Practices on Big Data Platform

Hadoop Introduction & Spark Practices

Qiong Zeng (曾琼)
qiong.zn@sdu.edu.cn

Research *is creative and systematic work undertaken to increase the stock of knowledge.*

学习目标

了解大数据历史及生态

了解大数据平台重要组件的基本原理

掌握Spark的基本原理

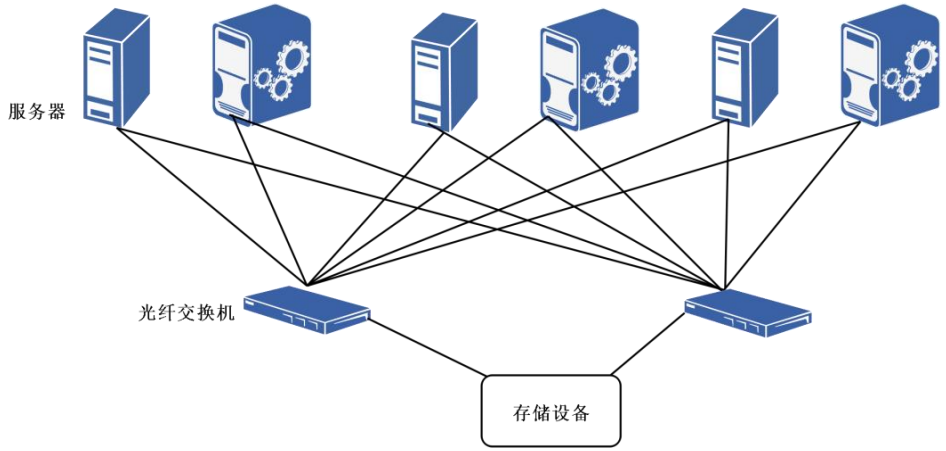
掌握Spark的初步使用

- Spark SQL
- Spark Core

目录

大数据发展史 - 大数据出现之前

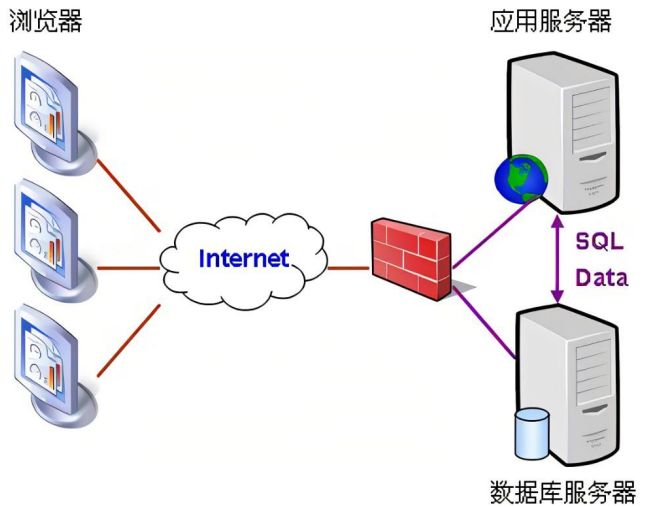
早期IT硬件拓扑



NAS
(Network Attached Storage)

SAN
(Storage Area Network)

早期IT软件拓扑



大数据发展史 - 推动力

个人电脑的普及

互联网的普及

搜索引擎以及网络广告价值凸显

海量的日志存储及分析具备了商业价值

低成本、分布式的日志存储分析具备了商业价值

大数据快速发展的核心原因

异常庞大的个人的互联网行为具备了非常高的经济价值

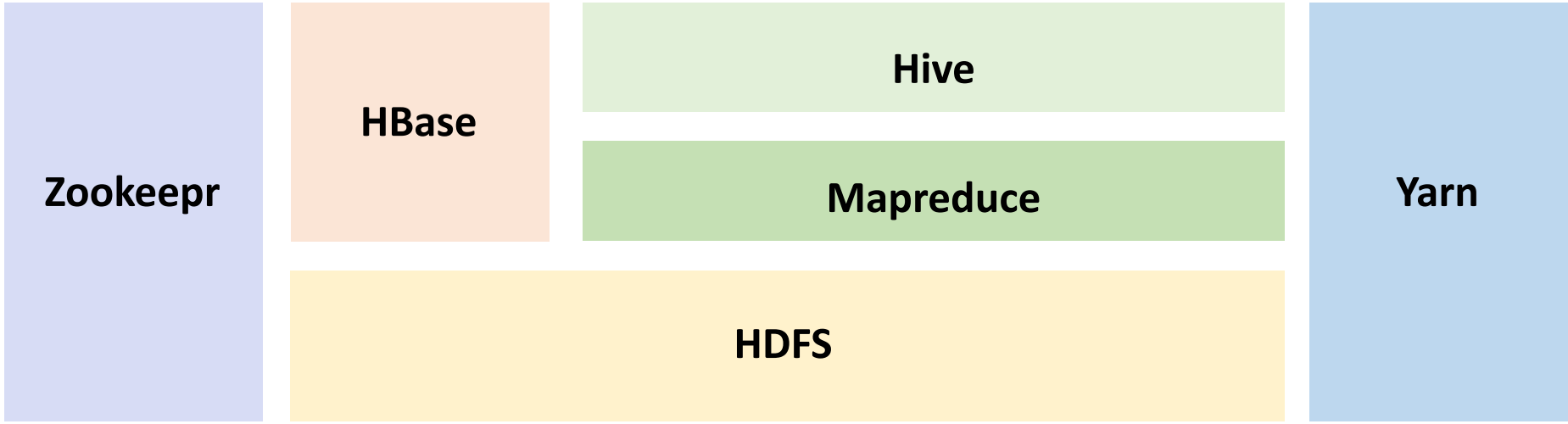
大数据发展史 - 神奇的谷歌公司

提出了三个大数据问题的关键解决方案：

- 如何用廉价的服务器来构建高可用**分布式文件系统**
- 如何实现高性能分布式的**半结构化数据库**
- 如何利用廉价的服务器来分析海量数据

Google File System
https://static.googleusercontent.com/media/research.google.com/zh-CN//archive/gfs-sosp2003.pdf
Google Mapreduce
https://static.googleusercontent.com/media/research.google.com/zh-CN//archive/mapreduce-osdi04.pdf
Google Bigtable
https://static.googleusercontent.com/media/research.google.com/zh-CN//archive/bigtable-osdi06.pdf

大数据发展史 - 早期Hadoop



- **(存)HDFS：分布式文件系统，所有功能的基础**

Zookeeper：分布式协调服务(存储公共元数据、维护应用程序心跳)

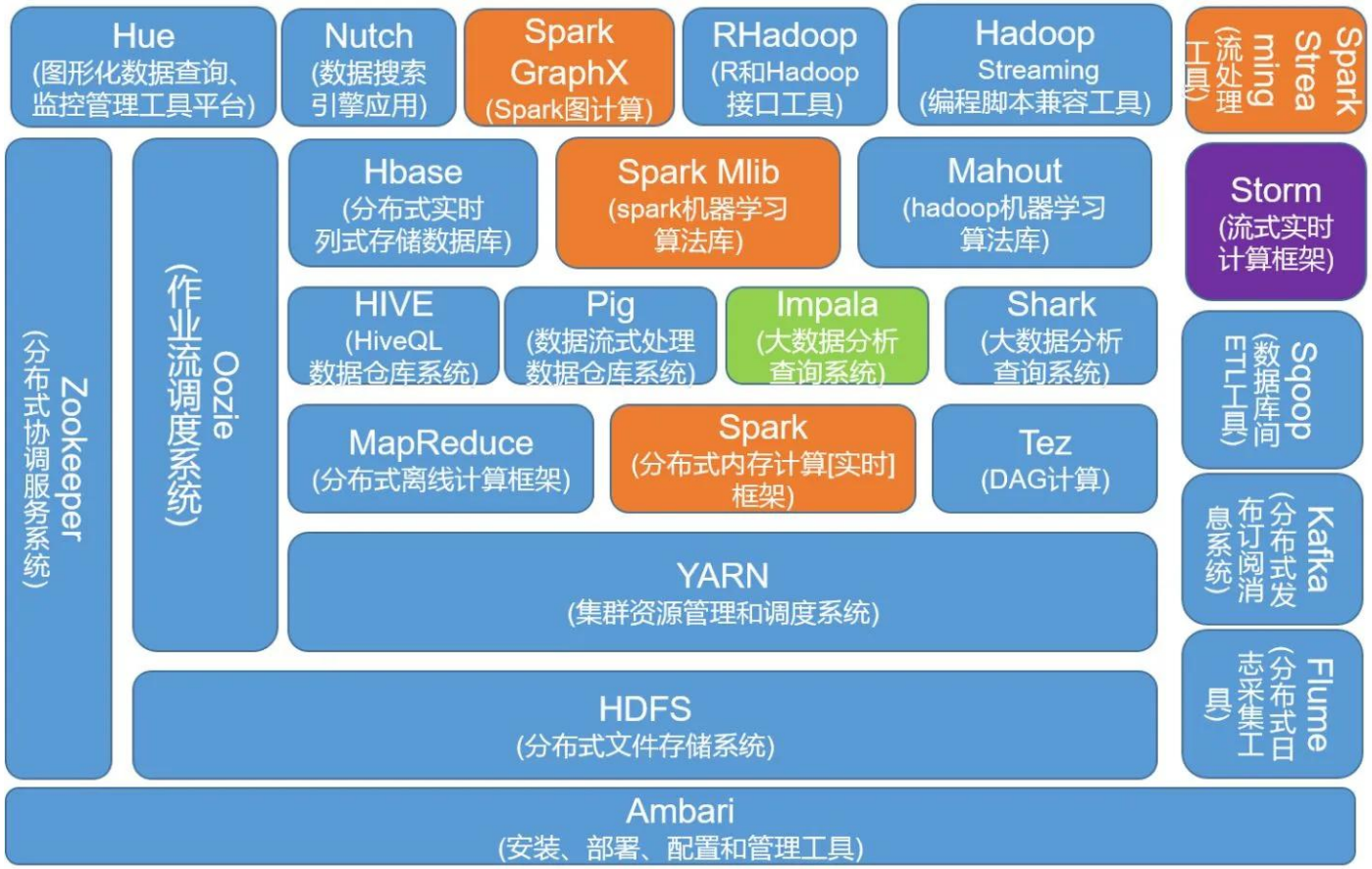
- **(数据库)HBase：分布式的NoSQL数据库**

Yarn：Hadoop的资源管理器

- **(算)Mapreduce：在HDFS和Yarn的基础上，实现分布式的计算框架**

Hive：Hadoop的数据仓库工具，将SQL语句转化成Mapreduce语句

大数据发展史 - 成熟的Hadoop



Flink 实时计算

重点扩充：

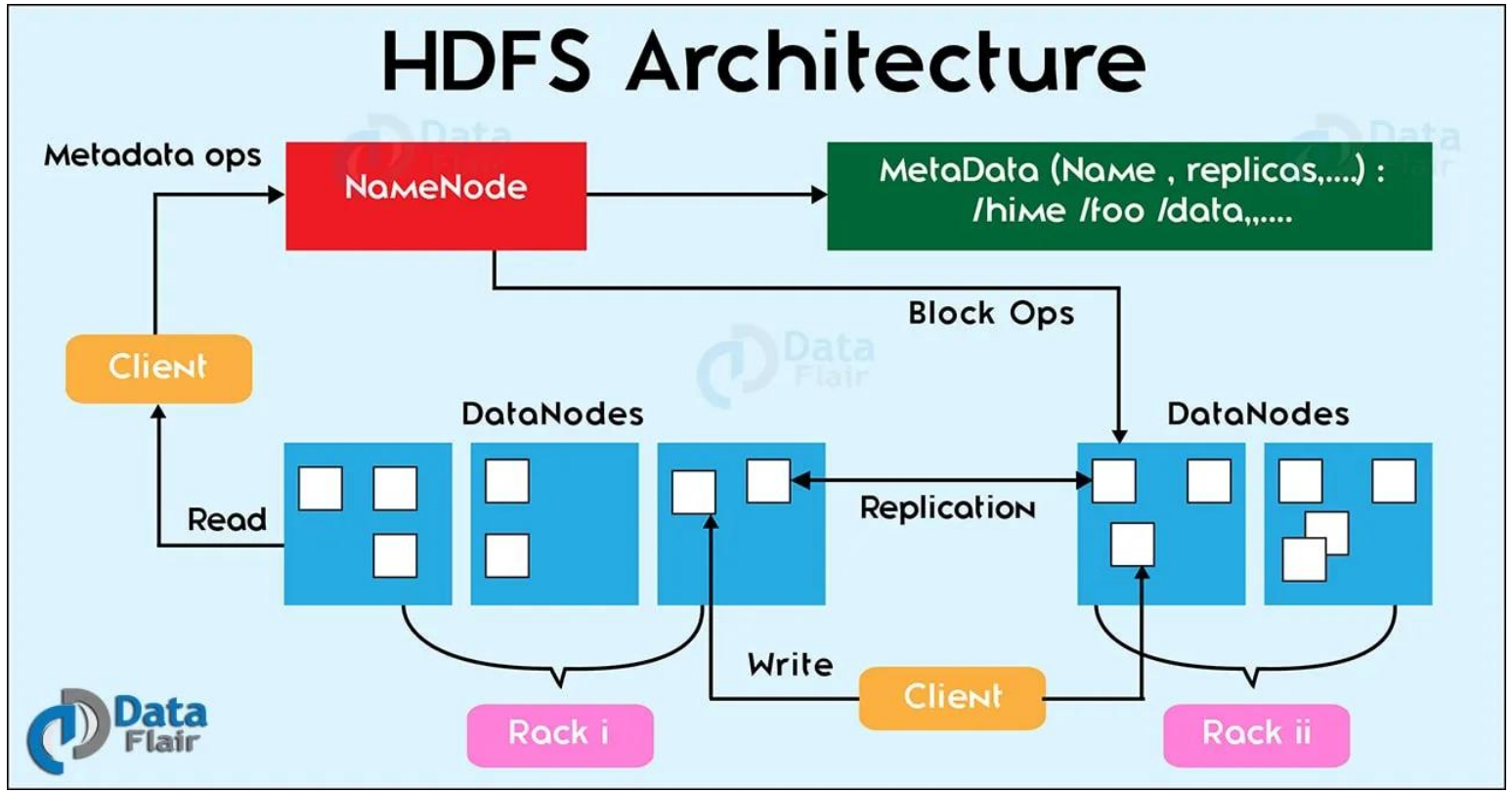
- 数据ETL
- 流计算/实时计算
- 数据仓库
- Spark生态
- 运维管理平台

名词解释：

ETL：指数据从业务系统抽取、转化、载入到大数据平台的过程

流计算：指消费实时产生的流数据，并进行即时计算，比如双十一的订单统计

重要组件原理 - HDFS逻辑架构



NameNode(NN)：存储元数据　　**DataNode(DN)：存储实际数据**

重要组件原理 - HDFS

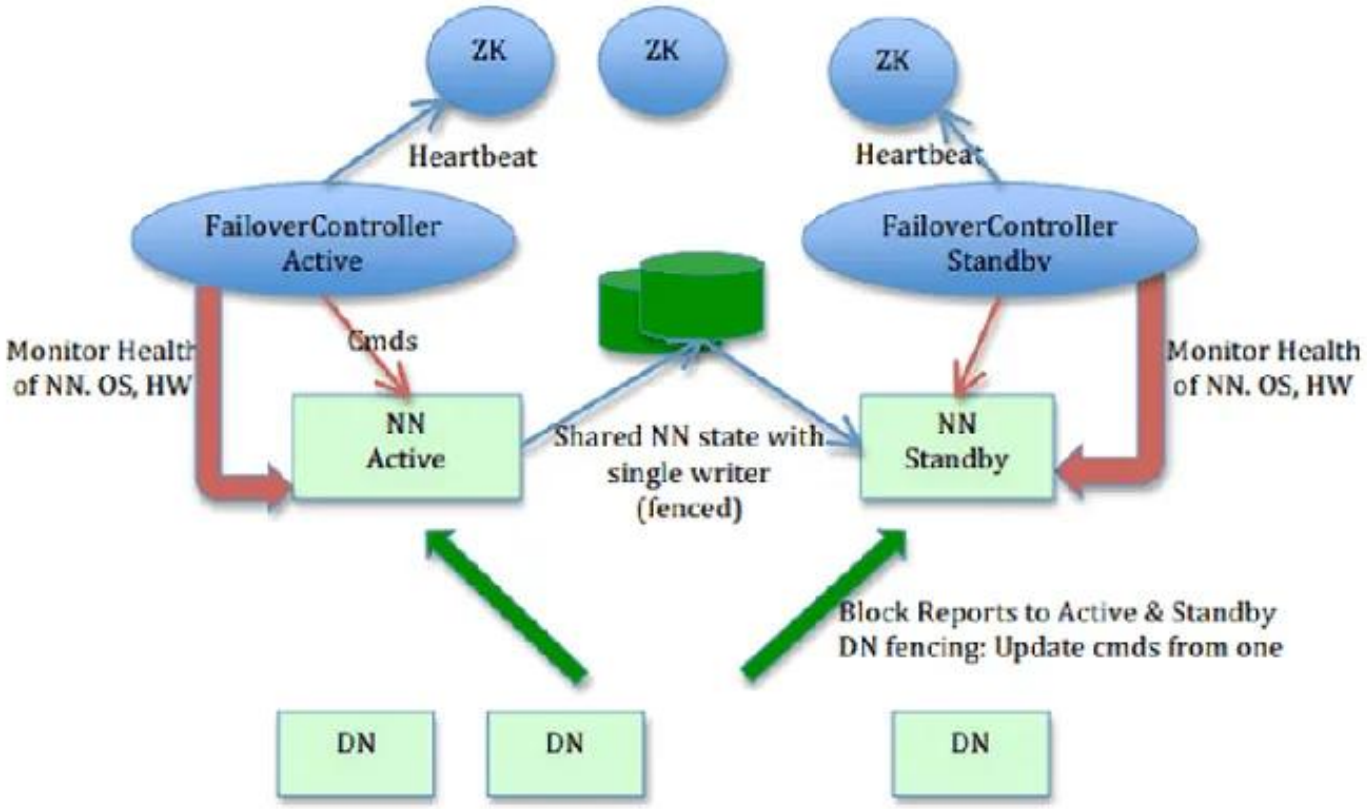
如何分布式

- 写入：文件拆分成多个Block，写入到不同的DataNode上，突破了单机存储能力的限制
- 读取：同时从多个DataNode读取文件对应的Block，重新组合成文件

如何高可用

- 元数据：采用热备方式
- 实际数据：采用多副本的方式，即一个Block会有多个副本，在一个DataNode不可用时，从其他副本获取数据
- 数据维护：在检测到有DataNode不可用时，此Node上维护的Block会在其他节点上重新产生，使系统一直保持固定的副本数

重要组件原理 - HDFS高可用拓扑架构

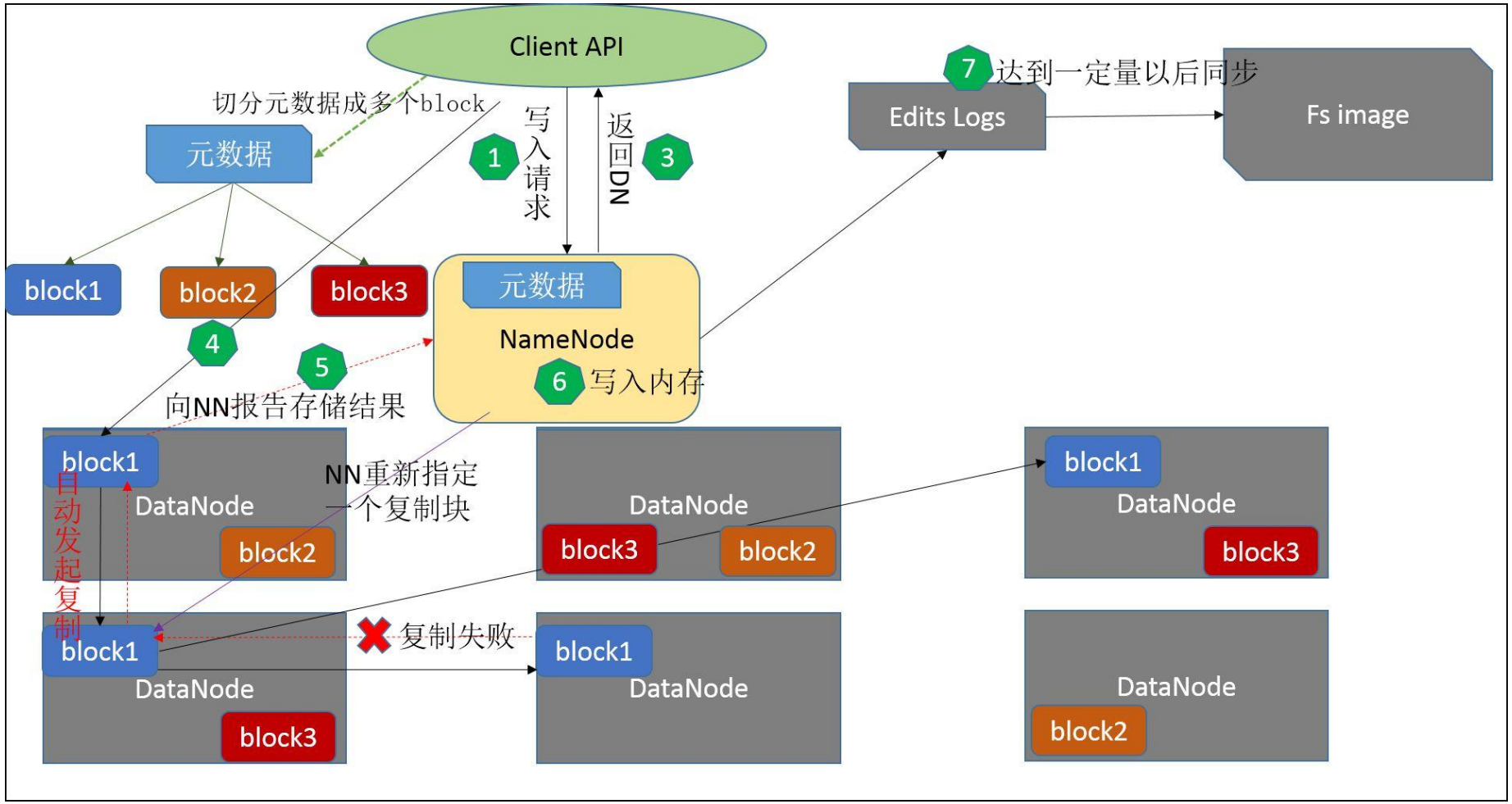


NameNode(NN)：存储元数据

DataNode(DN)：存储实际数据

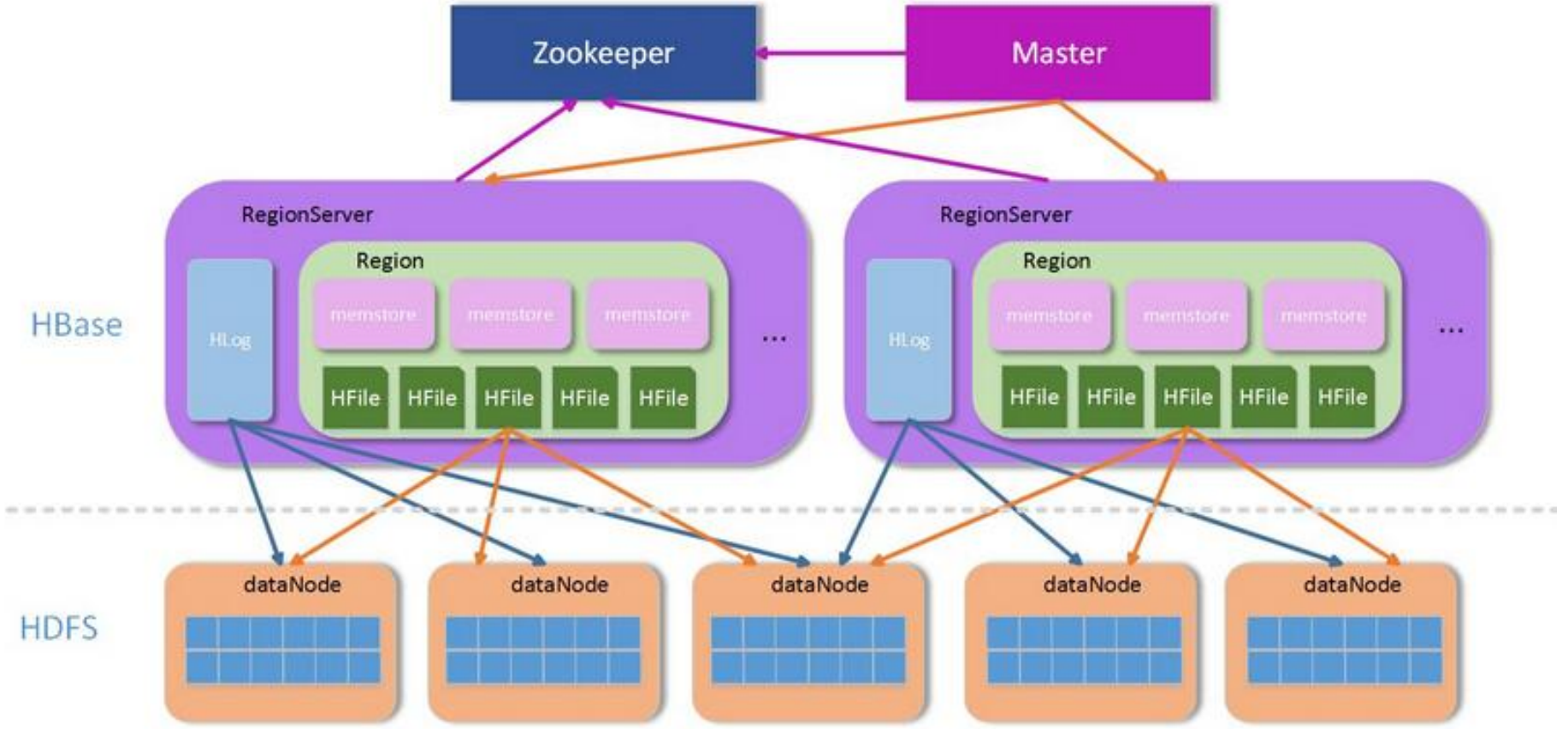
NameNode服务会定期向zookeeper发送心跳。当超过一定时间收不到心跳，认为节点故障，会触发主备切换操作。

重要组件原理 - HDFS逻辑流程



元数据：文件的分片及分片存储信息　　**Block：文件的实际分片**

重要组件原理 - HBase



HBase概念

Master：管理节点，维护元数据、索引等

RegionServer：实际工作节点，负责数据存储和读取

外部组件

Zookeeper：存储元数据、探活RegionServer

重要组件原理 - HBase概念

Region

- HBase的一个管理单元，一个表由一个或者多个region组成
- 管理节点维护region与regionserver的对应关系

记录

- 一条记录对应一个rowkey，可以理解为一个kv的数据库
- HBase表中，所有记录实际存储按rowkey的字典序排列
- 每条记录可以包含不固定的列名和值(区别与关系数据库)

HFile

- 对应实际的存储单元
- 一个Region对应多个HFile

重要组件原理 - HBase概念

HLog

- Write Ahead Log (WAL)
- HBase插入、修改、删除都会先记录到此日志中

MemStore

- HBase读取HLog，执行操作后会先在内存中形成副本即memstore
- memstore定期刷写成HFile
- HFile定期合并

重要组件原理 - HBase高可用

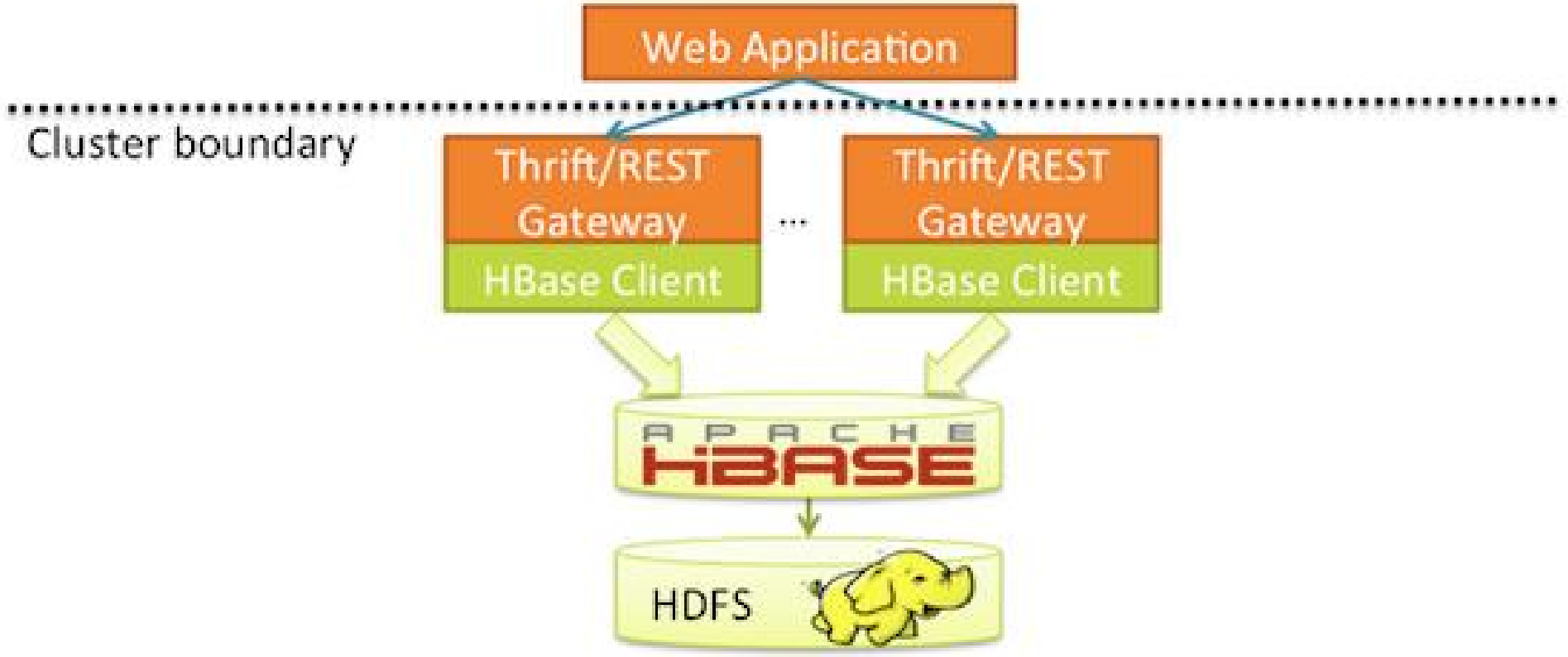
如何分布式

- 表纵向拆分：不同的列簇，分布在不同的region
- 表横向拆分：region可以在过大时，自动拆分，或者人为拆分

如何高可用

- 数据高可用
  - 元数据：在分布式协调服务zookeeper中，多节点存储，高可用
  - 实际数据：存储在HDFS中，天然高可用
- 服务高可用
  - Master节点：采用热备方式
  - RegionServer节点：探活机制，当发现有RegionServer不可用，其管理的Region交由其他RegionServer管理

重要组件原理 - HBase访问

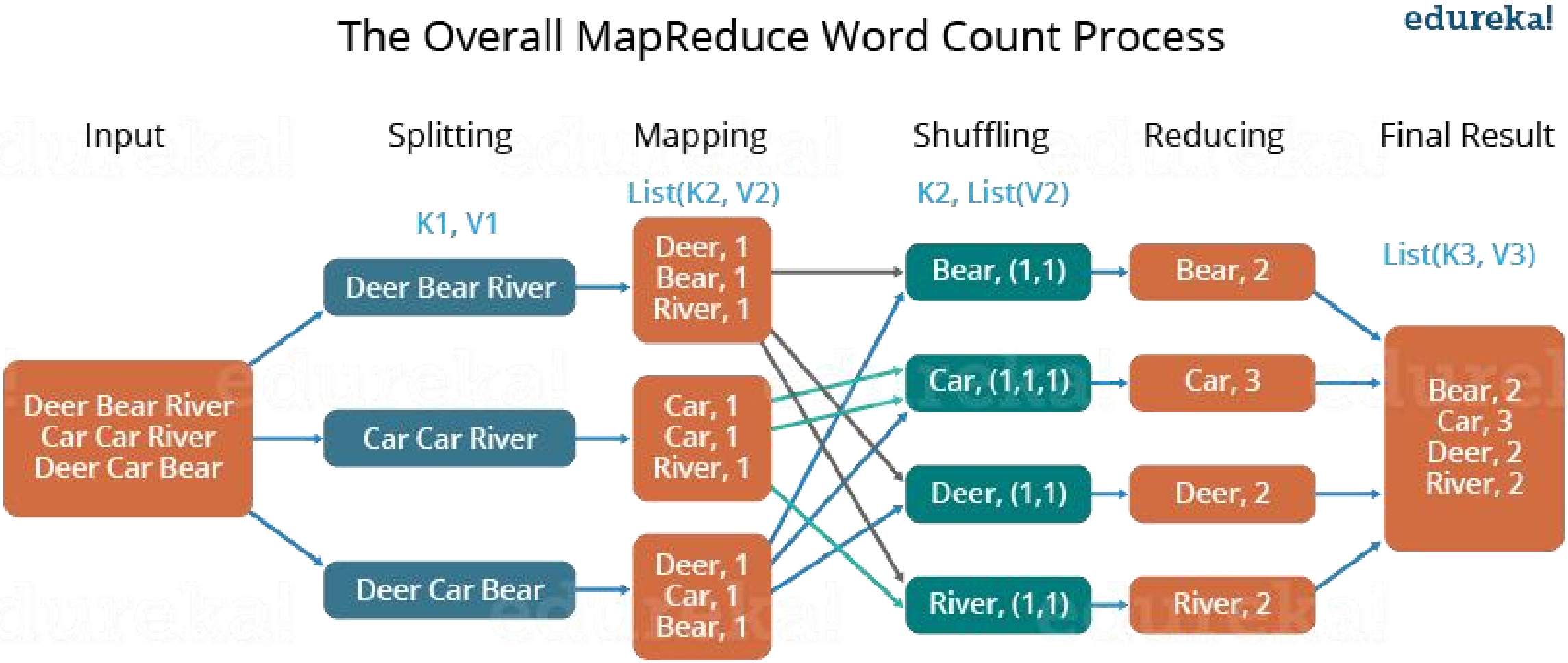


接口访问方式：

- **原生Java库**
- **Thrift Server(推荐) 支持多种语言： C++、 Java、 Python、 PHP 等**

重要组件原理 - Mapreduce

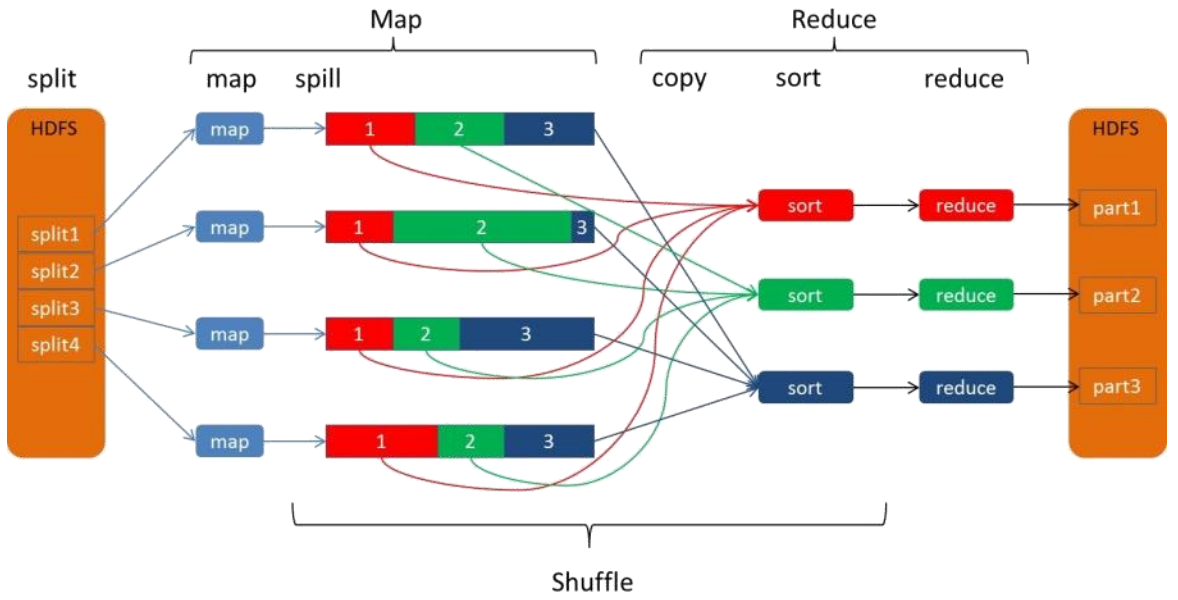
MapReduce 经典示例(word count)



重要组件原理 - Mapreduce

Map：从HDFS读取数据，根据key进行分区，并缓存在内存缓冲区

Spill：内存缓冲区满之后，对缓冲区内数据排序，记录到两个文件：spill0.out(数据)，spill0.index(索引，partition对应的位置)

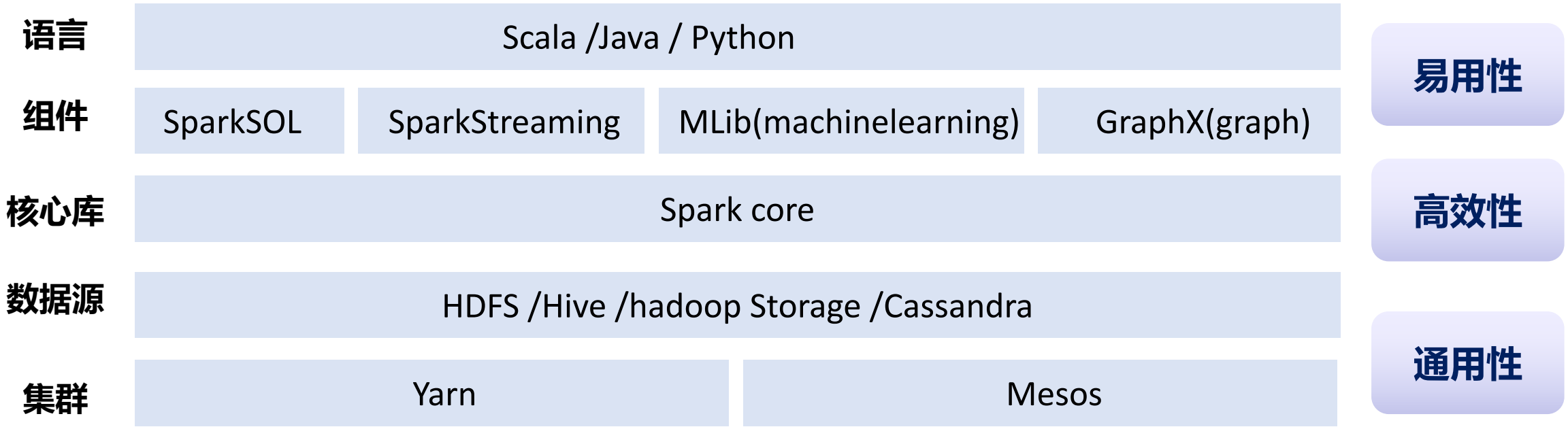


Copy：Reducer会根据自己对应的partition，向mapper请求已经spill的数据。

Sort：将索引数据按照partition和key两个关键字使用快排算法进行排序，排序结果是kvmeta中数据按照partition为单位聚集在一起，同一partition内的按照key有序

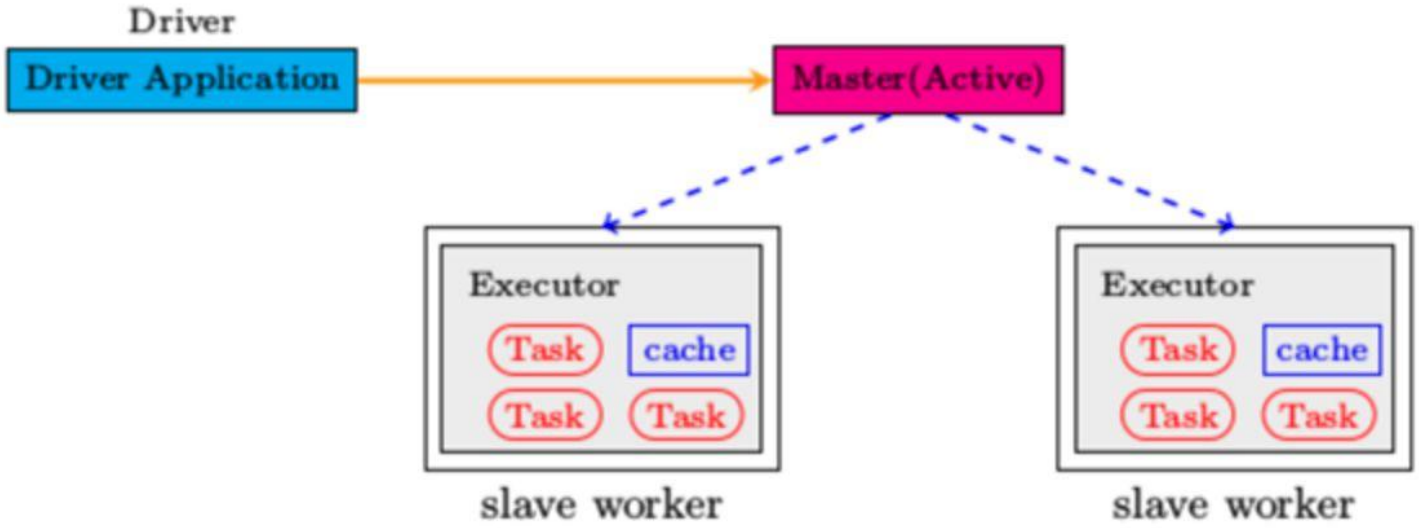
Merge：合并所有的spillxx.out 文件到file.out，合并所有的spillxx.index到file.out.index

重要组件原理 - Spark



- Spark core是Spark的核心框架，可以运行在多种资源管理器(如Yarn、Mesos)之上，支持多种存储系统以及数据库作为输入输出。
- Spark core基础上封装了Spark SQL、Spark Streaming、MLib、GraphX分别对应数据分析、流处理、机器学习、图计算等领域

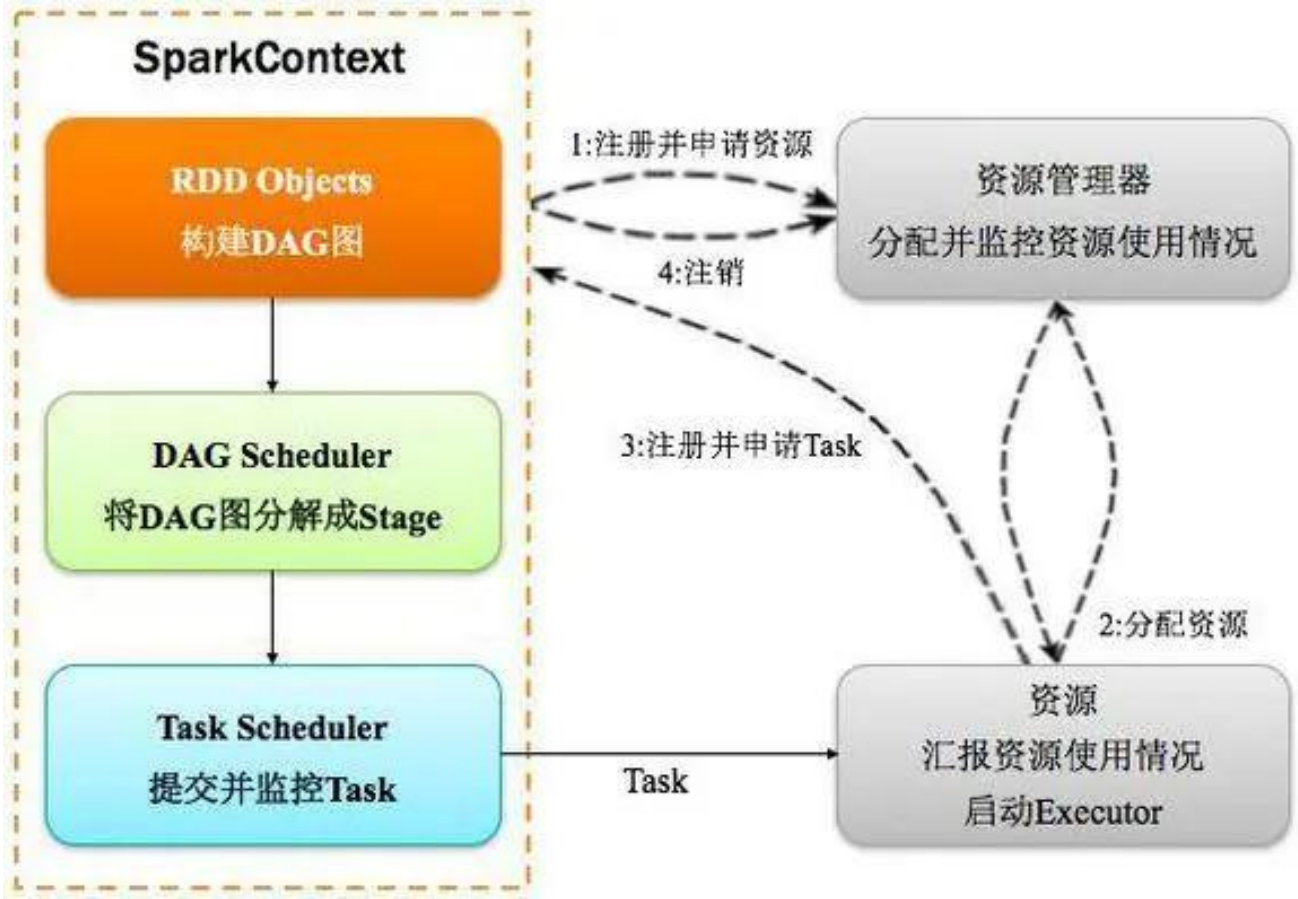
重要组件原理 - Spark



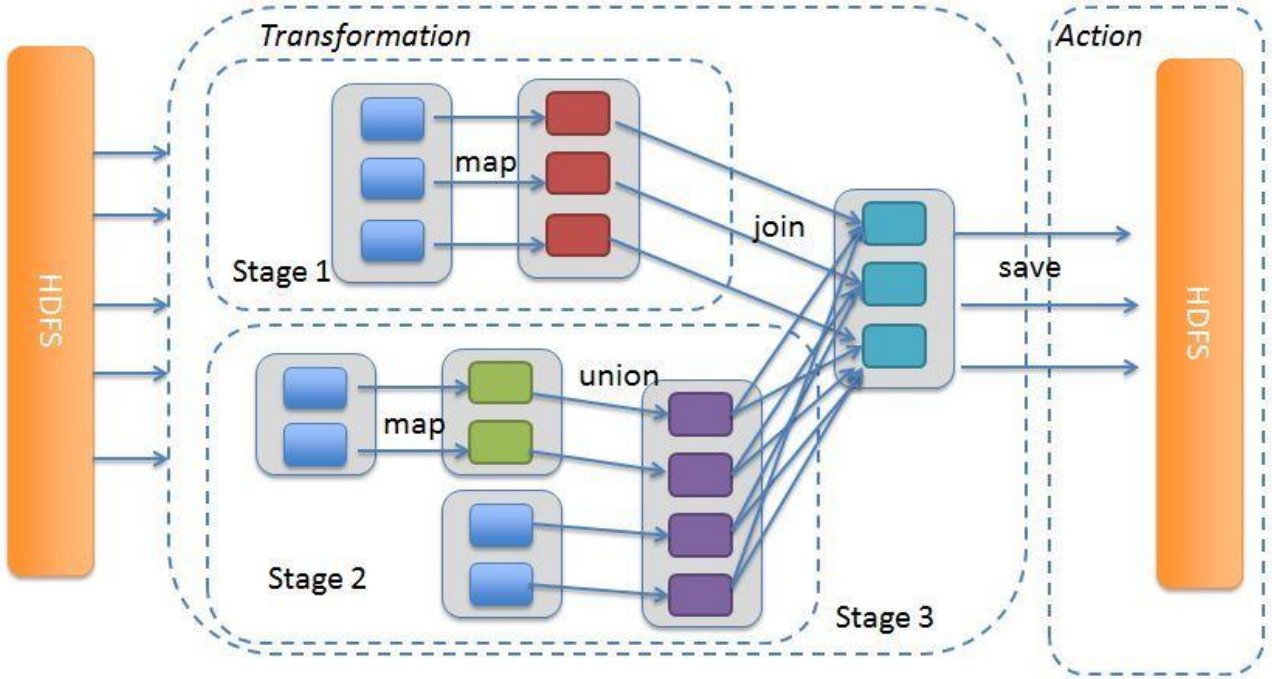
- 应用程序(Application): 基于Spark的用户程序，包含了一个Driver Program 和集群中多个的Executor；
- 驱动(Driver): 运行Application的main()函数并且创建SparkContext;
- 执行单元(Executor): 是为某Application运行在Worker Node上的一个进程，该进程负责运行Task，并且负责将数据存在内存或者磁盘上，每个Application都有各自独立的Executors;
- 集群管理程序(Cluster Manager): 在集群上获取资源的外部服务(例如：Local、Standalone、Mesos或Yarn等集群管理系统)；

重要组件原理 - Spark

Spark作业执行流程



重要组件原理 - Spark内部概念



算子（函数）是作用在partition上的。Spark中RDD的计算是以分片为单位的，每个RDD都会实现compute函数以达到这个目的。

- RDD(Resilient Distributed Datasets)，弹性分布式数据集，是分布式内存的一个抽象概念，RDD是由一系列parition组成。RDD之间有依赖关系，RDD的每次转换都会生成一个新的RDD。
- Parition提供数据最佳的计算位置，有利于数据处理的本地化。按照“移动数据不如移动计算”的理念，Spark在进行任务调度的时候，会尽可能地将计算任务分配到其所要处理数据块的存储位置。
- Stage是由一组并行的算子组成，Stage的划分依据就是看是否产生了shuflle(即宽依赖)，遇到一个shuffle操作就划分为前后两 个stage。

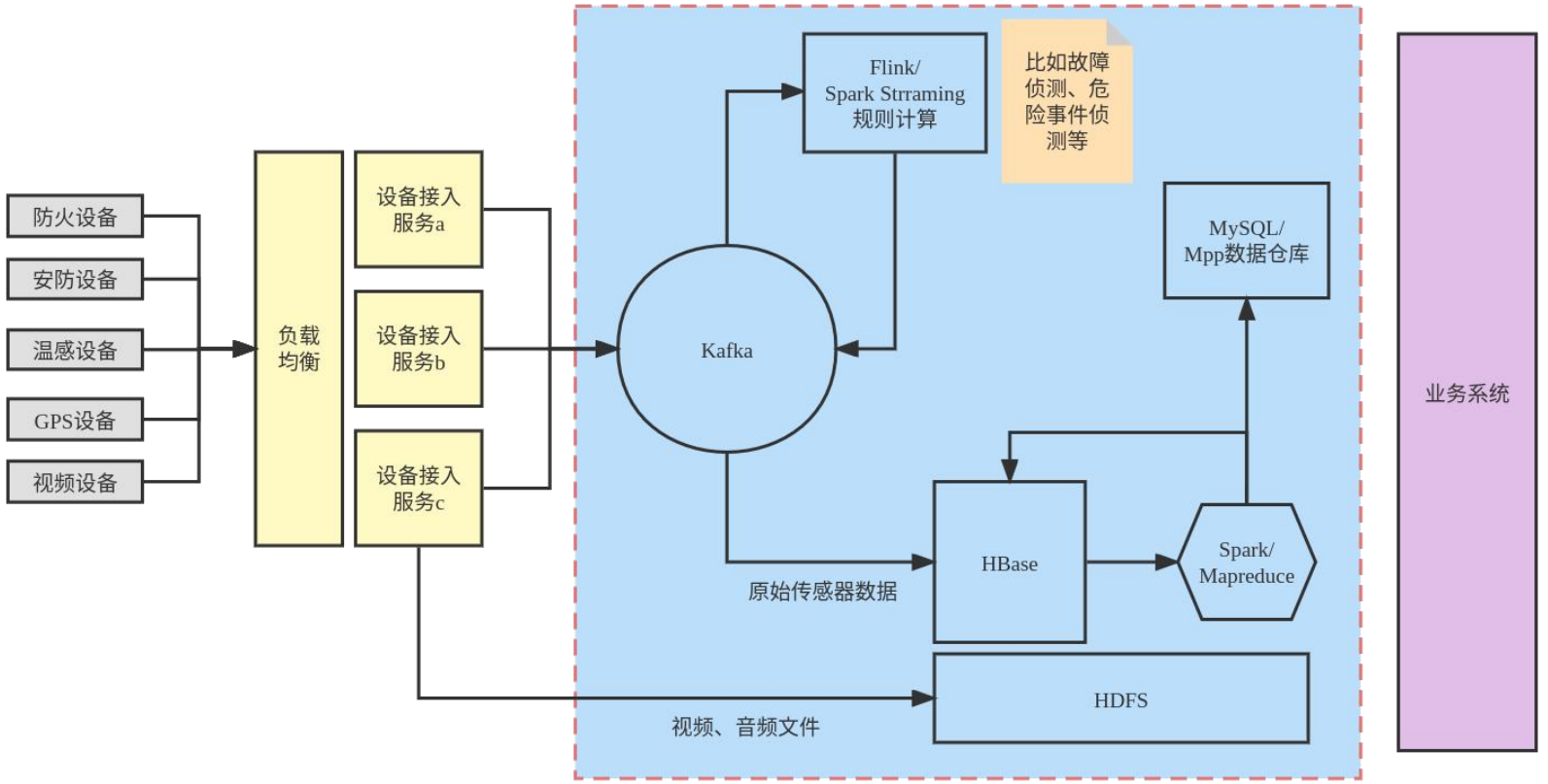
重要组件原理 - Spark VS Mapreduce

Hadoop的局限	Spark的改进
抽象层次低，代码编写难以上手	通过使用RDD的统一抽象，实现数据处理逻辑的代码非常简洁
只提供Map和Reduce两个操作,欠 缺表达力	通过RDD 提供了很多转换和动作，实现了很多基本操作，如 Sort、Join 等
一个Job只有 Map和Reduce 两个阶段，复杂的程序需要大量的 Job 来完成，且Job之间的依赖关系需要应用开发者自行管理	一个Job可以包含多个RDD的转换操作只需要在调度时生成多个 Stage。一个 Stage 中也可以包含多个 Map 操作，只需Map 操作所使用的 RDD分区保持不变。
处理逻辑隐藏在代码细节中,缺少整体逻辑视图	RDD的转换支持流式 API，提供处理逻辑的整体视图。
对迭代式数据处理性能比较差，Reduce与下一步 Map 之间的中间结果只能存放在 HDFS 文件系统中	通过内存缓存数据，可大大提高迭代式计算的性能，内存不足时可以溢出到磁盘上
ReduceTask需要等待所有MapTask都完成后才能开始执行	分区相同的转换可以在一个 Task 中以流水线形式执行，只有分区不同的转换需要 Shufle 操作
时延高，只适用批数据处理，对交互式数据处理和实时处理的支持不够	将流拆成小的Batch，提供 Discretized Stream 处理流数据

Hadoop & Spark对比

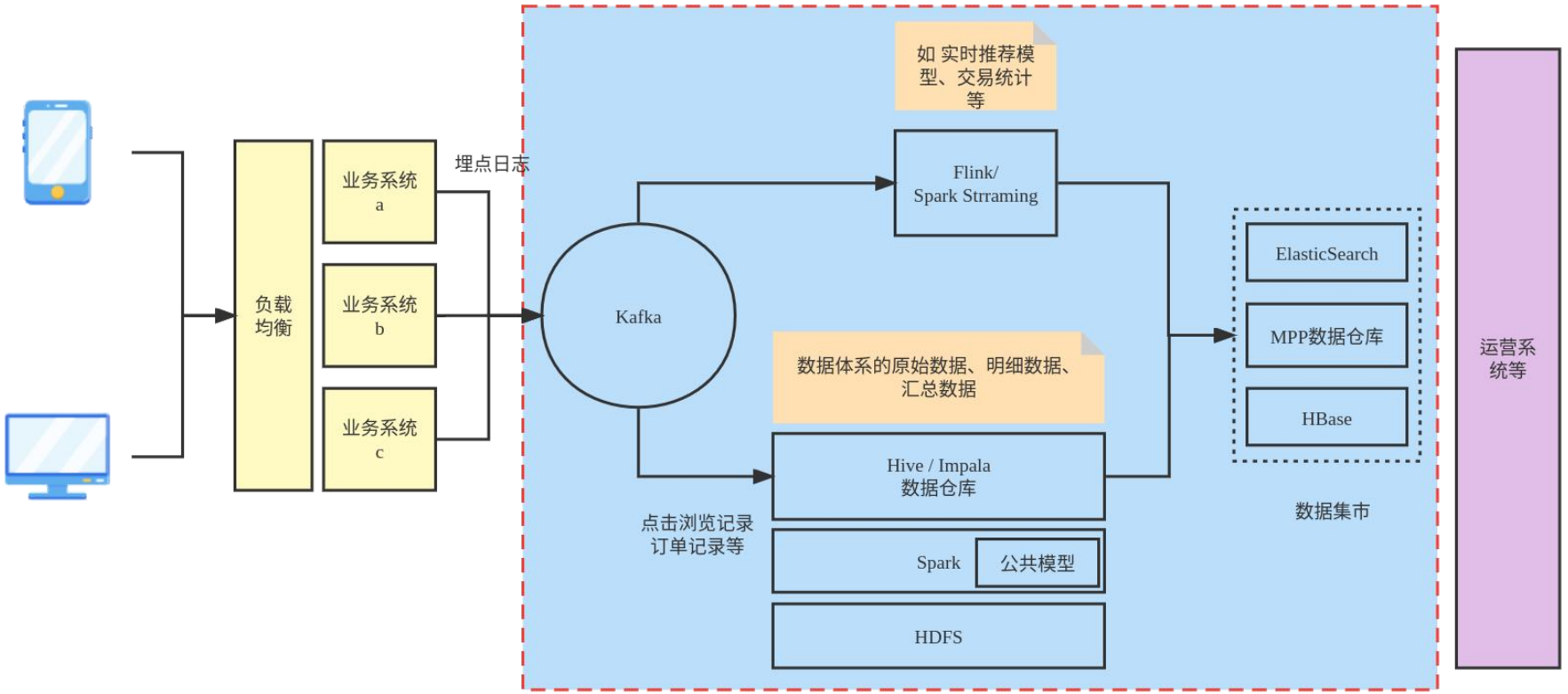
Spark 也有其劣势。由于 Spark 基于内存进行计算，在真正面对大数据的时候（比如一次操作针对10亿以上级别），在没有进行调优的情况下，可能会出现各种各样的问题，比如OOM内存溢出等等。导致Spark程序可能都无法完全运行起来，就报错挂掉了，而MapReduce即使是运行缓慢，但是至少可以慢慢运行完。

大数据平台物联网应用



物联网典型特点是对设备数据的存储、查询、分析，其中以原始数据的查询最为核心，HBase在物联网中一般是最核心的组件，Spark等对设备上报的海量数据提供计算支撑

大数据平台互联网应用



互联网典型特点是网络行为的存储和分析，是以数据仓库(如Hive、Impala等)为最核心的存储组件，并辅以Spark、Flink等对数据进行统计和模型分析(如推荐系统、用户画像等等)，支撑更好的运营

Spark实践 - Spark core

定义Spark的上下文、输入、输出、并行度，并将输入数据经过一系列算子处理后将数据输出的过程。

常用算子举例：

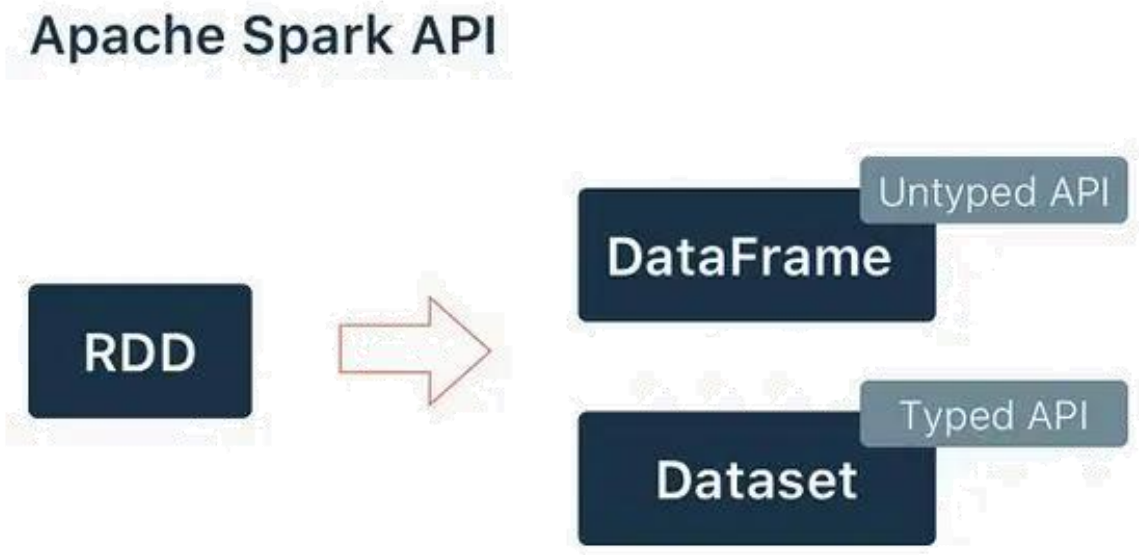
Transformations	*map*(*f* : T ⇒ U)	:	RDD[T] ⇒ RDD[U]
	filter(*f* : T ⇒ Bool)	:	RDD[T] ⇒ RDD[T]
	flatMap(*f* : T ⇒ Seq[U])	:	RDD[T] ⇒ RDD[U]
	sample(*fraction* : Float)	:	RDD[T] ⇒ RDD[T] (Deterministic sampling)
	groupByKey()	:	RDD[(K, V)] ⇒ RDD[(K, Seq[V])]
	reduceByKey(*f* : (V, V) ⇒ V)	:	RDD[(K, V)] ⇒ RDD[(K, V)]
	union()	:	(RDD[T], RDD[T]) ⇒ RDD[T]
	join()	:	(RDD[(K, V)], RDD[(K, W)]) ⇒ RDD[(K, (V, W))]
	cogroup()	:	(RDD[(K, V)], RDD[(K, W)]) ⇒ RDD[(K, (Seq[V], Seq[W]))]
	crossProduct()	:	(RDD[T], RDD[U]) ⇒ RDD[(T, U)]
	mapValues(*f* : V ⇒ W)	:	RDD[(K, V)] ⇒ RDD[(K, W)] (Preserves partitioning)
	sort(*c* : Comparator[K])	:	RDD[(K, V)] ⇒ RDD[(K, V)]
	partitionBy(*p* : Partitioner[K])	:	RDD[(K, V)] ⇒ RDD[(K, V)]
Actions	*count*()	:	RDD[T] ⇒ Long
	collect()	:	RDD[T] ⇒ Seq[T]
	reduce(*f* : (T, T) ⇒ T)	:	RDD[T] ⇒ T
	lookup(*k* : K)	:	RDD[(K, V)] ⇒ Seq[V] (On hash/range partitioned RDDs)
	save(*path* : String)	:	Outputs RDD to a storage system, *e.g.*, HDFS

Spark实践 - word count解读

```scala
import org.apache.spark.{SparkConf, SparkContext}

object WordCount {
  def main(args:Array[String]): Unit ={
    // 配置作业信息，并生成上下文
    val conf=new SparkConf().setAppName("wordcount").setMaster("local[*]")
    val sc=new SparkContext(conf)
    System.setProperty("HADOOP_USER_NAME", "root")

    // 设置数据来源/输入
    val lines=sc.textFile("hdfs://master-ubuntu:9000/wordcount/wordcount.txt")

    // 计算步骤/算子的串联
    val flatmaprdd=lines.flatMap(line=>line.split(" "))
    val maprdd=flatmaprdd.map(word=>(word,1))
    val reducerdd=maprdd.reduceByKey{case(x,y)=>x+y}

    // 输出结果
    reducerdd.saveAsTextFile("hdfs://master-ubuntu:9000/wordcount/output")
  }
}
```

Spark实践 - Spark Sql

SparkSQL：像使用数据库一样使用Spark



DataFrame是一种以RDD为基础的分布式数据集，类似于传统数据库中的二维表格。DataFrame与RDD的主要区别：DataFrame所表示的二维表数据集的每一列都带有名称和类型

SparkSQL是对Spark Core的封装及抽象

- 算子 --> SQL
- RDD --> DataFrame & DataSet

Spark实践 - 环境准备

- 准备Linux虚拟机/云主机
- 安装JDK1.8
  - (可参照：https://baijiahao.baidu.com/s?id=1738251493206917341&wfr=spider&for=pc)
- 安装scala 2.12
  - (可参照:https://blog.51cto.com/StarBigData/3643847)
- 安装Maven3.6+
  - (可参照:https://blog.csdn.net/qq_34845394/article/details/90674933)
- 下载Spark包 spark-3.3.1-bin-hadoop3.tgz
- 下载https://github.com/pconrad/Spark-Java-Maven-Hello-World

Spark实践 - 示例运行

- 进入Spark目录
- 计算Pi的值
  - ./bin/spark-submit --master local --class org.apache.spark.examples.SparkPi examples/jars/spark-examples_2.12-3.3.1.jar
- Word Count
  - ./bin/spark-submit --master local --class org.apache.spark.examples.JavaWordCount examples/jars/spark-examples_2.12-3.3.1.jar [your own file path]
- SparkSQL
  - ./bin/spark-submit --master local --class org.apache.spark.examples.sql.SparkSQLExample examples/jars/spark-examples_2.12-3.3.1.jar

Spark实践 - 手写代码spark-scala-shell

到spark的bin目录下，执行./spark-shell

```
import org.apache.spark.rdd.RDD

var file: String = s"/home/songli/test.txt"

// 读取文件内容
var lineRDD: RDD[String] = spark.sparkContext.textFile(file)

// 以行为单位做分词
var wordRDD: RDD[String] = lineRDD.flatMap(line => line.split(" "))
var cleanWordRDD: RDD[String] = wordRDD.filter(word => !word.equals(""))

// 把RDD元素转换为（Key，Value）的形式
var kvRDD: RDD[(String, Int)] = cleanWordRDD.map(word => (word, 1))
// 按照单词做分组计数
var wordCounts: RDD[(String, Int)] = kvRDD.reduceByKey((x, y) => x + y)

// 打印词频最高的5个词汇
wordCounts.map{case (k, v) => (v, k)}.sortByKey(false).take(5)
```

Spark实践 - 动手写代码spark java

- 进入Spark-Java-Maven-Hello-World工程
- 修改Java代码，完成如下功能：

  参照示例(spark-3.3.1-bin-hadoop3/examples/src/main/java/org/apache/spark/examples/JavaWordCount.java)
  自己编译并正常提交运行word count
- 完成后编译： mvn clean compile assembly:single
- 编译通过后使用spark-submit提交
- 注意点：需要引入依赖spark-core_2.12、 spark-sql_2.12，版本对应自己使用的spark版本
- 答案： https://github.com/sbookworm/Spark-Java-Maven-Word-Count

Spark实践 - 实验课题目

- 参照代码:
  spark-3.3.1-bin-hadoop3/examples/src/main/java/org/apache/spark/examples/sql/JavaSparkSQLExample.java
- 数据集链接:
  https://pan.baidu.com/s/1XF54wFWnHmWmB8JdHo_M6g?pwd=phv4
  提取码: phv4
- 题目： 使用Spark求工作日客流最大的门店ID(选做： 日均客流量)
- 数据集说明:

Store	DayOfWeek	Date	Sales	Customers	Open	Promo
商店ID	星期	日期	销量	顾客数	是否开门	是否优惠

Thank You