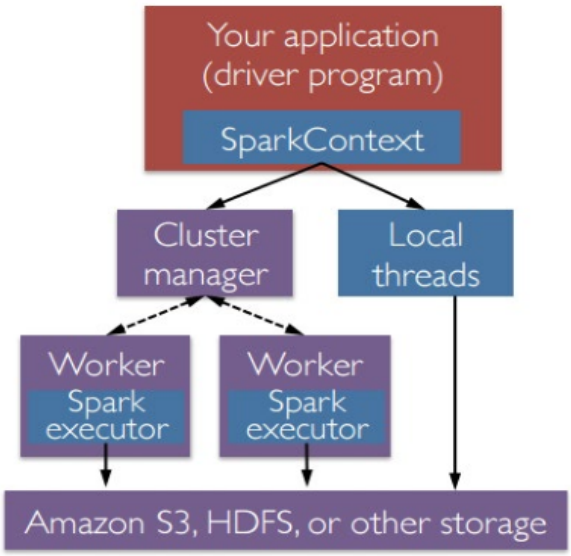
Spark介绍

大量数据会导致一台机器无法操作甚至存储所有数据，一个解决办法是将数据分布到集群计算机上。

Spark是快速通用的集群计算系统，可与Hadoop互操作，通过in memory computing primitives和general computation graphs提高效率，通过丰富的如scala, java, Python等API和交互式shell提高可用性。

Spark组件



一个Spark程序第一步创建SparkContext/SparkSession对象（driver），用来告诉Spark怎样并且在哪里访问一个cluster，并且能够连接到集中不同类型的cluster manager（YARN或者它自己的manager）

Cluster manager能够分配资源应用

Spark executor（worker）能够执行计算，访问内存

Spark和SQL组件 一个Spark程序可以分为：driver（驱动）程序和workers程序，worker programs在集群节点或者本地线程上运行。

一个Spark程序先创建SparkContext对象，告知Spark怎样和在哪儿访问集群，pySpark shell会自动创建SparkContext对象但ipython必须创建。然后创建一个sqlContext对象。最后运行sqlContext对象去创建DataFrames。

Spark Essentials: Master SparkContext / SparkSesion的主参数确定要使用的群集类型和大小

Master Parameter	Description
local	run Spark locally with one worker thread (no parallelism)
local[K]	run Spark locally with K worker threads (ideally set to number of cores)
spark://HOST:PORT	connect to a Spark standalone cluster; PORT depends on config (7077 by default)
mesos://HOST:PORT	connect to a Mesos cluster; PORT depends on config (5050 by default)

例子:**Log Mining**

```
lines = spark.textFile("hdfs://...")
#Transformed RDD（转换）
errors = lines.filter(lambda s: s.startswith("ERROR"))
messages = errors.map(lambda s: s.split("\t")[2])
messages.cache()
# count()在这里是Action（执行）
messages.filter(lambda s: "mysql" in s).count()
```

在这个任务中，driver先把task分配到每个worker，worker读入HDFS Block，处理并且缓存数据，然后将结果返回到driver。（缓存数据会得到更快的结果）

Spark程序的生命周期

1. 从外部数据或者来自驱动程序中的集合的createDataFrame来创建DataFrame
2. lazy转换数据到新的DataFrames
3. cache()一些DataFrames以便重复使用
4. 执行actions（操作）以执行并行运算产生结果

RDD: Resilient Distributed Datasets RDD是一种为内存化集群计算设计的容错抽象，提供并行功能转换（map, filter），故障时自动重建。它是Apache Spark中的主要数据抽象 Spark的核心。它支持对元素集合的操作平行。

Resilient：提供容错机制，能够重新计算因为node failure导致的遗失或者毁坏的partition

Distributed: 数据驻留在群集中的多个节点上

Dataset: 对分区元素的收集，例如元组或其他对象（代表使用数据的记录）。

RDD的特性

1. In-Memory: RDD中的数据被尽可能大而且尽可能久地储存在内存中
2. 不可变（只读）: 一旦创建它就不会改变，只能使用转换转换为新的RDD
3. Lazy evaluated: RDD内的数据不可用或转换，直到执行一个触发执行的动作
4. Cacheable: 你可以将所有数据保存在内存种（默认和最优选）或磁盘（最不优选，因为访问速度慢）
5. Parallel: 并行处理数据

6. Typed: RDD种地值有类型，比如RDD[Long]或者RDD[(Int, String)]
7. Partitioned: RDD中的数据被分区（拆分为分区）然后分布在集群中的节点上（每个分区一个JVM，它可能也可能不对应单个节点）

RDD操作

1. Transformation: 返回一个新的RDD

   调用Transformation函数时，没有任何内容被评估，只需要一个RDD并返回一个新的RDD。

   Transformation函数包括map, filter, flatMap, groupByKey, reduceByKey, aggregateByKey, join, etc...

2. Action: 评估并返回新的值 在RDD对象上调用Action函数时，所有数据处理请求在此时计算，并返回结果值。

   Action操作包括reduce, collect, count, first, take, countByKey, foreach, saveAsTextFile, etc...

RDD工作流程

1. 创建RDD： 通过并行现有数据 通过转换现有RDDs 通过储存在HDFS种的文件或者其他存储系统

   从HDFS，文本文件，Amazon S3，Apache HBase，SequenceFiles或者任何Haddop的输入形式，使用

   ```
   sc.parallelize()
   sc.hadoopFile()
   ```

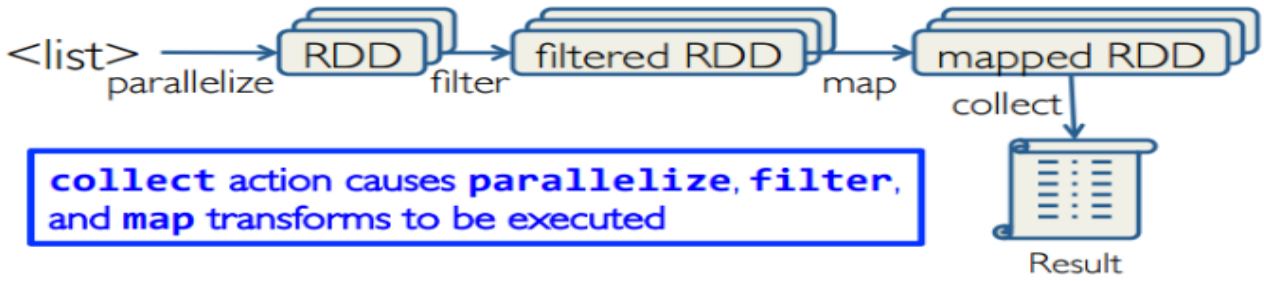
   从文件创造RDD，使用

   ```
   val inputfile = sc.textFile("...", 4)
   ```

   这表示RDD分4个区域，输入按行，lazy evaluation机制下，现在没有任何执行操作发生

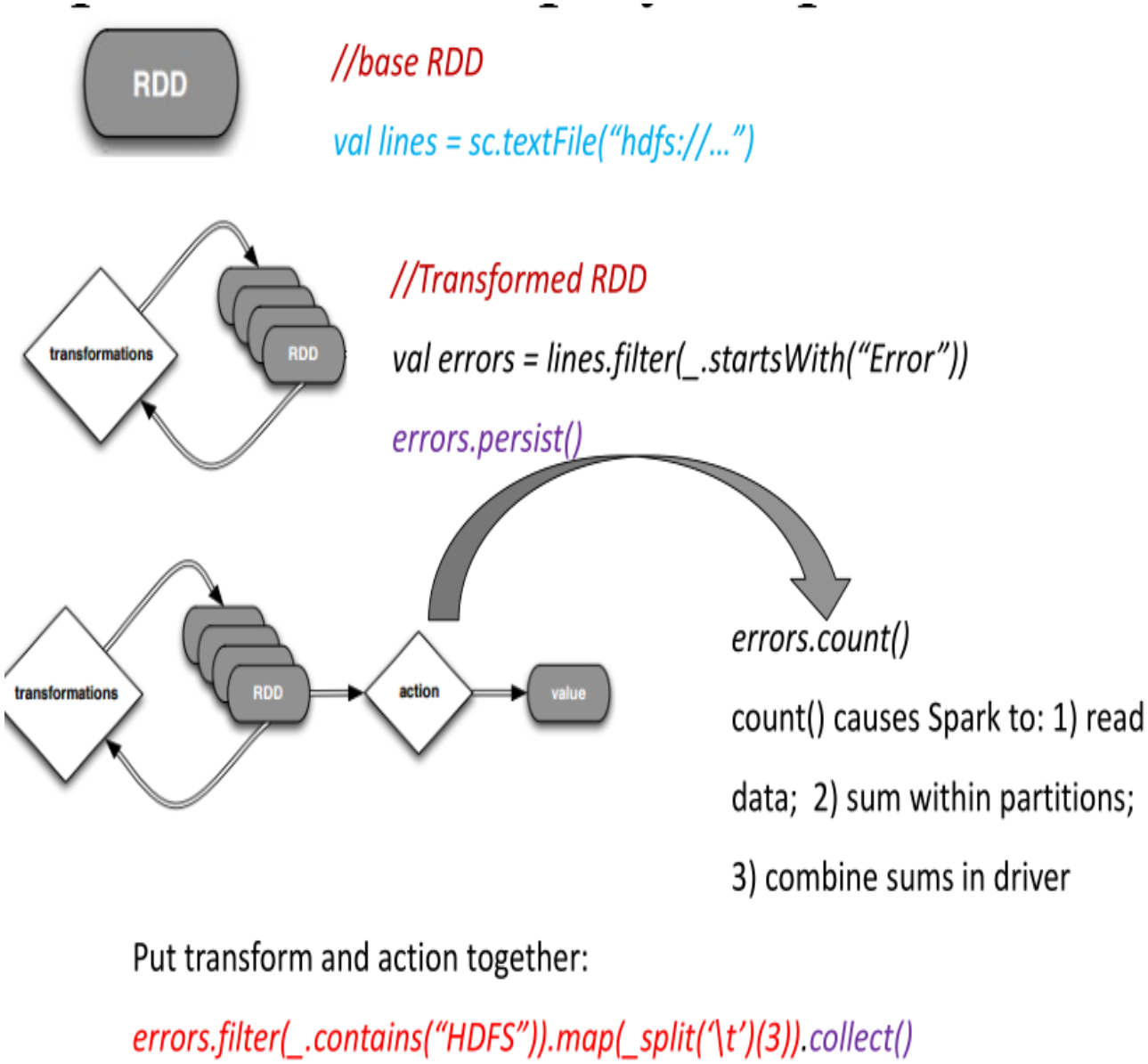
2. 将转transformations函数应用于RDD，如map 从现有数据库中创建新的数据库 使用lazy evaluation，结果不会立刻计算，而是记忆应用在数据上的transformations操作

3. 将actions应用与RDD，如collect 使Spark执行转换操作，Spark的获取结果机制

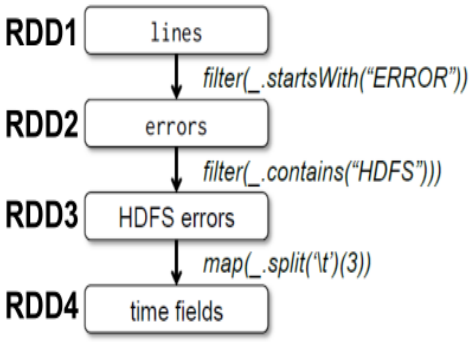


用户能控制其他方面：持久化（persistence）和分区（partition）

RDD工作流程图如下：



谱系图 RDD始终追踪谱系，它有足够的信息来说明如何从中计算出来，它的分区来自稳定存储中的数据。



比如，如果错误分区丢失，Spark会通过仅对其应用过滤器来重建它 相应的行分区。

可以在不同节点上并行重新计算分区，而无需滚动重现整个程序。

SparkContext SparkContext是Spark对Spark应用的进入点，它告诉Spark怎样访问集群一旦SparkContext instance被创建，用户可以用它创建RDD，创建accumulators，创建广播变量，访问Spark服务执行任务。SparkContext本质上是Spark执行的客户端环境，它还能充当Spark应用程序的master。在Spark shell中，一个特殊的解释器感知SparkContext已经为用户创建了一个名为sc的变量

RDD Persistence Spark的一个重要能力是通过操作缓存数据到内存上，每个节点都能存储RDD的任意一个分区，用户能在其他操作中重新使用已缓存的数据集。每个持久化的RDD都可以使用不同的存储级别进行存储，比如： MEMORY_ONLY,将RDD存储为JVM中的反序列化Java对象,如果RDD不没有fit进内存，则某些分区将不会被缓存,而是在需要时重新计算。这是默认级别 MEMORY_AND_DISK，如果RDD没有fit进内存，请存储在不适合磁盘的分区，并在需要时从那里读取它们。 cache() = persist(StorageLevel.MEMORY_ONLY)