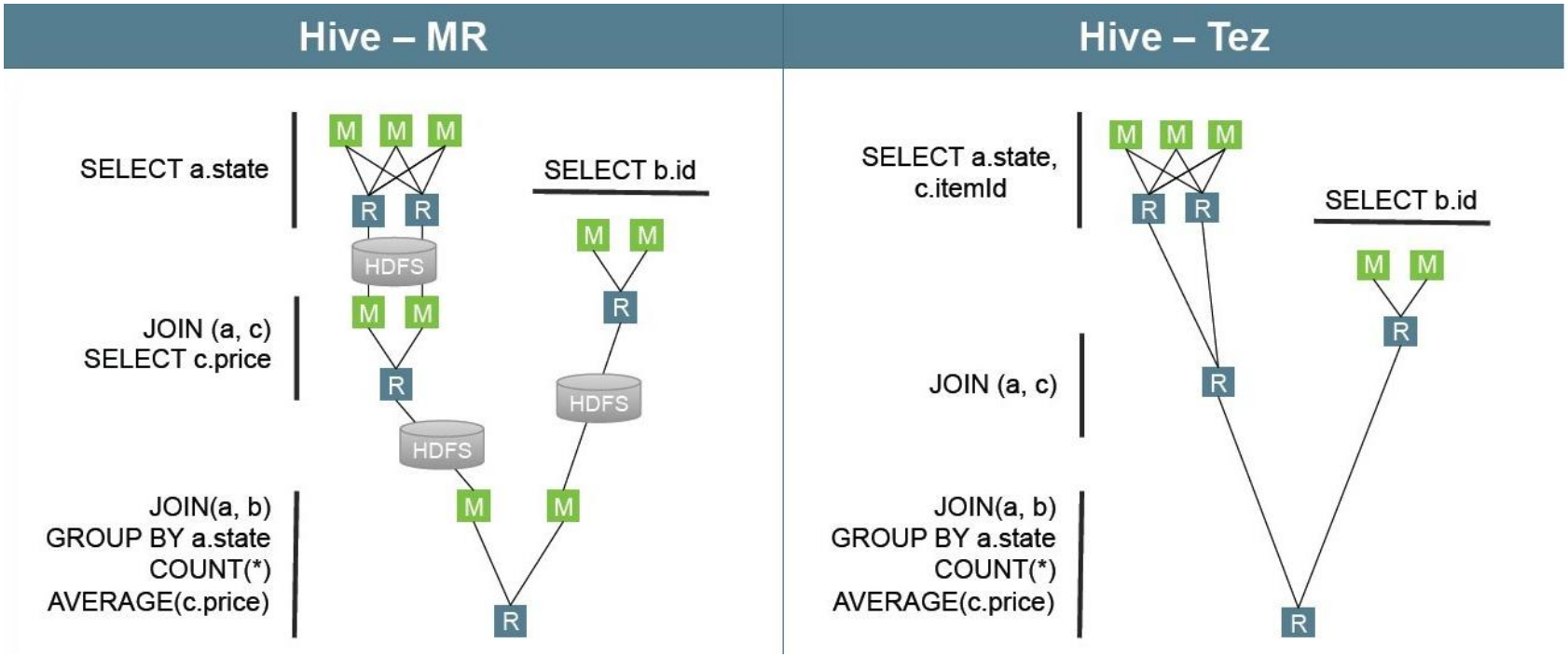
TPC性能测试: http://www.itweet.cn/2016/01/29/SQL-on-Hadoop-tpc-testing/

Hive



http://www.itweet.cn/2015/07/10/hive0.13.1-install/

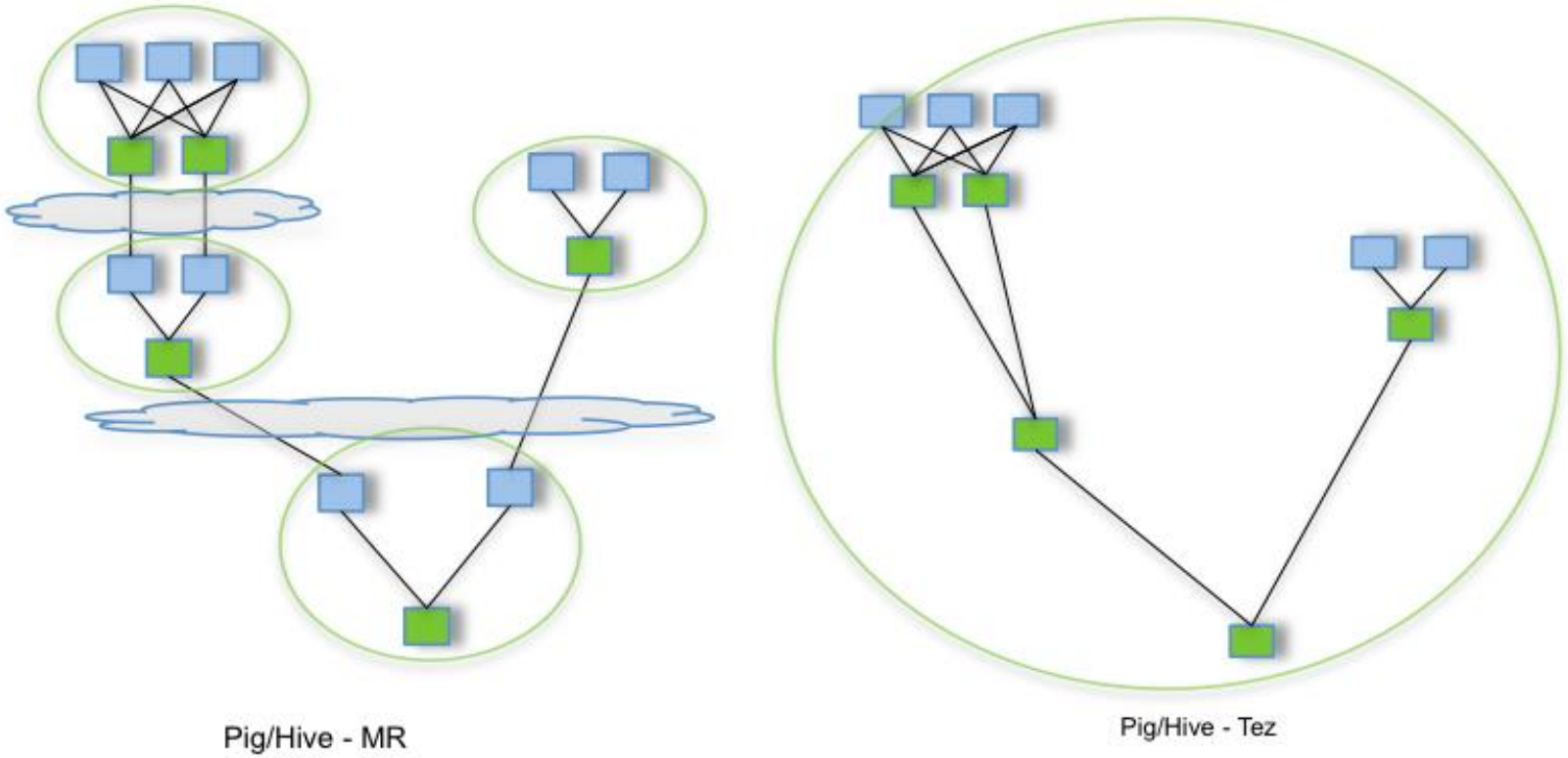
Hive on Tez



http://www.itweet.cn/2015/07/20/tez-use/

http://zh.hortonworks.com/hadoop-tutorial/supercharging-interactive-queries-hive-tez/

Hive on Tez



To download the Apache Tez software, go to the Releases page.

http://tez.apache.org/

Shark

Shark

Development ending; transitioning to Spark SQL

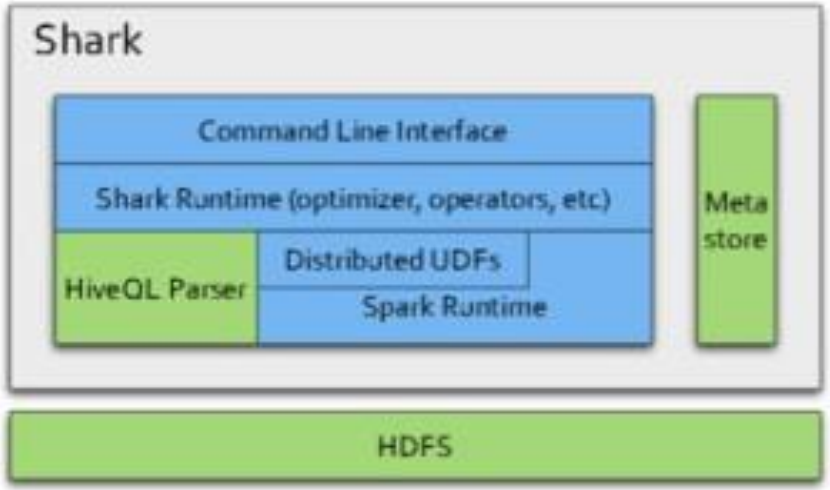


Figure 1: Shark Architecture

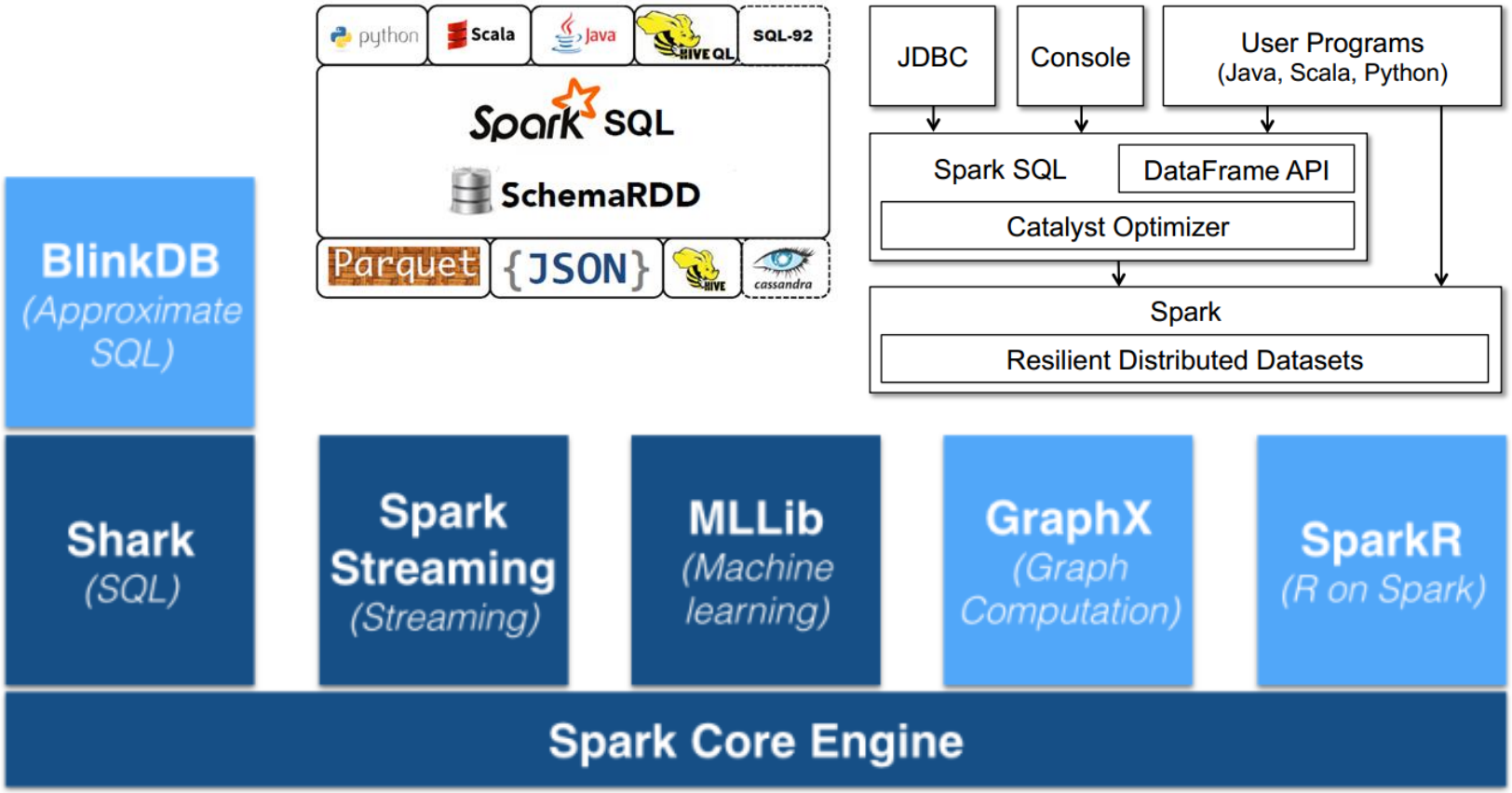
Spark SQL

A new SQL engine designed from ground-up for Spark

Hive on Spark

Help existing Hive users migrate to Spark

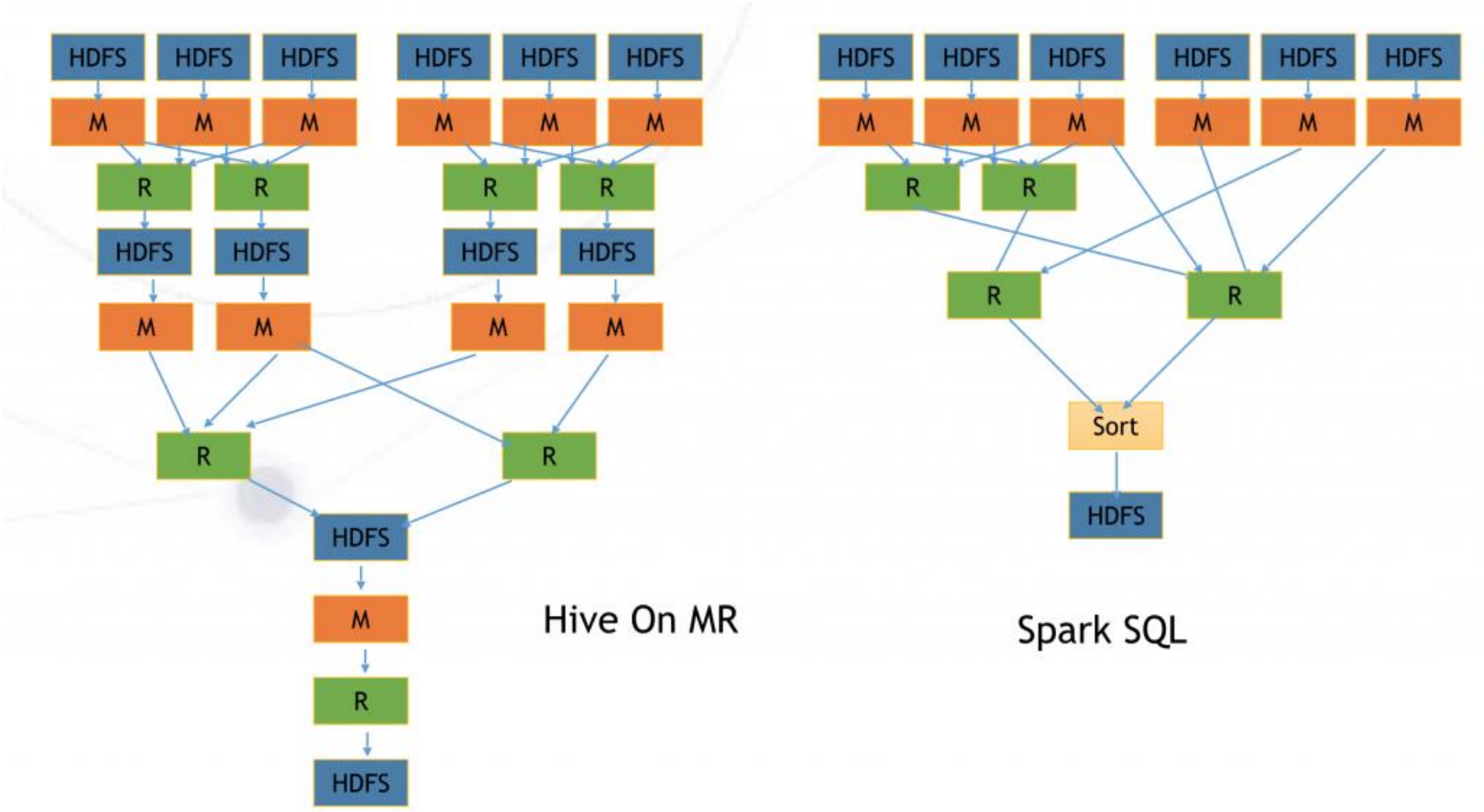
Spark-sql



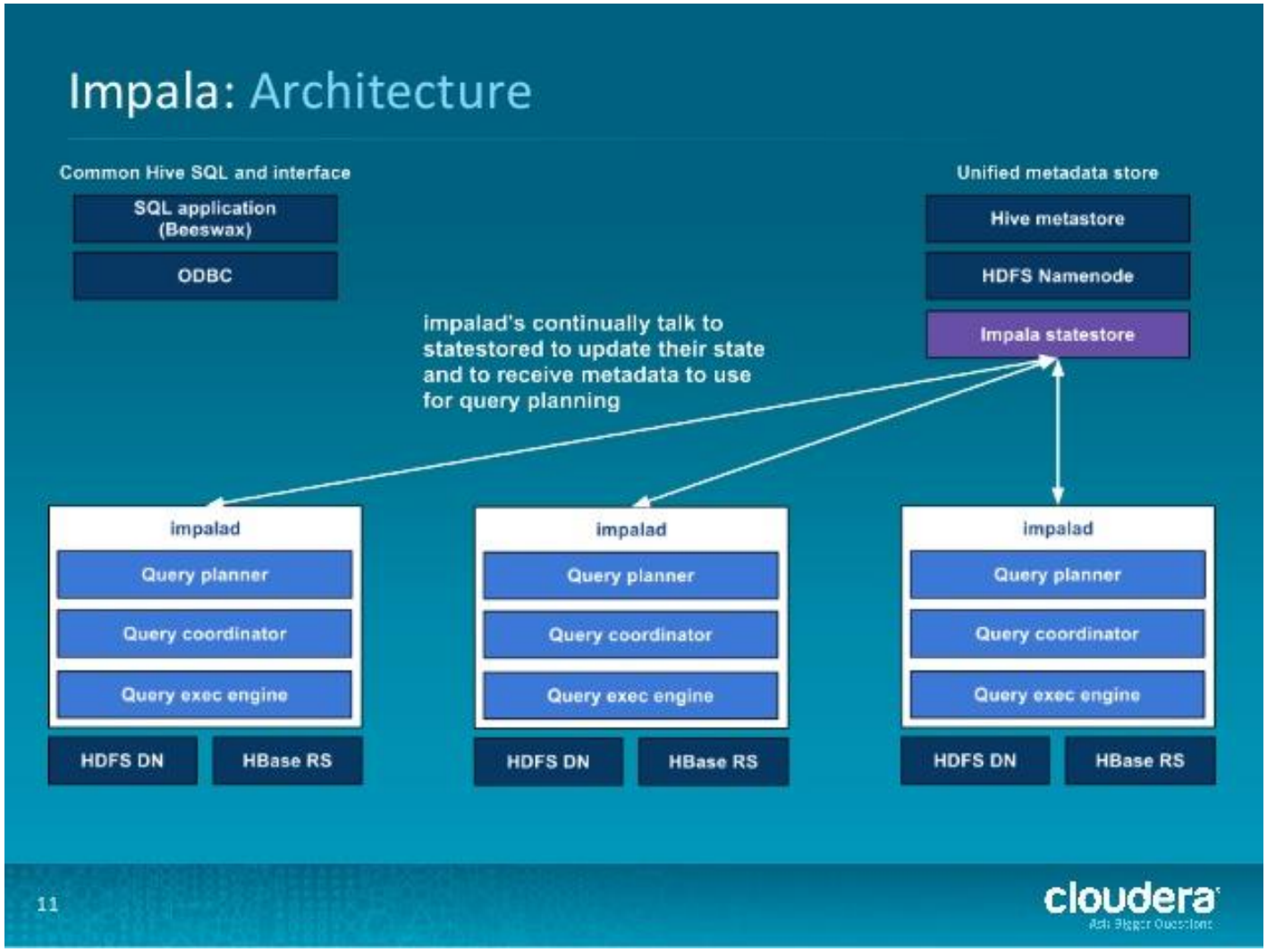
http://www.itweet.cn/2015/07/12/spark-manual/

http://www.itweet.cn/categories/spark/

Spark-SQL

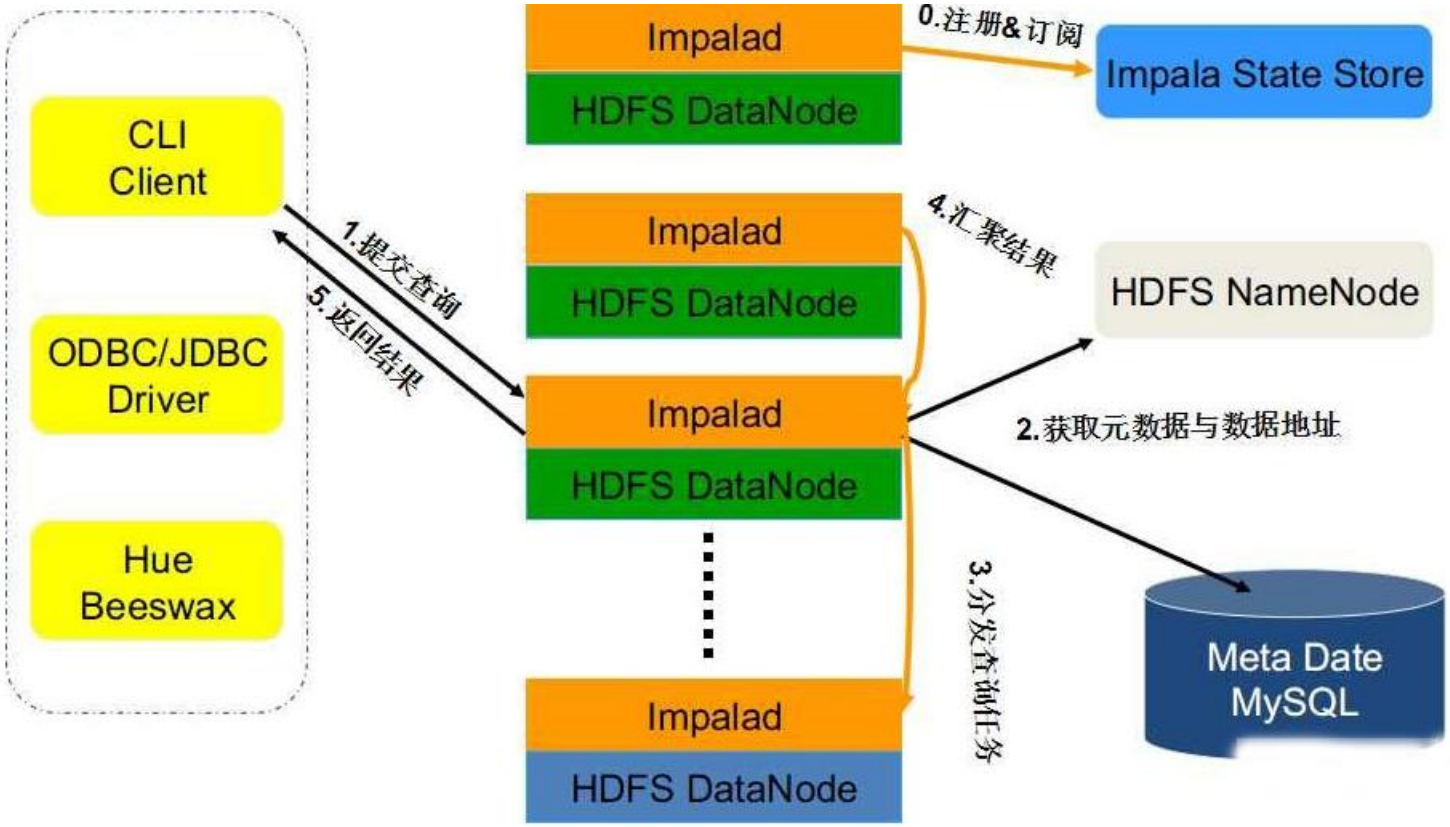


Impala



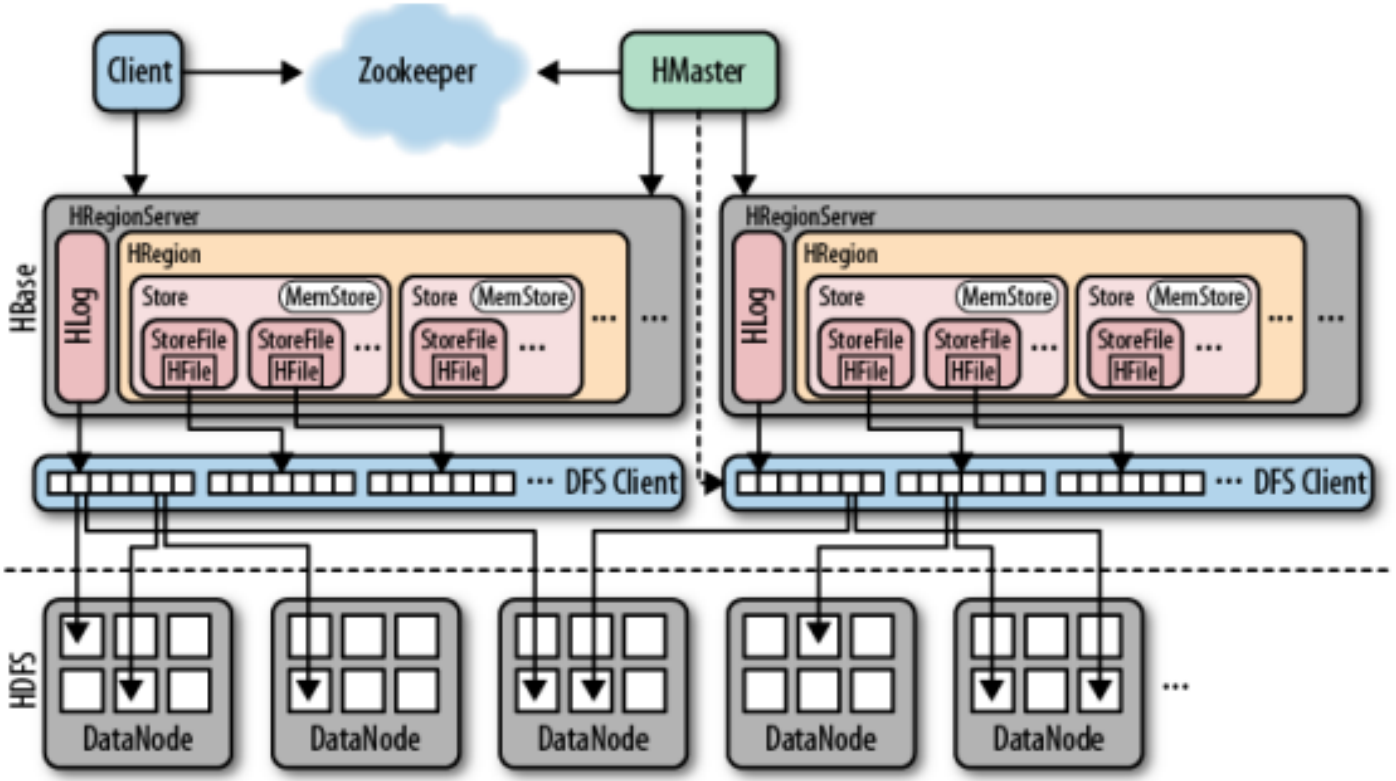
http://www.itweet.cn/2015/08/10/impala-develop/

Impala


CLI
Client
ODBC/JDBC
Driver
Hue
Beeswax
Impalad
HDFS DataNode
Impalad
HDFS DataNode
Impalad
HDFS DataNode
Impalad
HDFS DataNode
0.注册&订阅
Impala State Store
1.提交查询
5.返回结果
4.汇聚结果
HDFS NameNode
2.获取元数据与数据地址
3.分发查询任务
Meta Date
MySQL

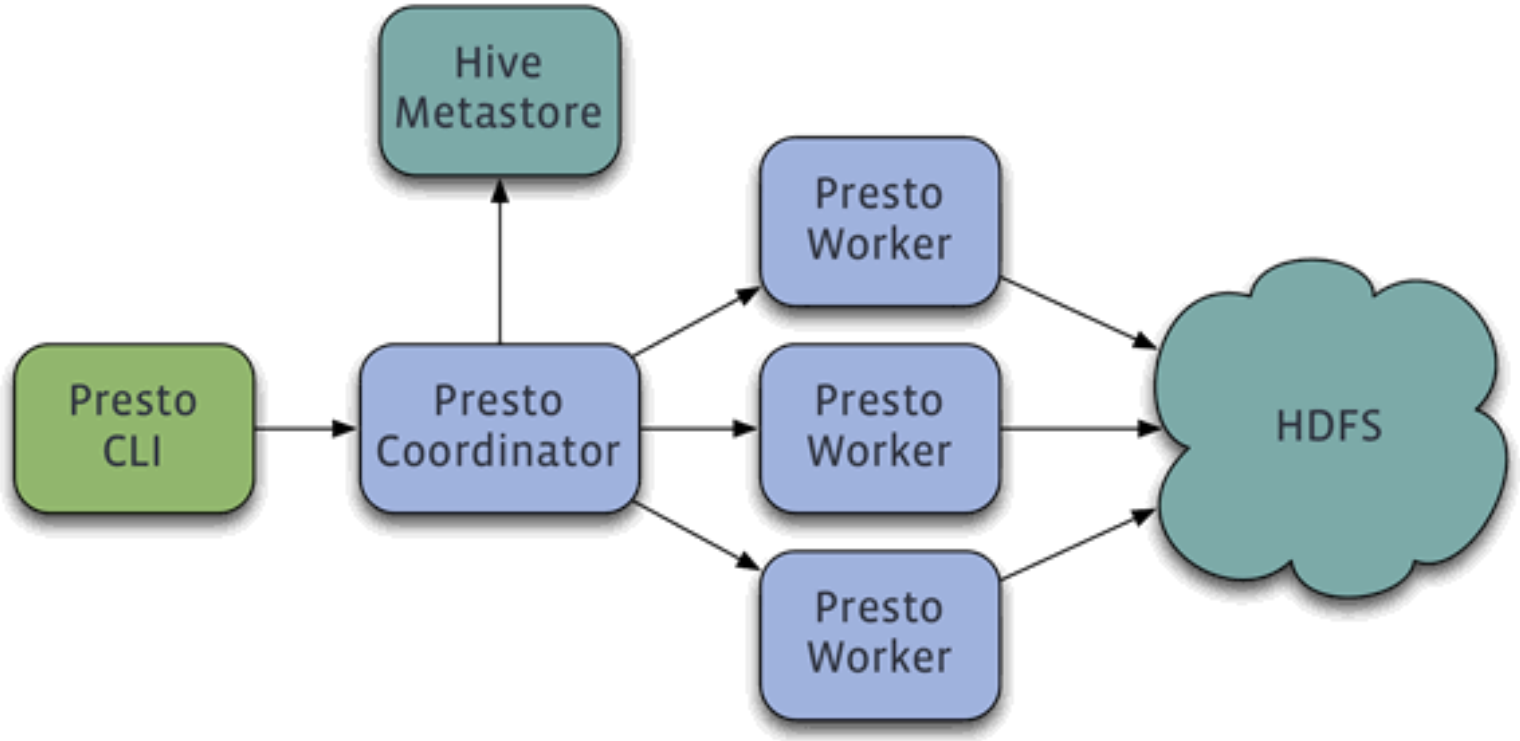
Phoenix(sql on hbase)

- Phoenix实现类sql查询Nosql数据库
- Phoenix将Query Plan直接使用HBaseAPI实现
- SQL语句解析为hbase查询语法



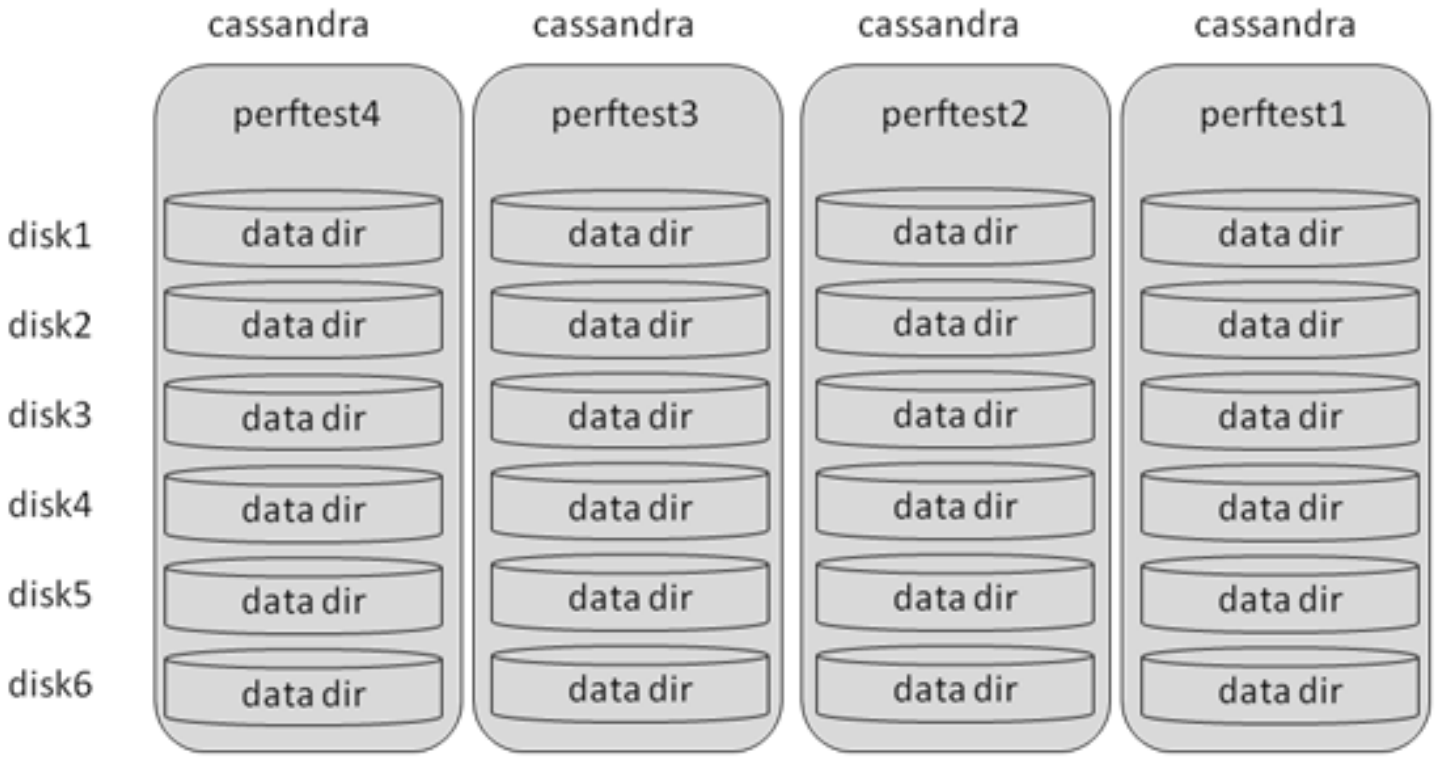
http://www.itweet.cn/2015/07/16/Phoenix-hbase/

Presto



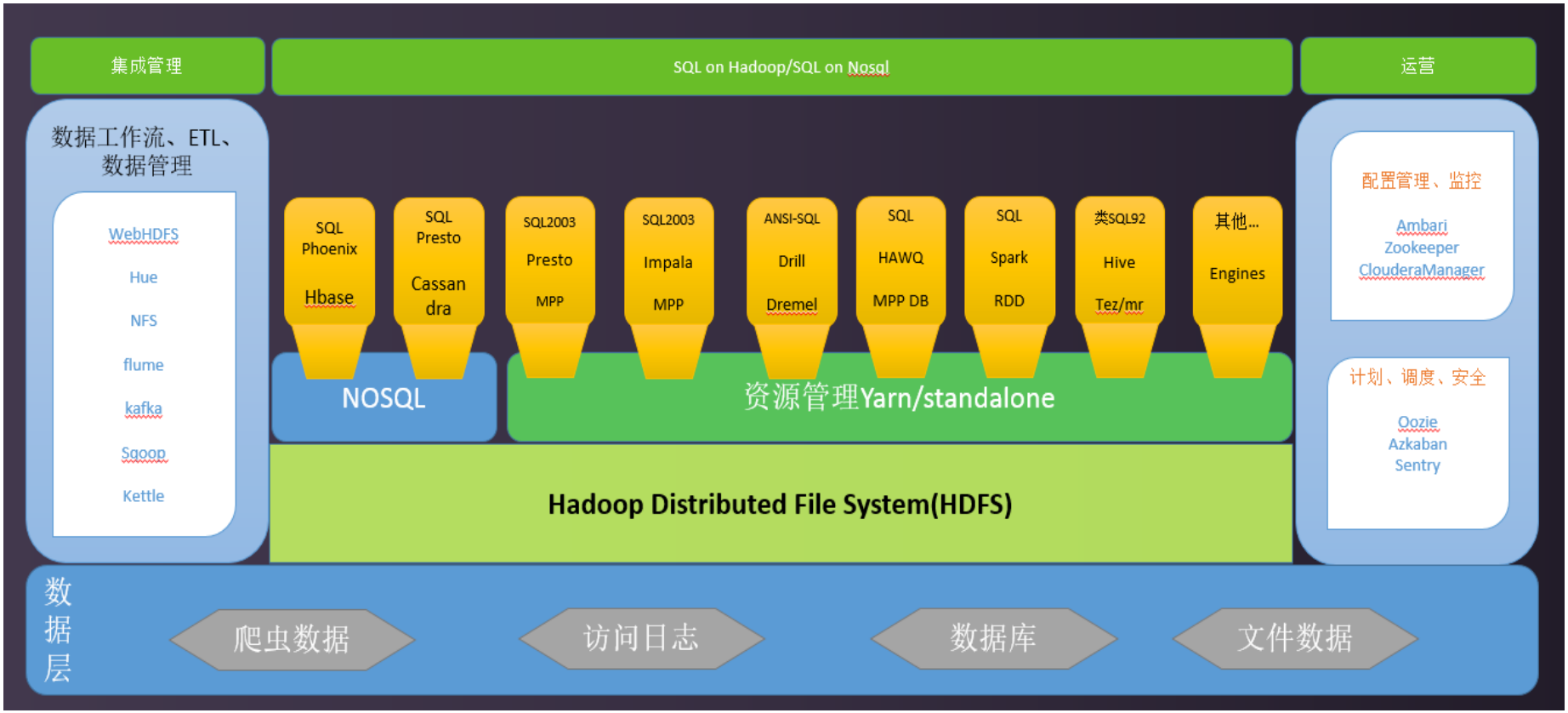
http://www.itweet.cn/2015/07/21/presto-use/

Cassandra



http://www.itweet.cn/2015/09/21/Apache-cassandra-cluster/

X on Yarn


集成管理
SQL on Hadoop/SQL on Nosql
运营
数据工作流、ETL、
数据管理
WebHDFS
Hue
NFS
flume
kafka
Sqoop
Kettle
SQL
Phoenix
Hbase
SQL
Presto
Cassandra
SQL2003
Presto
MPP
SQL2003
Impala
MPP
ANSI-SQL
Drill
Dremel
SQL
HAWQ
MPP DB
SQL
Spark
RDD
类SQL92
Hive
Tez/mr
其他...
Engines
NOSQL
资源管理Yarn/standalone
Hadoop Distributed File System(HDFS)
配置管理、监控
Ambari
Zookeeper
ClouderaManager
计划、调度、安全
Oozie
Azkaban
Sentry
数据层
爬虫数据
访问日志
数据库
文件数据

Parquet,textfile性能？

Parquet表(亿级,5个维度group by)

组件	耗时	资源消耗	其它问题记录
Hive	176.491s	3G	第一次执行
Impala	4.899s	5.4 GiB	第一次执行
Impala	4.584s	5 GiB	第二次执行
Spark-sql	9.92s	3.5 MB	第一次执行
Spark-sql	5.009s	3.5 MB	第二次执行

Text表(亿级, ,5个维度group by)

组件	耗时	资源消耗	其它问题记录
Hive	808.005s	2G	第一次执行
Impala	240.829s	2.7 G	第一次执行
Impala	238.893s	2.4 G	第二次执行
Spark-sql	318.5s	9.1 MB	第一次执行
Spark-sql	291.191s	8.9 MB	第二次执行

集群配置：mem:32g，cpu:32core，软件版本：saprk1.2,impala2.2,hive0.13.1,hadoop2.4

注意：这里的资源消耗情况并不准确，是根据个人想法填写，仅作参考！

缓存表？

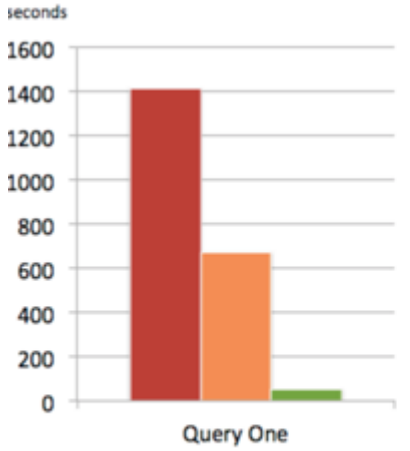
parksql cache table(亿级, ,5个维度group by)

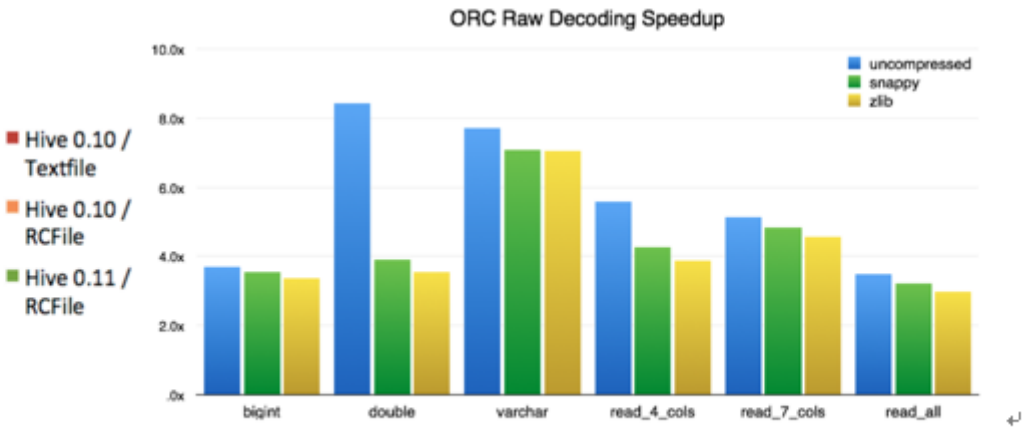
组件	耗时	资源消耗	其它问题记录
Hive	154.805s	5GB	第一次执行
Impala	128.204s	9.6 GiB	第一次执行
Impala	365.958s	9.3 GiB	第二次执行
Spark-sql[cache]	15.602s	9.5 MB	第一次执行
Spark-sql[cache]	7.707s	9.5 MB	第二次执行
Spark-sql[nocache]	32.382s	9.5 MB	第一次执行
Spark-sql[nocache]	32.941s	9.5MB	第二次执行

注意：这里的资源消耗情况并不准确，是根据个人想法填写，仅作参考！

文件格式

组件	格式	数据量	数据大小	不支持	支持
Imapla	Parquet	1118681226	30.1 G	Orcfile	Rcfile
Sparksql	Parquet	1118681226		无	全
Hive	Rcfile	1118681226	93.4 G	无	全
Presto	Orcfile	1118681226	16.2 G	无	全
Sparksql	Textfile	1118681226		无	全





文件格式

组件	格式	数据量	压缩大小	原始大小	压缩比例
Impala	Parquet		30.1G	98.6G	69.4 %
SparkSQL	Parquet		69.4G	98.6G	29.6 %
Hive	Rcfile		93.4G	98.6G	5.7 %
Presto	Orcfile		16.2G	98.6G	83.5 %
Hbase	Snappy		0.35 T	2.3 T	84.8 %

Hadoop压缩算法选择：

- mapreduce.map.output.compress.codec
- mapreduce.output.fileoutputformat.compress.codec
- mapreduce.output.fileoutputformat.compress.type
  - org.apache.hadoop.io.compress.DefaultCodec
  - org.apache.hadoop.io.compress.SnappyCodec [最佳选择]
  - org.apache.hadoop.io.compress.BZip2Codec /GzipCodec【GzipCodec压缩最高，但是时间上比较耗时】

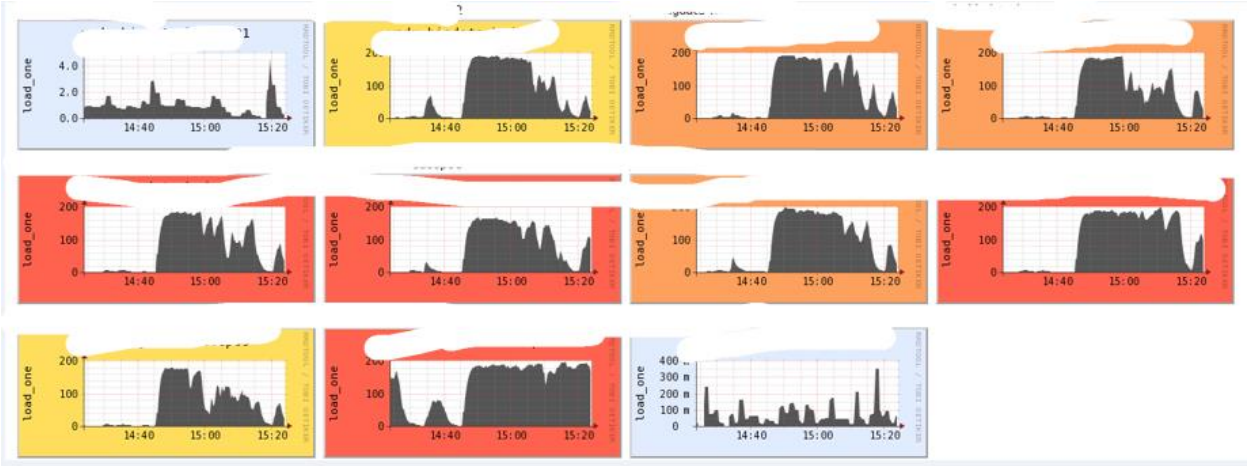
http://www.itweet.cn/2016/01/25/Hadoop-Disk-Planning/

ORC && Parquet比较

	Paruqet	ORC
现状	Apache顶级项目,开源，列式存储引擎	
开发语言	java	Java
主导公司	Twitter/Cloudera	Hortonworks
列编码	支持数据的词典编码、交错编码和RLE编码	支持主流编码
嵌套格式	支持比较完美	多层级嵌套表达起来复杂
ACID	不支持	支持
Update操作(delete,update等)	不支持	支持
支持索引(column列统计信息)	粗粒度索引	粗粒度索引
查询性能	不同的框架对列式存储格式有不同的优化，impala-parquet,prestodb-orc	
数据压缩	看图	
支持的查询引擎	Apache Drill/Impala/Spark	Apache Hive/PrestoDB

Hive格式转换？

Hive生成各种格式表	time	Filesize	问题记录
Rcfile	689.045s,	93.4 G	
Orcfile	1566.637s	30.1 G	
Parquetfile	1251.149s	68.1G	
Textfile	990.122s	98.6 G	



生成各种文件格式严重消耗集群资源，如果为parquet格式生成情况！

格式转换?

Impala生成各种格式表	Time	生成Filesize	问题记录
Rcfile			not support (RCFILE) file format. Supported formats are: (PARQUET, TEXTFILE)
Orcfile			
Parquetfile	8.84m	30.1 G	
Textfile	未测试	98.6 G	默认textfile

Presto生成各种格式表	Time	生成Filesize	问题记录
Rcfile	14:54m	93.5 G	task_writer_count='8';
Orcfile	61:00m	16.2 G	未加调优参数
Parquetfile	17.59m	82.2 G	未加调优参数
Textfile	未测试	98.6 G	默认presto就是rcfile格式

Sparksql生成各种格式表	Time	生成Filesize	问题记录
Rcfile	4.3m	93.4 G	
Orcfile	4.4 m	30.1 G	Orc压缩比例最高
Parquetfile	10.4m	69.4 G	生成part-r-00001.parquet
Textfile	未测试	98.6 G	默认就是textfile格式

性能_join

组件	耗时	资源消耗	其它问题记录	文件大小	数据量
Hive	1600s	内存：8g		93.4 G 文件为hive生成的rcfile表 dpi_rcfile	1118681226 262684
Presto	700s	内存：10g			
Impala	1175.29s	内存：230.4 GiB			
Sparksql	689.047s	内存：8.6 GB			
Sparksql[chache]		内存：?	效率提升不明显,未测试		

组件	耗时	资源消耗	其它问题记录	文件大小	数据量
Hive	300s	内存：8g		93.4 G 文件为hive生成的rcfile表 dpi_rcfile	1118681226 262684
Presto	60s	内存：100M			
Impala	2.67s	内存：10g			
Sparksql	40 s	内存：7.0 MB			
Sparksql[chache]		内存：?	效率提升不明显,未测试		

问题？

- 生成各种不同文件格式时间换空间
  -orc,rcfile,parquet,textfile
  -文件格式转换效率确实不高

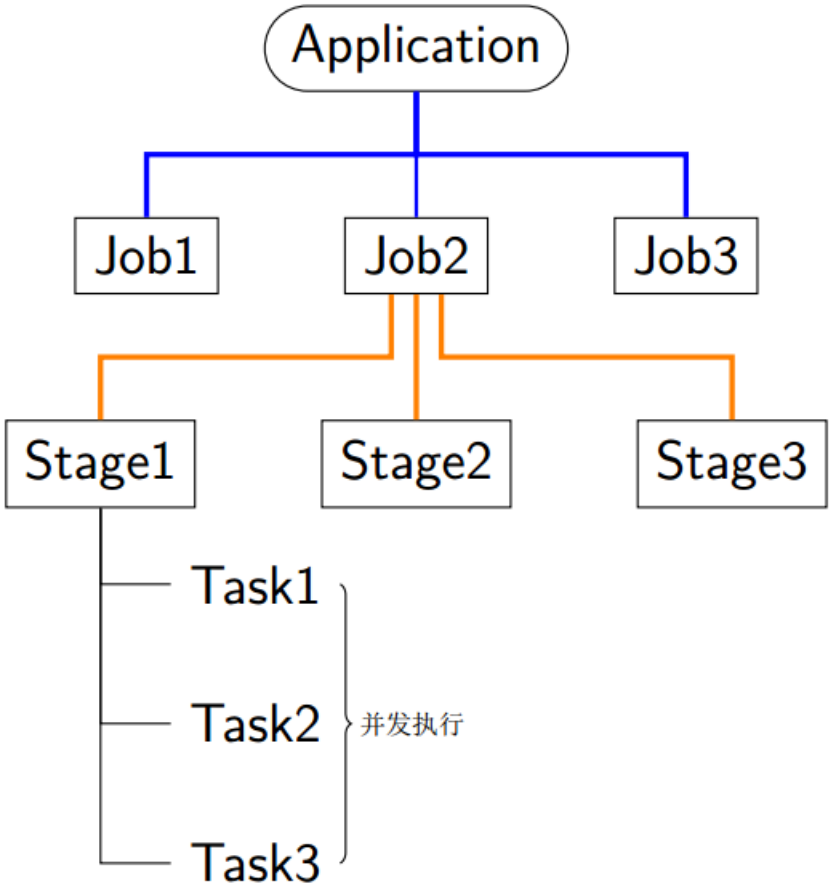
- 各种oom, overcommit
  - Yarn资源调优
  -Container xxx is running beyond physical memory limits
  - Java heap space(YARN)

- 都是吃内存大户,内存是个大问题！
  -各种列式存储格式超级耗内存
  -impala入门级mem:128g+

- 性能调优,调错参数,整个系统奔溃！
  -一步错,满盘皆输
  -运维需要有敬畏之心,充分验证理解各种调养参数,不如不如不调,选择绕过！

About performance-spark

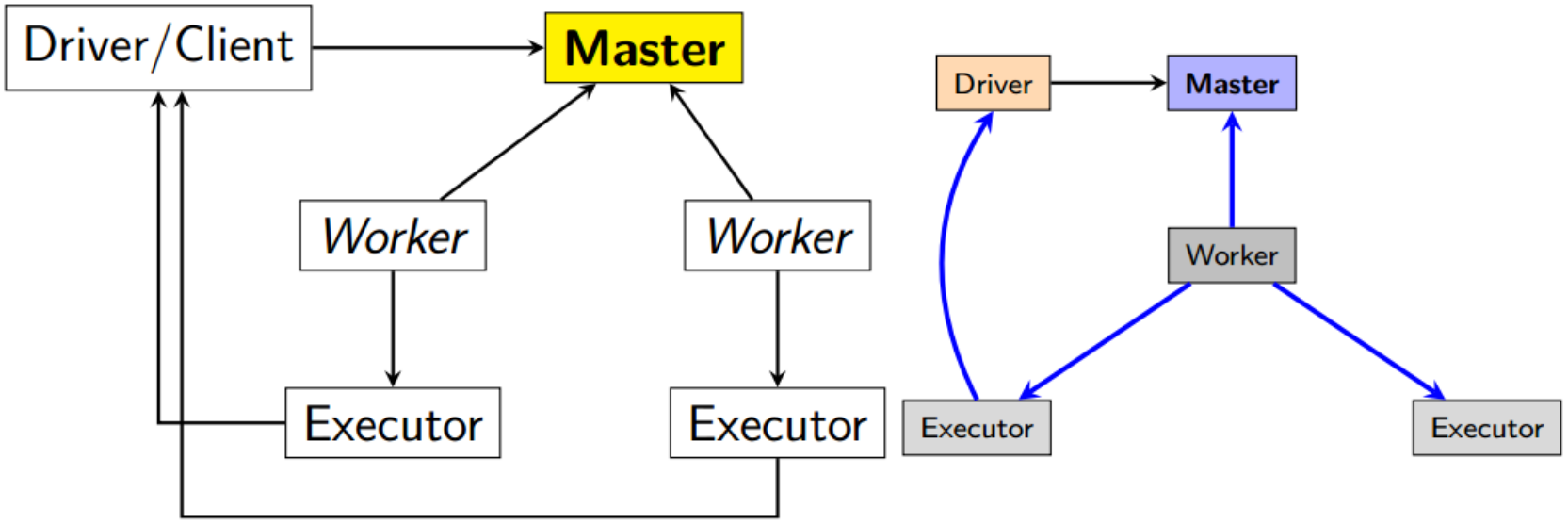
Application 规划阶段



Task 数目

每个 Stage 中 Task 数目取决于读入数据的 Partition 数，而真正可以并行处理的 Task 数目则取决于可用的计算资源，即 CPU Core 数目

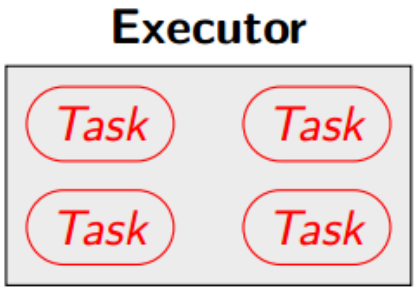
Standalone 集群组成



Task 到系统线程的映射关系

任务到线程的映射

1. Task 运行于 Executor 所在的 JVM 进程,Task 运行在具体的线程当中
2. 在同一个 Executor 中并行执行的 Task 个数取决于 Worker 上报给 Master 时声明的 Core 数目



spark-env.sh

假设要在一台有 24 核的机器上启动 4 个 worker

```
export SPARK_WORKER_INSTANCES=4
export SPARK_WORKER_CORES=6
```

两者相乘的结果不要超过 CPU 的物理核数 WORKER_INSTANCES * WORKER_CORES < PHYSICAL_CORES

Case1-广播变量

- 问题: Task序列化后太大
- 解决: 使用广播变量

  -当有小表时使用broadcast Join代替Common Join

  -Spark0.9开始序列化大小超过100K就会警告信息啦!

Case 2-filter

- 问题: val rdd = data.filter(f1).filter(f2).reduceBy…
  经过以上语句会有很多空任务或者小任务

- 解决: 使用coalesce或者repartition去减少RDD中partition数量

-coalesce与repartition的关系是什么？有什么异同？

Case 3-数据倾斜

- 问题: 任务执行速度倾斜

Aggregated Metrics by Executor

Executor ID	Address
1	133.37.135.213:40513
4	133.37.135.213:48329

- 解决:
  1、数据倾斜(一般是partition key取的不好)
  考虑其它的并行处理方式 中间可以加入一步aggregation

  2、 Worker倾斜(在某些worker上的executor不给力)
  设置spark.speculation=true 把那些持续不给力的node去掉

  -对比Hadoop MapReduce的speculation

Case 4-shuffle dir

- 问题:不设置spark.local.dir 这是spark写shuffle输出的地方

- 解决: 设置一组磁盘
  spark.local.dir=/mn1/spark, /mnt2/spar, /mnt3/spark

  -增加IO即加快速度

Case 5-任务数

- 问题: reducer数量不合适(默认reduce数量)
- 解决: 需要按照实际情况调整
  太多的reducer,造成很多的小任务,以此产生很多启动任务的开销。
  太少的reducer,任务执行慢!

Case 6-慢

- 问题：collect输出大量结果慢
- 解决：直接输出到分布式文件系统

RDD的collect方法是一个action操作，作用是将RDD中的数据返回到一个数组中。可以看到，在此action中，会触发Spark上下文环境SparkContext中的runJob方法，这是一系列计算的起点。

```
1.  abstract class RDD[T: ClassTag](
2.      @transient private var sc: SparkContext,
3.      @transient private var deps: Seq[Dependency[_]]
4.    ) extends Serializable with Logging {
5.    //….
6.  /**
7.     * Return an array that contains all of the elements in this RDD.
8.     */
9.    def collect(): Array[T] = {
10.     val results = sc.runJob(this, (iter: Iterator[T]) => iter.toArray)
11.     Array.concat(results: _*)
12.   }
13. }
```

Case7-序列化

- 问题:序列化Spark默认使用JDK自带的ObjectOutputStream
  -优点:兼容性好
  -缺点：体积大，速度慢

- 解决: 使用Kryo serialization-兼容性差
  -减少不必要的心跳检测
  -消息包压缩后传输, 使用 KryoSerializer
  优点:体积小，速度快
  缺点：兼容性稍差

```
spark.master            spark://5.6.7.8:7077
spark.executor.memory   512m
spark.eventLog.enabled  true
spark.serializer        org.apache.spark.serializer.KryoSerializer
```

Case 8-本地性

- 问题: 本地性不给力
  数据的远近会极大影响 Spark 任务的执行速度，当前支持以下几种不同的数据位置偏好，在调度的时候尽可能采取就近执行。
  -process_local 数据在同一进程内
  -node_local 数据在同一台机器中
  -no_pref 数据位置的远近不影响任务执行性能
  -rack_local 同一机架
  -any 任意机器

```
private def getLocalityWait(level: TaskLocality.TaskLocality): Long = {
  val defaultWait = conf.get("spark.locality.wait", "3s")
  val localityWaitKey = level match {
    case TaskLocality.PROCESS_LOCAL => "spark.locality.wait.process"
    case TaskLocality.NODE_LOCAL => "spark.locality.wait.node"
    case TaskLocality.RACK_LOCAL => "spark.locality.wait.rack"
    case _ => null
  }
```

- 解决: 考虑调整spark.locality.wait的值
  -delay scheduling
  -如果要将任务尽可能的分布到不同机器上,可以调节 spark.locality.wait

```
// Process local is expected to be used ONLY within TaskSetManager for now.
val PROCESS_LOCAL, NODE_LOCAL, NO_PREF, RACK_LOCAL, ANY = Value
```

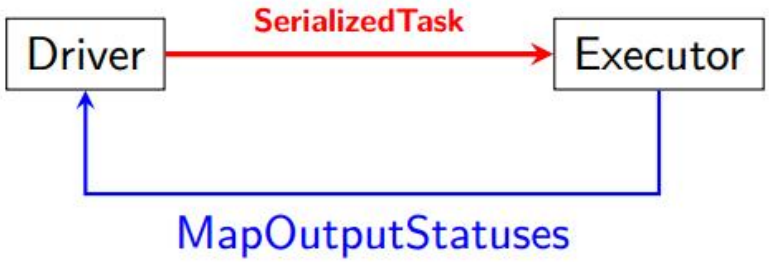
Case 9-OOM

- 问题: Task GC问题严重或者OOM
- 解决:调整spark.storage.memoryFraction的值
  -默认60%cach40%运行task
  -delay scheduling

spark.storage.memoryFraction	0.6	Fraction of Java heap to use for Spark's memory cache. This should not be larger than the "old" generation of objects in the JVM, which by default is given 0.6 of the heap, but you can increase it if you configure your own old generation size.

Case 10-序列化

spark.akka.framesize



消息类型

SerializedTask	Driver 需要将 Task 序列化后传递给 Executor
MapOutputStatuses	包含 Executor 将 Task 的执行结果，如果结果超过 framesize，将先存储在 BlockManager，然后 Driver 再从相应的 BlockManager 获取

问题重演

在 spark-defaults.conf 中将 *spark.akka.framesize* 设置为 1，单位为 M

```
val ll = 1L to 2L*1024L*1024L toList
sc.makeRDD(ll,1).collect
```

问题规避

尝试使用 broadcast

```
val ll = 1L to 2L*1024L*1024L toList
sc.broadcast(ll)
sc.makeRDD(ll,1).collect
```

方法 2：加大 spark.akka.framesize 的值

Case 11-临时文件

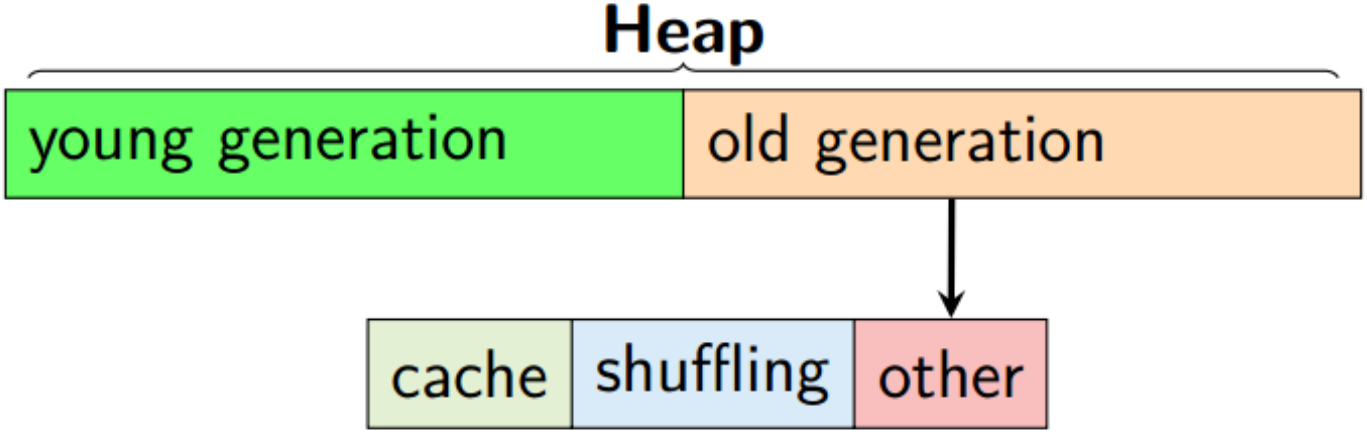
临时文件的存放和清理

SPARK_WORKER_DIR	存放依赖文件和 executor 执行生成的日志信息	$SPARK_HOME/work
SPARK_LOCAL_DIRS	存放 shuffle 和 RDD 缓存以及相应的第三方依赖包	/tmp

存放于 SPARK_LOCAL_DIRS 目录下的内容会在 Application 退出时被自动清除，对于那些需要长时间运行的程序，可以通过指定 spark.cleaner.ttl 来进行定时清理。这里存在一个问题，就是包含有第三方依赖的 cache 包不会被自动清除。

存放于 SPARK_WORKER_DIR 目录下的内容默认不会被自动清除，需要通过指定 spark.worker.cleanup.enabled 来进行定时清除。在定时清除时，只会将已经运行完成且超过指定时长的文件夹清理掉。

Case 12-内存



NewRatio 的默认值是 2, 意味着 Old Generation 可以占用 2/3 的 HeapSize

shuffle	spark.shuffle.memoryFraction	0.2
cache	spark.storage.memoryFraction	0.6

表中两者相加的总和不能超过 *Old Generation* 所能占用的最大比例

Case 13-调度策略

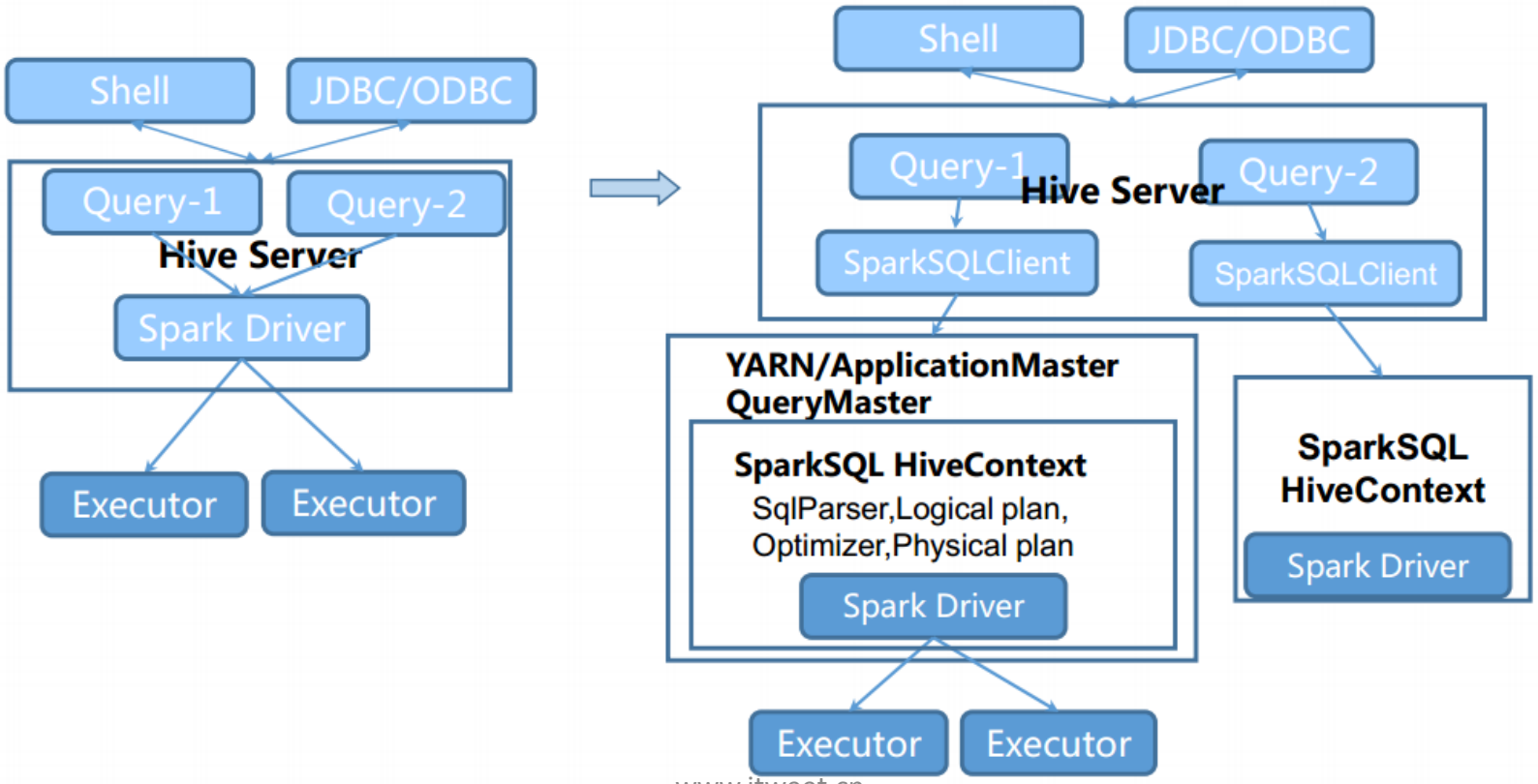
- 通过 *spark.scheduler.mode* 来设置调度策略
  > 通过多线程，同一个应用（application）可以提交多个作业（job），多个作业的调度策略有如下选择
  -FIFO 先入先出
  -FAIR 公平调度

- 避免单个应用占尽所有 CPU
  在 **spark-env.sh** 中设置 SPARK_MASTER_OPTS 为应用设定能获得的 CPU 个数的默认值
  export SPARK_MASTER_OPTS="-Dspark.deploy.defaultCores=12"
  应用程序可以在 **spark_defaults.conf** 中设置 *spark.cores.max* 来修改所要获取的最大核数

Case 14-Driver

为SparkSQL的每个Session分配一个Driver

问题: 一个HiveServer上所有的Query都使用一个Driver，导致query的内存和资源都会受到单个Driver的限制



最后

Spark1.4

经过4个RC版本，Spark 1.4最终还是赶在Spark Summit前发布了，本文简单谈下本版本中那些非常重要的新feature和improvement.

SparkR就不细说了，于data scientists而言，简直是望眼欲穿，千呼万唤始出来..... 这显然要用单独一篇文章来说下：)

Spark Core:

现在大家最关心什么？性能和运维呀！ 什么最影响性能？必须shuffle呀！什么是运维第一要务?必须是监控呀(就先不扯alert了)！1.4在这两点都做足了功夫。 1.4中，Spark为应用提供了REST API来获取各种信息(jobs / stages / tasks / storage info)，使用这个API搭建个自己的监控简直是分分钟的事情，不止于此，DAG现在也能可视化了，不清楚Spark的DAGScheduler怎么运作的同学，现在也能非常轻易地知道DAG细节了。再来说说shuffle, 大家都知道，从1.2开始sort-based shuffle已经成为默认的shuffe策略了，基于sort的shuffle不需要同时打开很多文件，并且也能减少中间文件的生成，但是带来的问题是在JVM的heap中留了大量的java对象，1.4开始，shuffle的map阶段的输出会被序列化，这会带来两个好处：1、spill到磁盘上的文件变小了 2、GC效率大增 ，有人又会说，序列化反序列化会产生额外的cpu开销啊，事实上，shuffle过程往往都是IO密集型的操作，带来的这点cpu开销，是可以接受。

大家期待的钨丝计划(Project Tungsten)也在1.4初露锋芒，引入了新的shuffle manager “UnsafeShuffleManager”, 来提供缓存友好的排序算法，及其它一些改进，目的是降低shuffle过程中的内存使用量，并且加速排序过程。 钨丝计划必定会成为接下来两个版本(1.5,1.6)重点关注的地方。

Spark Streaming：

Streaming在这个版本中增加了新的UI， 简直是Streaming用户的福音啊，各种详细信息尽收眼底。话说Spark中国峰会，TD当时坐我旁边review这部分的code，悄悄对说我”this is awesome”。对了，这部分主要是由朱诗雄做的，虽然诗雄在峰会上放了我鸽子，但必须感谢他给我们带来了这么好的特性！另外此版本也支持了0.8.2.x的Kafka版本。

Spark SQL(DataFrame)

支持老牌的ORCFile了，虽然比Parquet年轻，但是人家bug少啊：) 1.4提供了类似于Hive中的window function，还是比较实用的。本次对于join的优化还是比较给力的，特别是针对那种比较大的join，大家可以体会下。JDBC Server的用户肯定非常开心了，因为终于有UI可以看了呀。

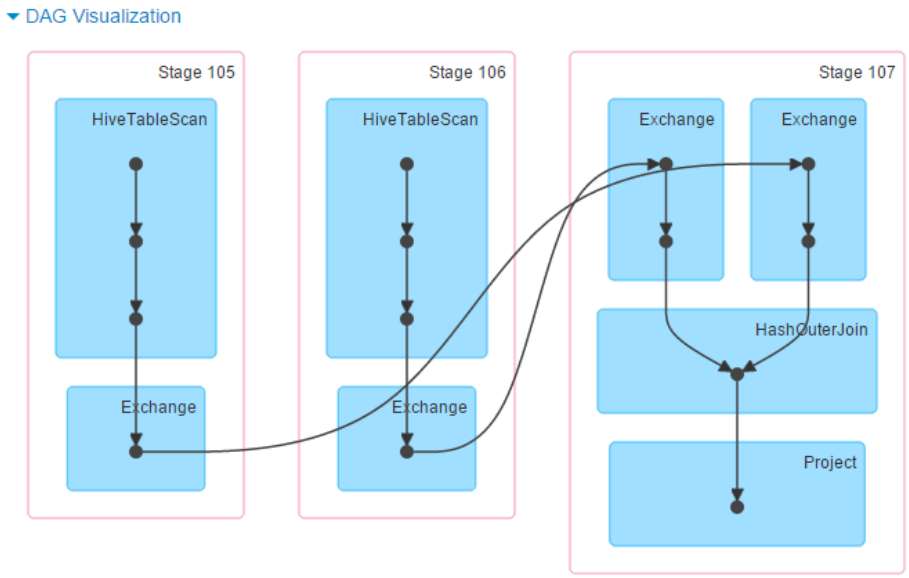
Spark ML/MLlib

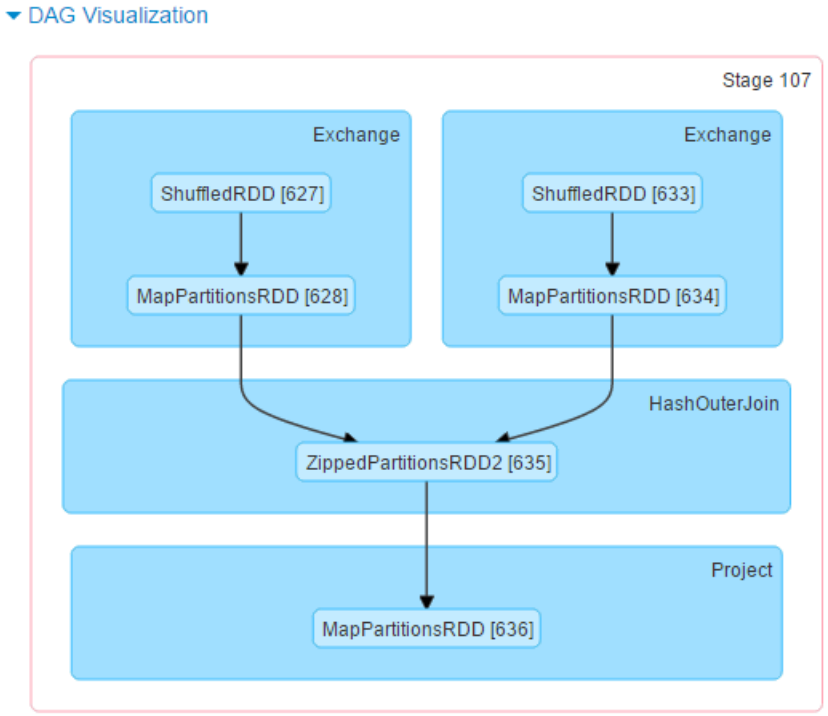
ML pipelines从alpha毕业了，大家对于ML pipelines的热情还真的蛮高的啊。我对Personalized PageRank with GraphX倒是蛮感兴趣的，与之相关的是recommendAll in matrix factorization model。 事实上大多数公司还是会在Spark上实现自己的算法。--来源,七牛云陈超

Ps:个人觉得sql模块还有很长的路要走...自己玩玩还行,用于生产挑战极大...

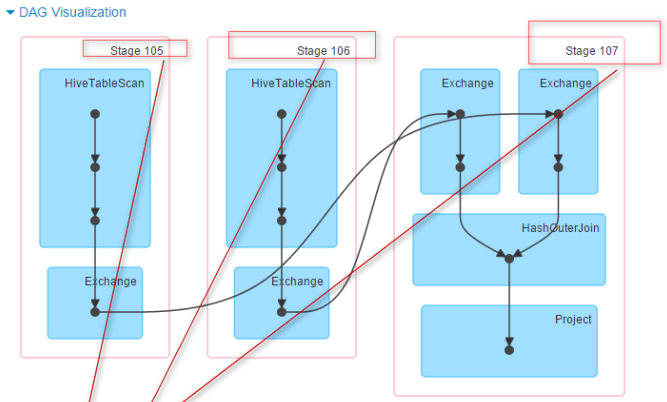
可视化-Event Timeline

可视化-DAG


DAG Visualization
Stage 105
HiveTableScan
Exchange
Stage 106
HiveTableScan
Exchange
Stage 107
Exchange
Exchange
HashOuterJoin
Project


DAG Visualization
Stage 107
Exchange
ShuffledRDD [627]
MapPartitionsRDD [628]
Exchange
ShuffledRDD [633]
MapPartitionsRDD [634]
HashOuterJoin
ZippedPartitionsRDD2 [635]
Project
MapPartitionsRDD [636]

可视化-DAG各阶段



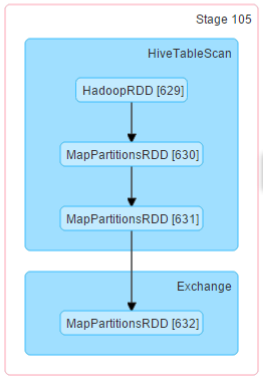
Completed Stages (3)

Stage Id	Description	Submitted	Duration	Tasks: Succeeded/Total	Input	Output	Shuffle Read	Shuffle Write
107	create table temp.temp_base_028_4 AS select a.*, f.pay_dt, f.pay_charge, f.max_charge, f.min_charge, f.mon3_paynum, f.mo… Spark JDBC Server Query +details	2015/06/14 07:26:41	2 s	200/200				
106	create table temp.temp_base_028_4 AS select a.*, f.pay_dt, f.pay_charge, f.max_charge, f.min_charge, f.mon3_paynum, f.mo… Spark JDBC Server Query +details	2015/06/14 07:26:40	0.6 s	200/200				
105	create table temp.temp_base_028_4 AS select a.*, f.pay_dt, f.pay_charge, f.max_charge, f.min_charge, f.mon3_paynum, f.mo… Spark JDBC Server Query +details	2015/06/14 07:26:40	0.3 s	200/200				

Details for Stage 105 (Attempt 0)

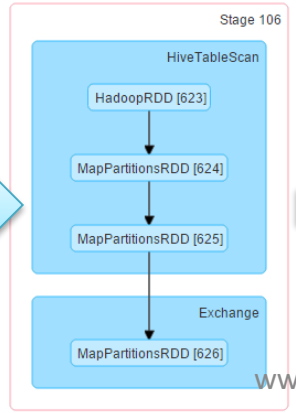
Total Time Across All Tasks: 5 s

DAG Visualization

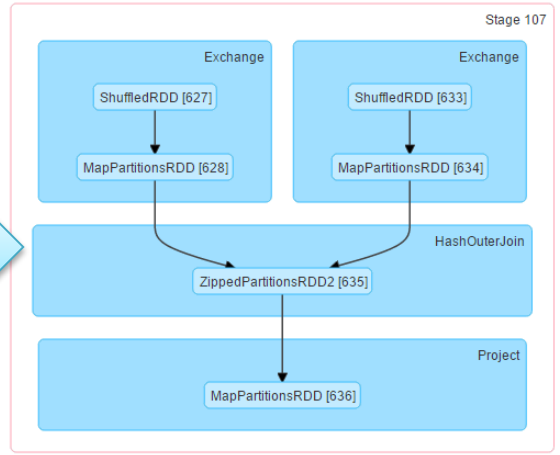


Total Time Across All Tasks: 4 s

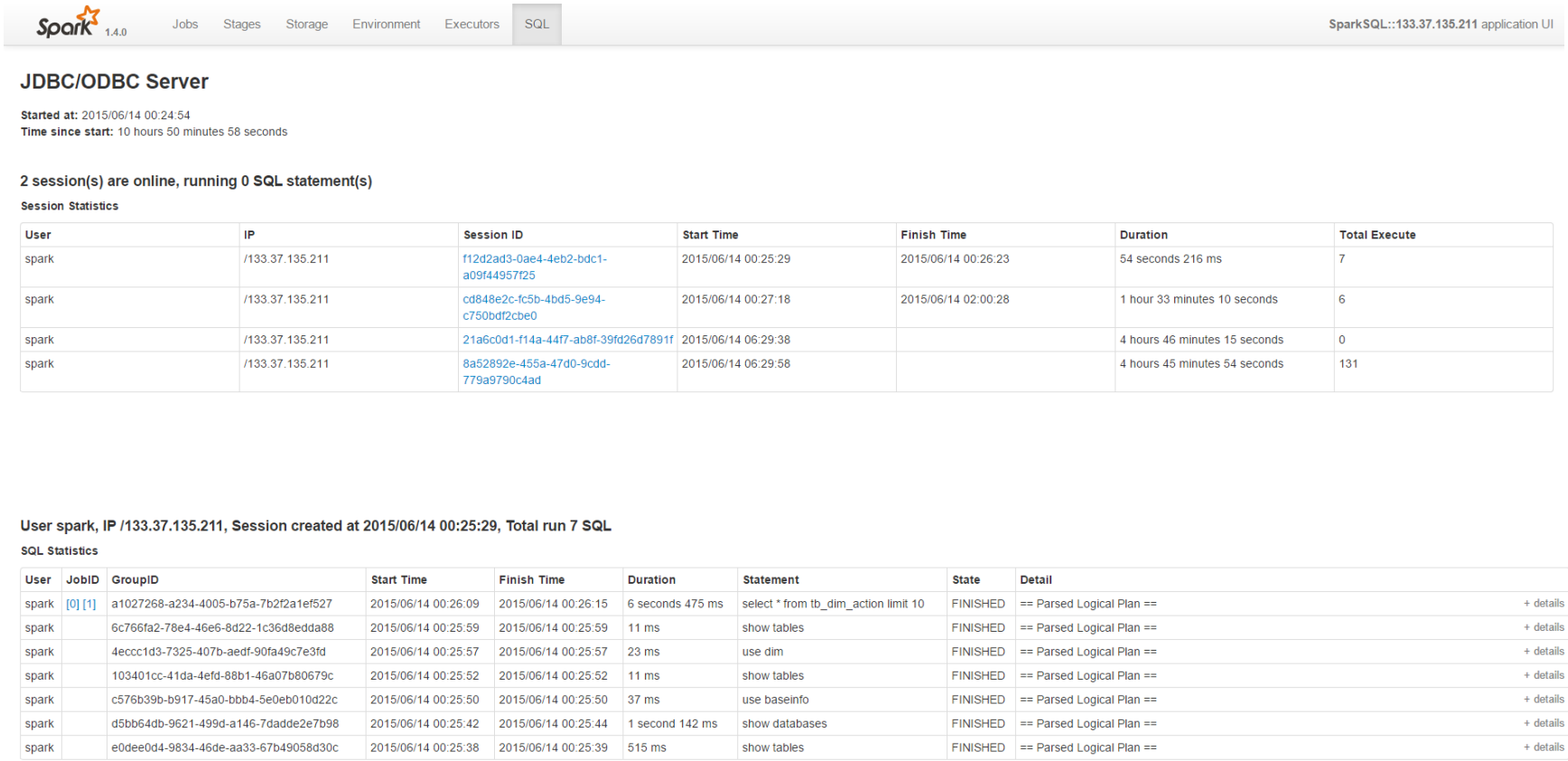
DAG Visualization



DAG Visualization



可视化-SparkSQL session



Spark 1.4.0 Jobs Stages Storage Environment Executors SQL SparkSQL::133.37.135.211 application UI

JDBC/ODBC Server

Started at: 2015/06/14 00:24:54
Time since start: 10 hours 50 minutes 58 seconds

2 session(s) are online, running 0 SQL statement(s)

Session Statistics

User	IP	Session ID	Start Time	Finish Time	Duration	Total Execute
spark	/133.37.135.211	f12d2ad3-0ae4-4eb2-bdc1-a09f44957f25	2015/06/14 00:25:29	2015/06/14 00:26:23	54 seconds 216 ms	7
spark	/133.37.135.211	cd848e2c-fc5b-4bd5-9e94-c750bdf2cbe0	2015/06/14 00:27:18	2015/06/14 02:00:28	1 hour 33 minutes 10 seconds	6
spark	/133.37.135.211	21a6c0d1-f14a-44f7-ab8f-39fd26d7891f	2015/06/14 06:29:38		4 hours 46 minutes 15 seconds	0
spark	/133.37.135.211	8a52892e-455a-47d0-9cdd-779a9790c4ad	2015/06/14 06:29:58		4 hours 45 minutes 54 seconds	131

User spark, IP /133.37.135.211, Session created at 2015/06/14 00:25:29, Total run 7 SQL

SQL Statistics

User	JobID	GroupID	Start Time	Finish Time	Duration	Statement	State	Detail
spark	[0] [1]	a1027268-a234-4005-b75a-7b2f2a1ef527	2015/06/14 00:26:09	2015/06/14 00:26:15	6 seconds 475 ms	select * from tb_dim_action limit 10	FINISHED	== Parsed Logical Plan == + details
spark		6c766fa2-78e4-46e6-8d22-1c36d8edda88	2015/06/14 00:25:59	2015/06/14 00:25:59	11 ms	show tables	FINISHED	== Parsed Logical Plan == + details
spark		4eccc1d3-7325-407b-aedf-90fa49c7e3fd	2015/06/14 00:25:57	2015/06/14 00:25:57	23 ms	use dim	FINISHED	== Parsed Logical Plan == + details
spark		103401cc-41da-4efd-88b1-46a07b80679c	2015/06/14 00:25:52	2015/06/14 00:25:52	11 ms	show tables	FINISHED	== Parsed Logical Plan == + details
spark		c576b39b-b917-45a0-bbb4-5e0eb010d22c	2015/06/14 00:25:50	2015/06/14 00:25:50	37 ms	use baseinfo	FINISHED	== Parsed Logical Plan == + details
spark		d5bb64db-9621-499d-a146-7dadde2e7b98	2015/06/14 00:25:42	2015/06/14 00:25:44	1 second 142 ms	show databases	FINISHED	== Parsed Logical Plan == + details
spark		e0dee0d4-9834-46de-aa33-67b49058d30c	2015/06/14 00:25:38	2015/06/14 00:25:39	515 ms	show tables	FINISHED	== Parsed Logical Plan == + details

SparkStraming

[hadoop@bigdatat02 ~]$ nc -lk 9999

hello

a

bc

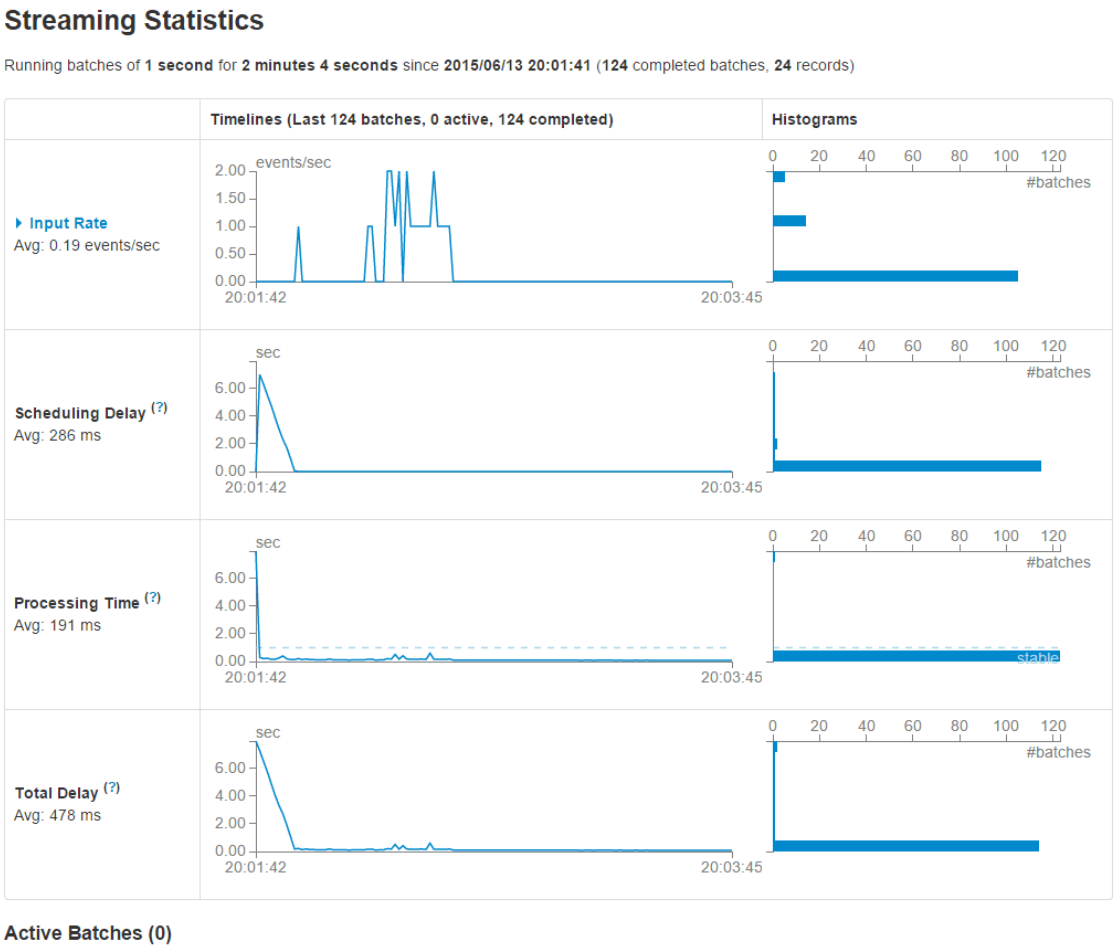
d

e

f

$ spark/bin/spark-submit --class com.sparkjvm.streaming.yarn.NetHdfsWordCount --master spark://bigdatat01:18888 sparkstreaming-NetHdfsWordCount-1.0-SNAPSHOT.jar bigdatat02 9999 10

可视化-SparkStreaming



性能Spark 1.4?

Spark1.4+版本开始支持窗口函数，Row_number action:create

Job Id (Job Group)	Description	Submitted	Duration	Stages: Succeeded/Total	Tasks (for all stages): Succeeded/Total
6 (f96d0650-d7c2-41cf-baa7-57f020ad2d0c)	create table default.test_1 as select month_no,date_no,mdn,row_number() over (partition by mdn,date_no or...	2015/06/14 23:20:32	5.2 min	2/2 (1 skipped)	232/232 (291 skipped)

Completed Stages (2)

Stage Id	Description	Submitted	Duration	Tasks: Succeeded/Total	Input	Output	Shuffle Read	Shuffle Write
14	create table default.test_1 as select month_no,date_no,mdn,row_number() over (partition by mdn,date_no order by keyword desc) as rn from tb_interface_keyword_table order by month_no,date_no Spark JDBC			32/32			978.9 MB	
13	create table default.test_1 as select month_no,date_no,mdn,row_number() over (partition by mdn,date_no order by keyword de... Spark JDBC Server Query +details	2015/06/14 23:20:32	50 s	200/200			3.0 GB	978.9 MB

- mdn=b.a_nbr

数据量：32.6G 表名：default.TEST_TABLE a ---150397539
数据量：774.6 MB 表名：src.F_1_D_2 b --894534
完成时间：22min 生成数据字段：120 生成数据：85.2 GB

Job Id (Job Group)	Description	Submitted	Duration	Stages: Succeeded/Total	Tasks (for all stages): Succeeded/Total
54 (e03ac224-345f-454a-863c-c181b8084416)	create table default.test as select a.date_no,a.cndomain,a.domain,a.url,a.keyword,a.mdn,b.* from default.T... Spark JDBC Server Query	2015/06/14 17:48:55	22 min	3/3	512/512

```
0: jdbc:hive2://bigdatat01:10000> create table default.test as select a.date_no,a.cndomain,a.domain,a.url,a.keyword,a.mdn,b.* from default.TB_INTERFACE
_KEYWORD_TABLE a left outer join src_eda.F_1_SERV_D_2 b on a.mdn=b.acc_nbr;
+---------+
| Result  |
+---------+
+---------+
No rows selected (1319.43 seconds)
0: jdbc:hive2://bigdatat01:10000> select count(1) from default.test;
+------------+
|    _c0     |
+------------+
| 151135744  |
+------------+
1 row selected (221.374 seconds)
```


测试库：mem:24GB Cores:24

结论

Impala2.0

- 1．目前最好的SQL on hadoop解决方案，单纯的sql分析海量数据框架，效率最高！
- 2．Impala做海量数据分析容易出现内存不足！
- 3．内存评估非常不合理，通常非常小的数据，可以用出几倍的内存！
- 4．如果内存不足走了磁盘，那么效率并不会比其他sql on hadoop效率高！
- 5．Sql语法支持度全，join能力也强，这一点是需要称赞的！
- 6. 不需要有专门的人员去优化，本身效率就很高！当然Mem永远是甩不开的问题！

Spark 1.3.1

- 1．spark在group by,dictinct,join等操作会导致大量shuffle,而影响性能,这个需要调优！
- 2．使用spark有利于一个软件栈解决所有大数据问题！(流计算,离线分析,机器学习,数据挖掘(sparkR),图计算等…)
- 3．大数据应用到最后都是在挖掘机器学习等,而saprk就是为此而生,可以为后续引入更加复杂算法提供便利！
- 4．支持和RDBMS,NOSQL数据库创建映射表进行关联,支持缓存到内存！
- 5．有Mllib，Streaming，graphX等让应用更加智能化的模块！
- 6．更加亲民的Sspark-sql模块,也是目前应用最广泛的,在1.4即将发布版本支持窗口分析函数而变得更加强大！
- 7．选择spark不是仅仅一个框架而是一个大数据计算生态系，是一个体系，而不像其他sql on hadoop解决方案,仅仅是一个框架而已！即它与众不同！
- 8．Sparksql的分布式缓存功能并不优秀,数据量太大会导致一些莫名其妙的问题，比如执行groupby,join等需要的shuffle操作，效率提升和没缓存效果没差别,因为会对数据重新调整顺序,我感觉他是内存数据移动！所以tachyon分布式内存文件系统就是为此而生！tachyon商业公司已经获取7500万美元融资，相信发展会越来越好！目前tachyon和spark结合可以做到中间数据写到tachyon供不同应用程序间共享数据！数据放到tachyon可以减少大量GC问题！
- 9. 如果没有非常优秀的团队维护，要用好非常困难，版本更新快，join，shuffle永远是大问题，全靠优化！需要专业人员随时跟进社区发展！
- 10. spark 1.3.0坑，升级有风险(低)，对于计算框架来说，特别spark框架升级不会超过10分钟就完成升级和回滚！
  1.3.0读取parquet表会报错！Parquet data source may use wrong Hadoop FileSystem (SPARK-6330)！1.3.1修复！
  Error: java.lang.IllegalArgumentException: Wrong FS: hdfs://hadoop01:8020/user/hive/warehouse/test.db/parquet_test, expected: f ile:/// (state=,code=0)
- 11. 目前社区对jdbc一块,数据仓库，BI,ETL投入不是很大，感觉发展方向也是有所调整，sparksql做sql解决方案还是有点牵强，不宜用！Jdbc经常崩溃！
- 12.如果做挖掘机器学习,sparkR等建议使用,sql分析一块建议presto,impala等！

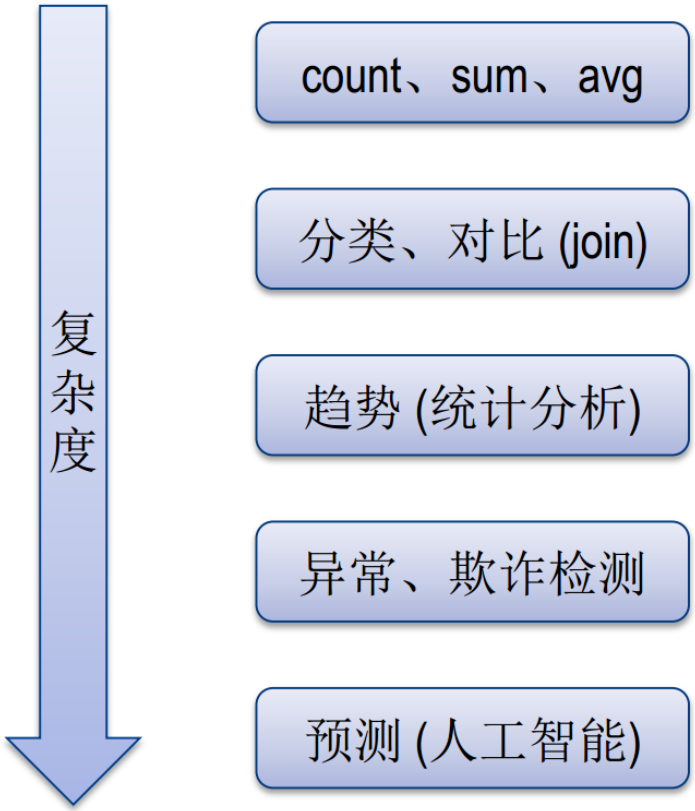
Hive

- 1．hive是目前大数据应用必备的sql on hadoop解决方案,发展长久，基于MR非常稳定！
- 2．Hive基于MR一大特点就是慢！
- 2．海量数据离线分析还得靠Hive来做！稳定就是花费很长时间！
- 3．想要用好Hive确实需要话一定时间在优化上面，特别是Yarn的优化！
- 4．Hive使用出现oom,overcommit,java heap space等问题！都需要优化！
- 5．Hive0.14支持update,delete等操作！功能越来越健全！开源产品很多细节的东西并不完善！
- 6．更多细节需要使用者自己完善加以修饰！

Presto

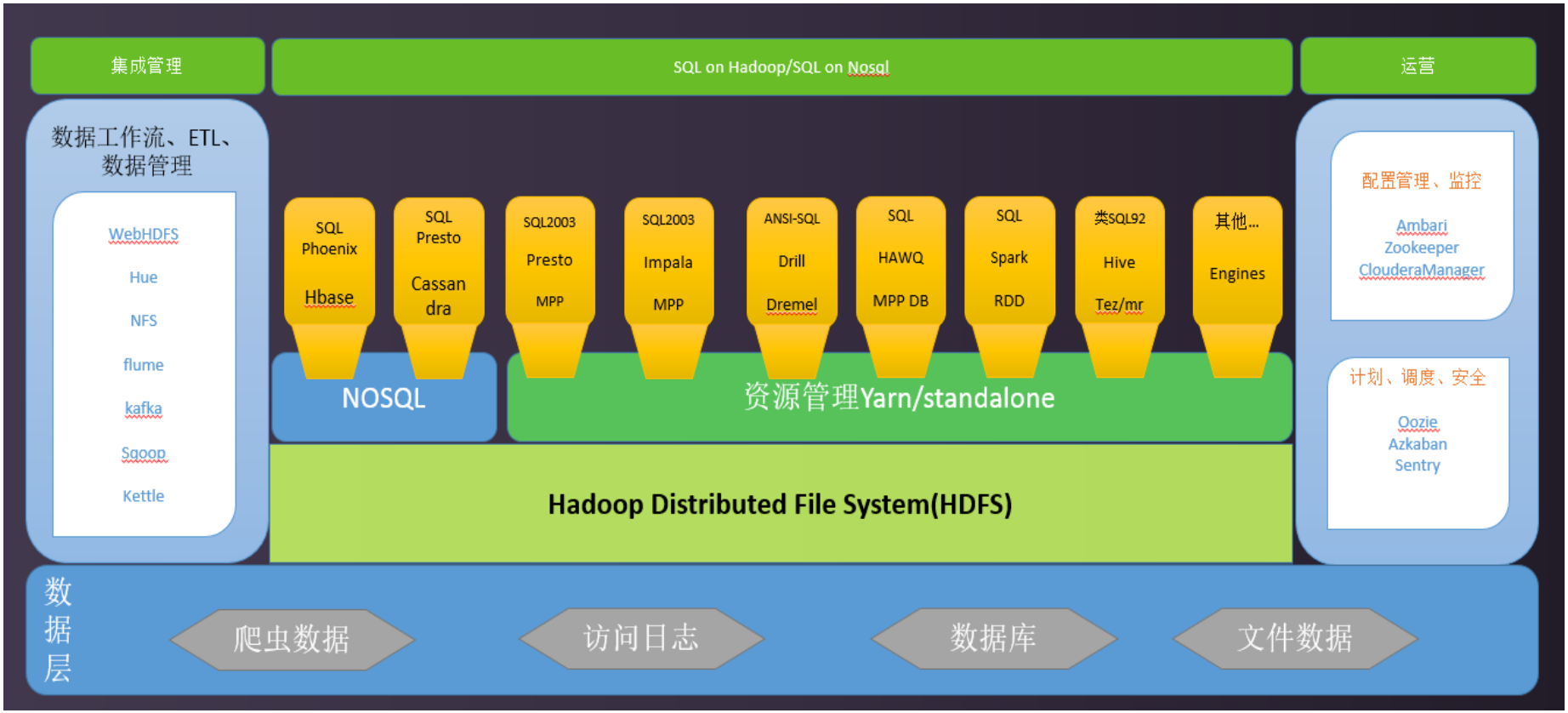
- 1．支持的语法比较全面
- 2. 用好全靠自己调节参数和经验
- 3. 主要还是在维护上面需要有人专门跟踪开源社区发展，不然遇到问题难解决资料少
- 4. 稳定性和可靠性经过Facebook海量数据挑战，基本无需顾虑
- 5. 目前算是hive一个比较好的替代品，还是需要更多应用场景来检测！
- 6. 目前国内使用情况，JD,美团等公司在大量使用！
- 7. 没有缓存功能，是优点也是缺点！后续版本会增加缓存功能
- 8. 能够和RDBMS(mysql,oracle,sql server, postgresql等)和NOSQL(Hbase, Cassandra)直接关联！
- 9. 支持mysql(支持缓存)分表分库关联查询,直接hive数据和RDBMS,NOSQL直接关联查询！
- 10. 某些特定分析函数效用还有待优化，优势并不比其他sql on hadoop解决方案优秀！
- 11. 在查询数据超越内存大小后直接保存内存不足！
- 12. 可以直接和kafka这样的消息队列直接对接实现实时流计算目的！
- 13． Mlib模块目前还在开发中，并没有实际使用！
- 14. 面对更大量数据,presto更加有优势，效率比其他sql on hadoop方案更加优秀！

大数据应用？

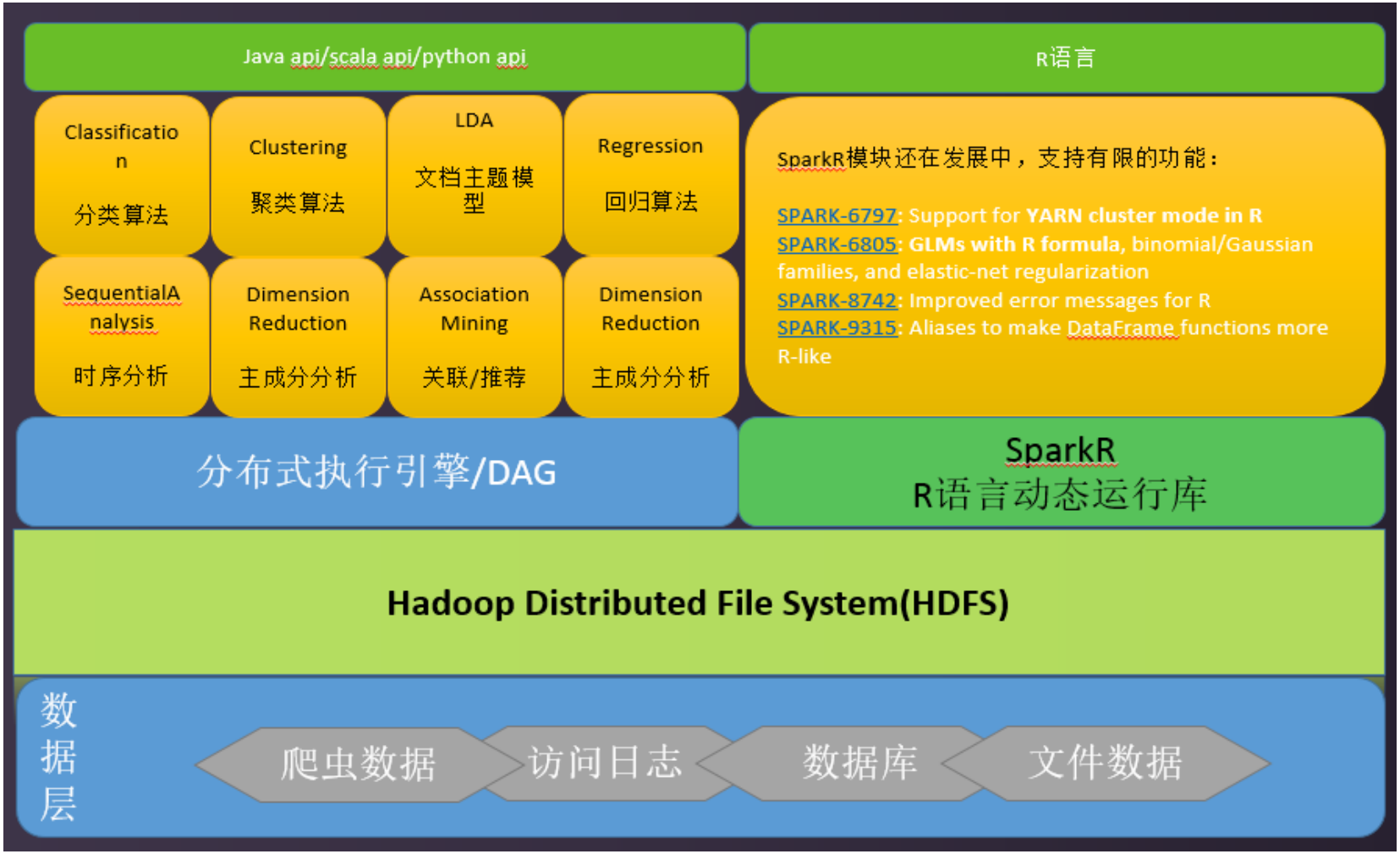


http://www.36dsj.com/archives/11913

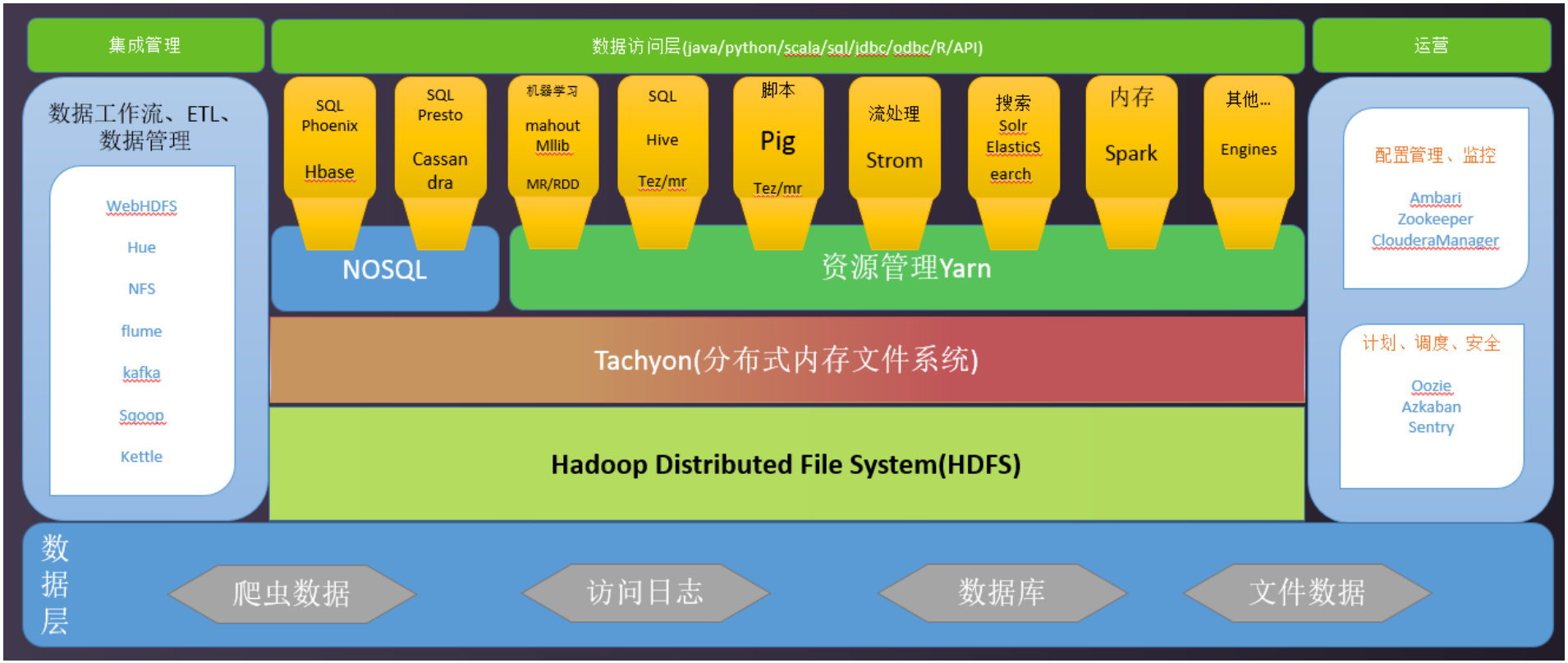
X on Yarn


集成管理
SQL on Hadoop/SQL on Nosql
运营
数据工作流、ETL、
数据管理
WebHDFS
Hue
NFS
flume
kafka
Sqoop
Kettle
SQL
Phoenix
Hbase
SQL
Presto
Cassandra
SQL2003
Presto
MPP
SQL2003
Impala
MPP
ANSI-SQL
Drill
Dremel
SQL
HAWQ
MPP DB
SQL
Spark
RDD
类SQL92
Hive
Tez/mr
其他...
Engines
NOSQL
资源管理Yarn/standalone
Hadoop Distributed File System(HDFS)
配置管理、监控
Ambari
Zookeeper
ClouderaManager
计划、调度、安全
Oozie
Azkaban
Sentry
数据层
爬虫数据
访问日志
数据库
文件数据

分布式机器学习


Java api/scala api/python api
R语言
Classification
分类算法
Clustering
聚类算法
LDA
文档主题模型
Regression
回归算法
SequentialAnalysis
时序分析
Dimension Reduction
主成分分析
Association Mining
关联/推荐
Dimension Reduction
主成分分析
SparkR模块还在发展中，支持有限的功能：
SPARK-6797: Support for YARN cluster mode in R
SPARK-6805: GLMs with R formula, binomial/Gaussian families, and elastic-net regularization
SPARK-8742: Improved error messages for R
SPARK-9315: Aliases to make DataFrame functions more R-like
分布式执行引擎/DAG
SparkR
R语言动态运行库
Hadoop Distributed File System(HDFS)
数据层
爬虫数据
访问日志
数据库
文件数据

Hadoop Platform



SQL on Hadoop选型？

Thank you

提问时间？

PPT： https://github.com/itweet/course