Life is short, you need Spark!

从零开始

不需要任何基础，带领您无痛入门 Spark

云计算分布式大数据 Spark 实战高手之路

王家林著

Spark 亚太研究院系列丛书

伴随着大数据相关技术和产业的逐步成熟，继 Hadoop 之后，Spark 技术以其无可比拟的优势，发展迅速，将成为替代 Hadoop 的下一代云计算、大数据核心技术。

本书特点

- 云计算分布式大数据 Spark 实战高手之路三部曲之第一部
- 网络发布版为图文并茂方式，边学习，边演练
- 不需要任何前置知识，从零开始，循序渐进

本书作者

Spark 亚太研究院院长和首席专家，中国目前唯一的移动互联网和云计算大数据集大成者。在 Spark、Hadoop、Android 等方面有丰富的源码、实务和性能优化经验。彻底研究了 Spark 从 0.5.0 到 0.9.1 共 13 个版本的 Spark 源码，并已完成 2014 年 5 月 31 日发布的 Spark1.0 源码研究。

Hadoop 源码级专家，曾负责某知名公司的类 Hadoop 框架开发工作，专注于 Hadoop 一站式解决方案的提供，同时也是云计算分布式大数据处理的最早实践者之一。

Android 架构师、高级工程师、咨询顾问、培训专家。

通晓 Spark、Hadoop、Android、HTML5，迷恋英语播音和健美。

“真相会使你获得自由。”

—— 耶稣《圣经》约翰 8:32KJV

“所有人类的不幸都来源于不能直面事实。”

—— 释迦摩尼

“道法自然”

—— 老子《道德经》第 25 章

《云计算分布式大数据 Spark 实战高手之路》

系列丛书三部曲

《云计算分布式大数据 **Spark** 实战高手之路---从零开始》：

不需要任何基础，带领您无痛入门 Spark 并能够轻松处理 Spark 工程师的日常编程工作，内容包括 Spark 集群的构建、Spark 架构设计、RDD、Shark/SparkSQL、机器学习、图计算、实时流处理、Spark on Yarn、JobServer、Spark 测试、Spark 优化等。

《云计算分布式大数据 **Spark** 实战高手之路---高手崛起》：

大话 Spark 源码，全世界最有情趣的源码解析，过程中伴随诸多实验，解析 Spark 1.0 的任何一句源码！更重要的是，思考源码背后的问题场景和解决问题的设计哲学和实现招式。

《云计算分布式大数据 **Spark** 实战高手之路---高手之巅》：

通过当今主流的 Spark 商业使用方法和最成功的 Hadoop 大型案例让您直达高手之巅，从此一览众山小。

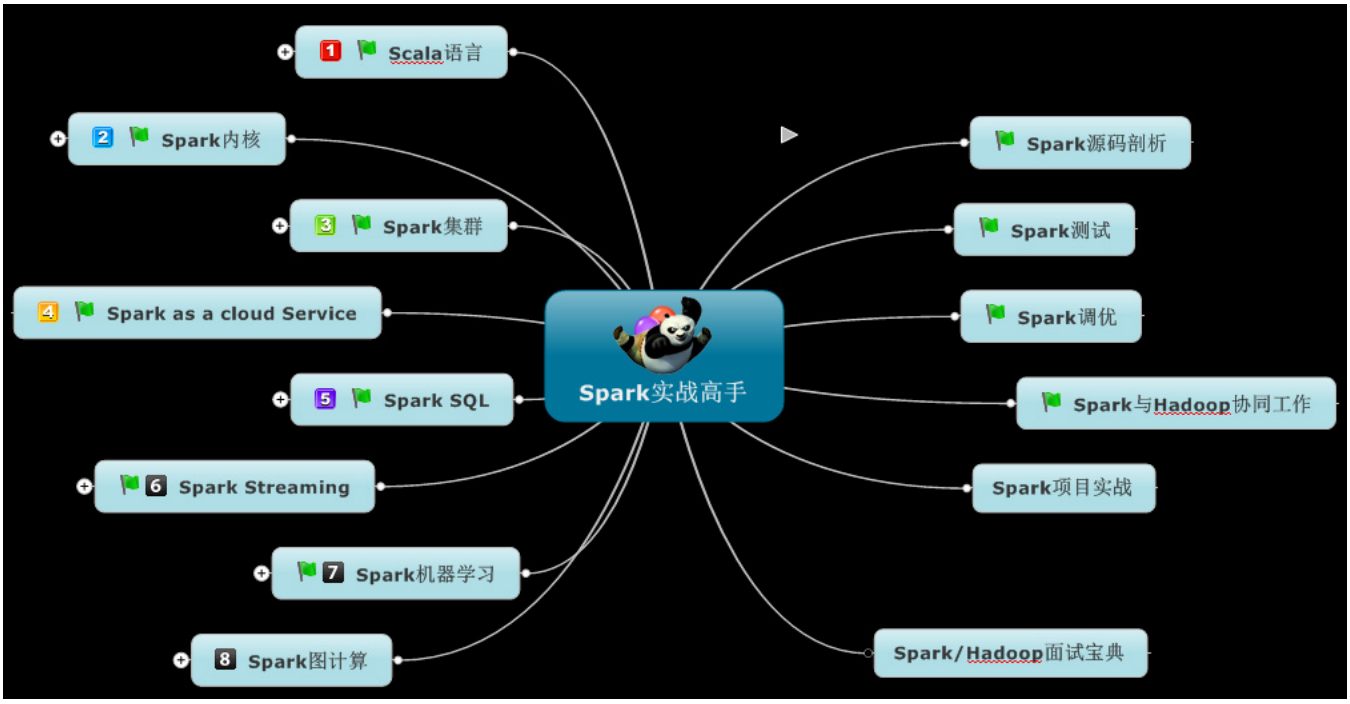
www.sparkinchina.com
TEL: 4006-998-758

《前言》

Spark采用一个统一的技术堆栈解决了云计算大数据的如流处理、图技术、机器学习、NoSQL查询等方面的所有核心问题，具有完善的生态系统，这直接奠定了其一统云计算大数据领域的霸主地位；

要想成为Spark高手，需要经历六大阶段

Spark 实战高手之核心技能点



第一阶段：熟练的掌握Scala语言

1. Spark 框架是采用 Scala 语言编写的，精致而优雅。要想成为 Spark 高手，你就必须阅读 Spark 的源代码，就必须掌握 Scala,；
2. 虽然说现在的 Spark 可以采用多语言 Java、Python 等进行应用程序开发，但是最快速的和支持最好的开发 API 依然并将永远是 Scala 方式的 API，所以你必须掌握 Scala 来编写复杂的和高性能的 Spark 分布式程序；
3. 尤其要熟练掌握 Scala 的 trait、apply、函数式编程、泛型、逆变与协变等；

推荐课程：" 精通Spark的开发语言：Scala最佳实践"

第二阶段：精通Spark平台本身提供给开发者API

1. 掌握Spark中面向RDD的开发模式 掌握各种transformation和action函数的使用；
2. 掌握 Spark 中的宽依赖和窄依赖以及 lineage 机制；
3. 掌握 RDD 的计算流程，例如 Stage 的划分、Spark 应用程序提交给集群的基本过程和 Worker 节点基础的工作原理等

推荐课程： "18 小时内掌握Spark：把云计算大数据速度提高 100 倍以上!"

第三阶段：深入Spark内核

此阶段主要是通过 Spark 框架的源码研读来深入 Spark 内核部分：

1. 通过源码掌握 Spark 的任务提交过程；
2. 通过源码掌握 Spark 集群的任务调度；
3. 尤其要精通 DAGScheduler、TaskScheduler 和 Worker 节点内部的工作的每一步的细节；

推荐课程： “Spark 1.0.0 企业级开发动手：实战世界上第一个Spark 1.0.0 课程，涵盖Spark 1.0.0 所有的企业级开发技术”

第四阶级:掌握基于Spark上的核心框架的使用

Spark 作为云计算大数据时代的集大成者，在实时流处理、图技术、机器学习、NoSQL 查询等方面具有显著的优势，我们使用 Spark 的时候大部分时间都是在使用其上的框架例如 Shark、Spark Streaming 等：

1. Spark Streaming 是非常出色的实时流处理框架 ,要掌握其 DStream、transformation 和 checkpoint 等；
2. Spark 的离线统计分析功能，Spark 1.0.0 版本在 Shark 的基础上推出了 Spark SQL，离线统计分析的功能的效率有显著的提升，需要重点掌握；
3. 对于 Spark 的机器学习和 GraphX 等要掌握其原理和用法；

推荐课程： “Spark企业级开发最佳实践”

第五阶级:做商业级别的Spark项目

通过一个完整的具有代表性的 Spark 项目来贯穿 Spark 的方方面面 ,包括项目的架构设计、用到的技术的剖析、开发实现、运维等，完整掌握其中的每一个阶段和细节，这样就可以让您以后可以从容面对绝大多数 Spark 项目。

推荐课程： “Spark架构案例鉴赏：Conviva、Yahoo！、优酷土豆、网易、腾讯、淘宝等公司的实际Spark案例”

第六阶级：提供Spark解决方案

1. 彻底掌握 Spark 框架源码的每一个细节；
2. 根据不同的业务场景的需要提供 Spark 在不同场景的下的解决方案；
3. 根据实际需要，在 Spark 框架基础上进行二次开发，打造自己的 Spark 框架；

推荐课程： “精通Spark：Spark内核剖析、源码解读、性能优化和商业案例实战”

《第四章：Spark 内核揭秘》

掌握 Spark 内核是精通 Spark 的关键，也是驾驭 Spark 的精髓所在。

基于 Spark 内核，Spark 构建起了一体化多元化的大数据处理流水线，在一个技术堆栈中即可以同时完成批处理、实时流处理、交互式查询、机器学习、图计算以及这些子框架之间数据和 RDD 算子的无缝共享与互操作。

可以说，Spark 内核是每个想彻底掌握 Spark 的人员的必修课，通过对内核的探索，我们对整个 Spark 的运行机制会了如指掌，这对 Spark 的大规模应用、性能优化、系统自定义开发 Spark 系统都是至关重要的。

本章首先带你初探 Spark 内核，接着通过源码游离 Spark 内核中，然后通过源码细致解析 Spark 作业的全生命周期，最后分享 Spark 性能优化，内容组织循序渐进，希望助力诸位 Spark 爱好者能够掌握 Spark 内核。

Spark 内核揭秘共分四个部分：

- 第一部分：Spark 内核初探
- 第二部分：Spark 内核核心源码解析
- 第三部分：Job 全生命周期源码解读
- 第四部分：Spark 性能优化

本讲是 **Spark** 内核揭秘的第一部分：**Spark** 内核初探，具体内容如下所示：

- 1，Spark 内核核心术语解析；
- 2，Spark 集群概览；
- 3，Spark 核心组件；
- 4，Spark 任务调度系统初见；

不需任何前置知识，从零开始，循序渐进，成为 Spark 高手！

目录

一、Spark内核核心术语解析.....8

二、Spark集群概览.....10

三、Spark核心组件.....12

四、Spark任务调度系统初见.....13

一、Spark 内核核心术语解析

Application：

Application 是创建了 SparkContext 实例对象的 Spark 用户，包含了 Driver 程序：

```
/**
 * Main entry point for Spark functionality. A SparkContext represents the connection to a Spark
 * cluster, and can be used to create RDDs, accumulators and broadcast variables on that cluster.
 *
 * @param config a Spark Config object describing the application configuration. Any settings in
 *   this config overrides the default configs as well as system properties.
 */

class SparkContext(config: SparkConf) extends Logging {
```

Spark-shell 是一个应用程序，因为 spark-shell 在启动的时候创建了 SparkContext 对象，其名称为 sc：

```
root@SparkMaster:/usr/local/spark/spark-1.0.2-bin-hadoop2/bin# spark-shell
Spark assembly has been built with Hive, including Datanucleus jars on classpath
Picked up _JAVA_OPTIONS: -Xms512m -Xmx1024m -XX:PermSize=1024m
14/09/08 17:45:25 INFO spark.SecurityManager: Changing view acls to: root
14/09/08 17:45:25 INFO spark.SecurityManager: SecurityManager: authentication di
sabled; ui acls disabled; users with view permissions: Set(root)
14/09/08 17:45:25 INFO spark.HttpServer: Starting HTTP Server
14/09/08 17:45:25 INFO server.Server: jetty-8.y.z-SNAPSHOT
14/09/08 17:45:25 INFO server.AbstractConnector: Started SocketConnector@0.0.0.0
:43747
Welcome to
      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
   /___/ .__/\_,_/_/ /_/\_\   version 1.0.2
      /_/

Using Scala version 2.10.4 (Java HotSpot(TM) Client VM, Java 1.7.0_67)
Type in expressions to have them evaluated.
```

```
Spark context available as sc.

scala> sc
```

Job：

和 Spark 的 action 相对应，每一个 action 例如 count、savaAsTextFile 等都会对应一个 Job 实例，该 Job 实例包含多任务的并行计算。

Driver Program：

运行 main 函数并且新建 SparkContext 实例的程序。

Cluster Manager：

集群资源管理的外部服务，在 Spark 上现在主要有 Standalone、Yarn、Mesos 等三种集群资源管理器，Spark 自带的 Standalone 模式能够满足绝大部分纯粹的 Spark 计算环境中对集群资源管理的需求，基本上只有在集群中运行多套计算框架的时候才建议考虑 Yarn 和 Mesos。

Worker Node：

集群中可以运行应用程序代码的工作节点，相当于 Hadoop 的 slave 节点。

Executor：

在一个 Worker Node 上为应用启动的工作进程，在进程中负责任务的运行，并且负责将数据存放在内存或磁盘上，必须注意的是，每个应用在一个 Worker Node 上只会有一个 Executor，在 Executor 内部通过多线程的方式并发处理应用的任务。

```
/**
 * Spark executor used with Mesos, YARN, and the standalone scheduler.
 */
private[spark] class Executor(
    executorId: String,
    slaveHostname: String,
    properties: Seq[(String, String)],
    isLocal: Boolean = false)
  extends Logging
{
```

Task：

被 Driver 送到 executor 上的工作单元，通常情况下一个 task 会处理一个 split 的数据，每个 split 一般就是一个 Block 块的大小：

```
/**
 * A unit of execution. We have two kinds of Task's in Spark:
 * - [[org.apache.spark.scheduler.ShuffleMapTask]]
 * - [[org.apache.spark.scheduler.ResultTask]]
 *
 * A Spark job consists of one or more stages. The very last stage in a job consists of multiple
 * ResultTasks, while earlier stages consist of ShuffleMapTasks. A ResultTask executes the task
 * and sends the task output back to the driver application. A ShuffleMapTask executes the task
 * and divides the task output to multiple buckets (based on the task's partitioner).
 *
 * @param stageId id of the stage this task belongs to
 * @param partitionId index of the number in the RDD
 */
private[spark] abstract class Task[T](val stageId: Int, var partitionId: Int) extends Serializable {
```

Stage：

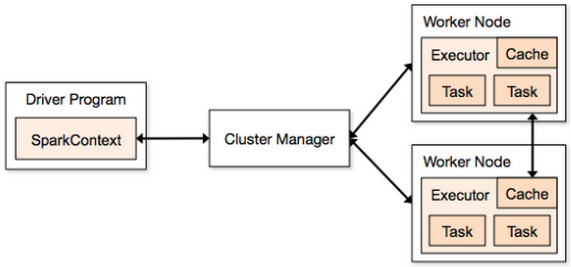
一个 Job 会被拆分成很多任务，每一组任务被成为 Stage，这个 MapReduce 的 map 和 reduce 任务很像，划分 Stage 的依据在于：Stage 开始一般是由于读取外部数据或者 Shuffle 数据、一个 Stage 的结束一般是由于发生 Shuffle (例如 reduceByKey 操作) 或者整个 Job 结束时例如要把数据放到 hdfs 等存储系统上：

```
/**
 * A stage is a set of independent tasks all computing the same function that need to run as part
 * of a Spark job, where all the tasks have the same shuffle dependencies. Each DAG of tasks run
 * by the scheduler is split up into stages at the boundaries where shuffle occurs, and then the
 * DAGScheduler runs these stages in topological order.
 *
 * Each Stage can either be a shuffle map stage, in which case its tasks' results are input for
 * another stage, or a result stage, in which case its tasks directly compute the action that
 * initiated a job (e.g. count(), save(), etc). For shuffle map stages, we also track the nodes
 * that each output partition is on.
 *
 * Each Stage also has a jobId, identifying the job that first submitted the stage.  When FIFO
 * scheduling is used, this allows Stages from earlier jobs to be computed first or recovered
 * faster on failure.
 *
 * The callSite provides a location in user code which relates to the stage. For a shuffle map
 * stage, the callSite gives the user code that created the RDD being shuffled. For a result
 * stage, the callSite gives the user code that executes the associated action (e.g. count()).
 *
 */
private[spark] class Stage(
    val id: Int,
    val rdd: RDD[_],
    val numTasks: Int,
```

二、Spark 集群概览

官方文档对 Spark 集群的初步概述如下所示，这是一个典型的主从结构：

Spark applications run as independent sets of processes on a cluster, coordinated by the SparkContext object in your main program (called the *driver program*). Specifically, to run on a cluster, the SparkContext can connect to several types of *cluster managers* (either Spark's own standalone cluster manager or Mesos/YARN), which allocate resources across applications. Once connected, Spark acquires *executors* on nodes in the cluster, which are processes that run computations and store data for your application. Next, it sends your application code (defined by JAR or Python files passed to SparkContext) to the executors. Finally, SparkContext sends *tasks* for the executors to run.



同时，官方文档对 Spark 集群中的一些关键点也给出了详细的指导：

There are several useful things to note about this architecture:

1. Each application gets its own executor processes, which stay up for the duration of the whole application and run tasks in multiple threads. This has the benefit of isolating applications from each other, on both the scheduling side (each driver schedules its own tasks) and executor side (tasks from different applications run in different JVMs). However, it also means that data cannot be shared across different Spark applications (instances of SparkContext) without writing it to an external storage system.
2. Spark is agnostic to the underlying cluster manager. As long as it can acquire executor processes, and these communicate with each other, it is relatively easy to run it even on a cluster manager that also supports other applications (e.g. Mesos/YARN).
3. Because the driver schedules tasks on the cluster, it should be run close to the worker nodes, preferably on the same local area network. If you'd like to send requests to the cluster remotely, it's better to open an RPC to the driver and have it submit operations from nearby than to run a driver far away from the worker nodes.

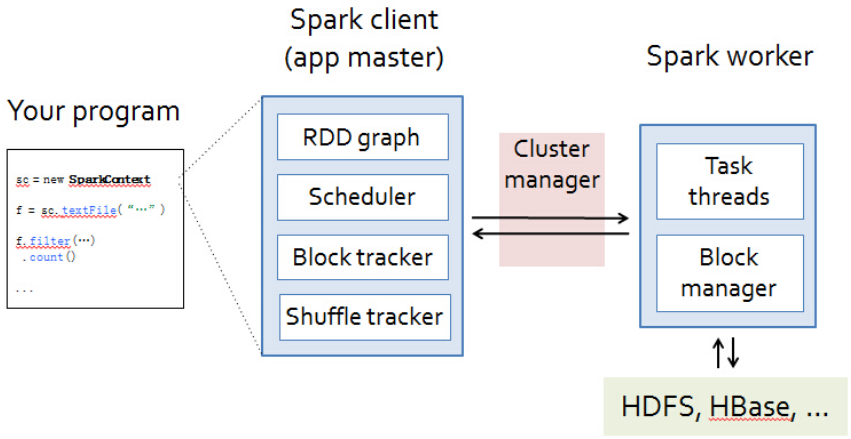
其中 Worker 的定义如下所示：

```
/**
  * @param masterUrls Each url should look like spark://host:port.
  */
private[spark] class Worker(
    host: String,
    port: Int,
    webUiPort: Int,
    cores: Int,
    memory: Int,
    masterUrls: Array[String],
    actorSystemName: String,
    actorName: String,
    workDirPath: String = null,
    val conf: SparkConf,
    val securityMgr: SecurityManager)
  extends Actor with ActorLogReceive with Logging {
  import context.dispatcher
```

需要注意的是Spark Driver所在的机器需要和Spark集群最好位于同一个网络环境中，因为 Driver 中的 SparkContext 实例要要发送任务给不同 Worker Node 的 Executor 并接受 Executor 的一些执行结果信息，一般而言，在企业实际的生产环境中 Driver 所在机器的配置往往都是比较不错的，尤其是其 CPU 的处理能力往往都很强悍。

三、Spark 核心组件

Spark 核心组件如下所示：



在初始话 SparkContext 的时候会初始化一系列的内容，例如会查看内存的设置情况：

```
private[spark] val executorMemory = conf.getOption("spark.executor.memory")
  .orElse(Option(System.getenv("SPARK_EXECUTOR_MEMORY")))
  .orElse(Option(System.getenv("SPARK_MEM")).map(warnSparkMem))
  .map(Utils.memoryStringToMb)
  .getOrElse(512)
```

SparkContext 在初始化的时候也会创建和启动 scheduler：

```
// Create and start the scheduler
private[spark] var taskScheduler = SparkContext.createTaskScheduler(this, master)
private val heartbeatReceiver = env.actorSystem.actorOf(
  Props(new HeartbeatReceiver(taskScheduler)), "HeartbeatReceiver")
@volatile private[spark] var dagScheduler: DAGScheduler = _
try {
  dagScheduler = new DAGScheduler(this)
} catch {
  case e: Exception => throw
    new SparkException("DAGScheduler cannot be initialized due to %s".format(e.getMessage))
}

// start TaskScheduler after taskScheduler sets DAGScheduler reference in DAGScheduler's
// constructor
taskScheduler.start()
```

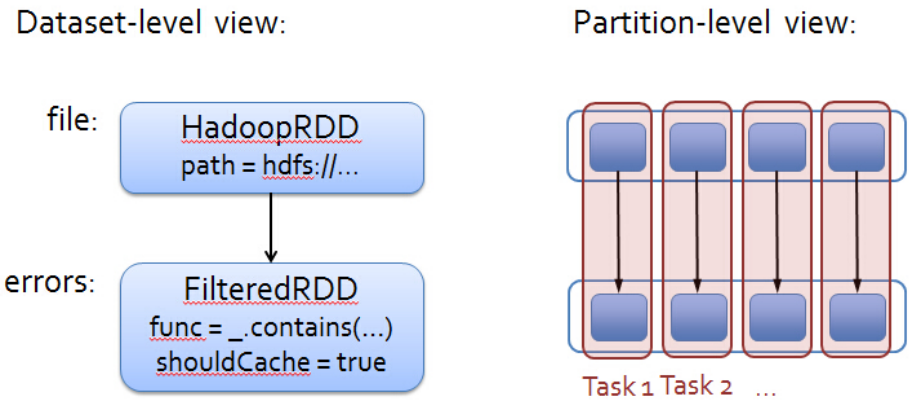
集群核心组件中的 Block tracker 是用于 Block 和 partition 对应关系的管理器。

集群核心组件中的 Shuffle tracker 是用于记录 Shuffle 操作过程细节的。

从集群中也可以清晰的看出，Executor 在执行任务的时候是采用多线程的方式执行的

并能够在 HDFS 或者 HBase 等系统上存取数据。

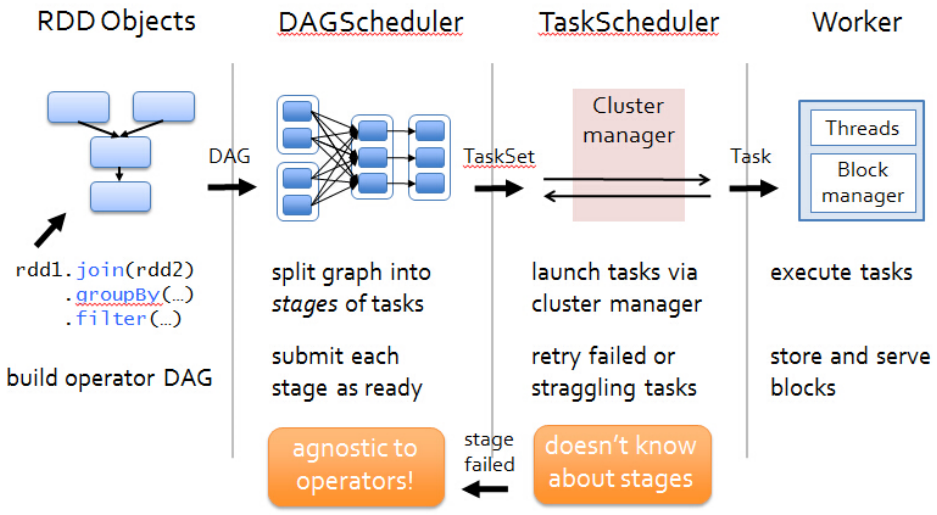
在实际的 Driver Program 运行的时候每个 Partition 都会由一个 Task 负责运行：



也就是说有多少 partition 就会有多少 task 在运行，而这些 task 是并发的运行在 executor 中的。

四、Spark 任务调度系统初见

Spark 的任务调度系统如下所示：



从上图中可以看出由 RDD Objects 产生 DAG，然后进入了 DAGScheduler 阶段，DAGScheduler 是面向 Stage 的高层次的调度器，DAGScheduler 把 DAG 拆分成很多的 Tasks，每组的 tasks 都是一个 Stage，每当遇到 Shuffle 就会产生新的 Stage，可以看出

上图一共有三个 stage；DAGScheduler 需要记录那些 RDD 被存入磁盘等物化动作，同时要寻求 Task 的最优化调度，例如数据本地性等；DAGScheduler 还要监视因为 Shuffle 输出导致的失败，如果发现这种失败，可能就要重新提交 该 Stage：

```
/**
 * The high-level scheduling layer that implements stage-oriented scheduling. It computes a DAG of
 * stages for each job, keeps track of which RDDs and stage outputs are materialized, and finds a
 * minimal schedule to run the job. It then submits stages as TaskSets to an underlying
 * TaskScheduler implementation that runs them on the cluster.
 *
 * In addition to coming up with a DAG of stages, this class also determines the preferred
 * locations to run each task on, based on the current cache status, and passes these to the
 * low-level TaskScheduler. Furthermore, it handles failures due to shuffle output files being
 * lost, in which case old stages may need to be resubmitted. Failures *within* a stage that are
 * not caused by shuffle file loss are handled by the TaskScheduler, which will retry each task
 * a small number of times before cancelling the whole stage.
 *
 */
private[spark]
class DAGScheduler(
    private[scheduler] val sc: SparkContext,
    private[scheduler] val taskScheduler: TaskScheduler,
    listenerBus: LiveListenerBus,
    mapOutputTracker: MapOutputTrackerMaster,
    blockManagerMaster: BlockManagerMaster,
    env: SparkEnv,
    clock: Clock = SystemClock)
  extends Logging {
```

DAGScheduler 划分 Stage 后以 TaskSet 为单位把任务交给低层次的可插拔的调度器 TaskScheduler 来处理：

```
/**
 * Low-level task scheduler interface, currently implemented exclusively by TaskSchedulerImpl.
 * This interface allows plugging in different task schedulers. Each TaskScheduler schedulers tasks
 * for a single SparkContext. These schedulers get sets of tasks submitted to them from the
 * DAGScheduler for each stage, and are responsible for sending the tasks to the cluster, running
 * them, retrying if there are failures, and mitigating stragglers. They return events to the
 * DAGScheduler.
 */
private[spark] trait TaskScheduler {

  def rootPool: Pool

  def schedulingMode: SchedulingMode

  def start(): Unit
```

可以看出 TaskScheduler 是一个 trait ,在目前的 Spark 系统中 TaskScheduler 的实现类只有一个 TaskSchedulerImpl：

```scala
/**
 * Schedules tasks for multiple types of clusters by acting through a SchedulerBackend.
 * It can also work with a local setup by using a LocalBackend and setting isLocal to true.
 * It handles common logic, like determining a scheduling order across jobs, waking up to launch
 * speculative tasks, etc.
 *
 * Clients should first call initialize() and start(), then submit task sets through the
 * runTasks method.
 *
 * THREADING: SchedulerBackends and task-submitting clients can call this class from multiple
 * threads, so it needs locks in public API methods to maintain its state. In addition, some
 * SchedulerBackends synchronize on themselves when they want to send events here, and then
 * acquire a lock on us, so we need to make sure that we don't try to lock the backend while
 * we are holding a lock on ourselves.
 */
private[spark] class TaskSchedulerImpl(
    val sc: SparkContext,
    val maxTaskFailures: Int,
    isLocal: Boolean = false)
  extends TaskScheduler with Logging
{
  def this(sc: SparkContext) = this(sc, sc.conf.getInt("spark.task.maxFailures", 4))
```

一个 TaskScheduler 只为一个 SparkContext 实例服务，TaskScheduler 接受来自 DAGScheduler 发送过来的分组的任务，DAGScheduler 给 TaskScheduler 发送任务的时候是以 Stage 为单位来提交的，TaskScheduler 收到任务后负责把任务分发到集群中 Worker 的 Executor 中去运行，如果某个 task 运行失败，TaskScheduler 要负责重试；另外如果 TaskScheduler 发现某个 Task 一直未运行完，就可能启动同样的任务运行同一个 Task，那个任务先运行完就用哪个任务的结果。

TaskScheduler 发送的任务交给了 Worker 上的 Executor 以多线程的方式运行，每一个线程负责一个任务：

```scala
/**
  * @param masterUrls Each url should look like spark://host:port.
  */
private[spark] class Worker(
    host: String,
    port: Int,
    webUiPort: Int,
    cores: Int,
    memory: Int,
    masterUrls: Array[String],
    actorSystemName: String,
    actorName: String,
    workDirPath: String = null,
    val conf: SparkConf,
    val securityMgr: SecurityManager)
  extends Actor with ActorLogReceive with Logging {
  import context.dispatcher

  Utils.checkHost(host, "Expected hostname")
  assert (port > 0)
```

```scala
/**
 * Spark executor used with Mesos, YARN, and the standalone scheduler.
 */
private[spark] class Executor(
    executorId: String,
    slaveHostname: String,
    properties: Seq[(String, String)],
    isLocal: Boolean = false)
  extends Logging
{
  // Application dependencies (added through SparkContext) that we've fetched so far on this node.
  // Each map holds the master's timestamp for the version of that file or JAR we got.
  private val currentFiles: HashMap[String, Long] = new HashMap[String, Long]()
  private val currentJars: HashMap[String, Long] = new HashMap[String, Long]()

  private val EMPTY_BYTE_BUFFER = ByteBuffer.wrap(new Array[Byte](0))

  @volatile private var isStopped = false
```

其中的存储系统的管理是 BlockManager 来负责的：

```scala
private[spark] class BlockManager(
    executorId: String,
    actorSystem: ActorSystem,
    val master: BlockManagerMaster,
    defaultSerializer: Serializer,
    maxMemory: Long,
    val conf: SparkConf,
    securityManager: SecurityManager,
    mapOutputTracker: MapOutputTracker,
    shuffleManager: ShuffleManager)
  extends Logging {

  private val port = conf.getInt("spark.blockManager.port", 0)
  val shuffleBlockManager = new ShuffleBlockManager(this, shuffleManager)
  val diskBlockManager = new DiskBlockManager(shuffleBlockManager,
    conf.get("spark.local.dir", System.getProperty("java.io.tmpdir")))
  val connectionManager =
    new ConnectionManager(port, conf, securityManager, "Connection manager for block manager")

  implicit val futureExecContext = connectionManager.futureExecContext
```

下面看一下 TaskSet 的源码：

```scala
/**
 * A set of tasks submitted together to the low-level TaskScheduler, usually representing
 * missing partitions of a particular stage.
 */
private[spark] class TaskSet(
    val tasks: Array[Task[_]],
    val stageId: Int,
    val attempt: Int,
    val priority: Int,
    val properties: Properties) {
    val id: String = stageId + "." + attempt

  def kill(interruptThread: Boolean) {
    tasks.foreach(_.kill(interruptThread))
  }

  override def toString: String = "TaskSet " + id
}
```

从 TaskSet 源码的第一个参数 tasks 就可以看出其是一个 Task 的数组 ,包含一组 Task。

Spark 亚太研究院

Spark 亚太研究院，提供 Spark、Hadoop、Android、Html5、云计算和移动互联网一站式解决方案。以帮助企业规划、部署、开发、培训和使用为核心，并规划和实施人才培训完整路径，提供源码研究和应用技术训练。

近期活动及相关课程

1、决战云计算大数据时代 Spark 亚太研究院 100 期公益大奖堂

每周四晚上 20：00—21:00
课程介绍：http://edu.51cto.com/course/course_id-1659.html#showDesc
报名参与：http://ke.qq.com/cgi-bin/courseDetail?course_id=6167

2、大数据 Spark 实战高手之路—熟练掌握 Scala 语言视频课程

国内第一个 Scala 视频学习课程！
成为 Spark 高手必备技能，必修课程！
现在购买，即可享受套餐优惠！

课程地址： http://edu.51cto.com/pack/view/id-124.html

近期公开课：

《决胜大数据时代：Hadoop、Yarn、Spark 企业级最佳实践》

集大数据领域最核心三大技术：Hadoop 方向 50%：掌握生产环境下、源码级别下的 Hadoop 经验，解决性能、集群难点问题；Yarn 方向 20%：掌握最佳的分布式集群资源管理框架，能够轻松使用 Yarn 管理 Hadoop、Spark 等；Spark 方向 30%：未来统一的大数据框架平台，剖析 Spark 架构、内核等核心技术，对未来转向 SPARK 技术，做好技术储备。课程内容落地性强，即解决当下问题，又有助于驾驭未来。
开课时间：9 月 26—28 日 上海、10 月 26—28 日北京、11 月 1—3 日深圳

咨询电话：4006-998-758

QQ 交流群： 1 群：317540673 （已满）
2 群 297931500

微信公众号：spark-china