

分布式计算框架

范颖捷 | 2018年7月

目录 > CONTENTS

- 1 MapReduce
- 2 Spark

The background of the slide features a complex network diagram. It consists of numerous small, dark grey circular nodes connected by thin, light grey lines. These connections form a dense web of triangles and other geometric shapes, creating a sense of interconnectedness and data flow. The overall aesthetic is technical and modern, typical of a presentation on distributed computing or data science.

1 chapter

MapReduce

- ✓ MR简介
- ✓ MR原理
- ✓ 作业管理

➤ 起源

- 2004年10月Google发表了MapReduce论文
- 设计初衷：解决搜索引擎中大规模网页数据的并行处理
- Hadoop MapReduce是Google MapReduce的开源实现
- MapReduce是Apache Hadoop的核心子项目

➤ 概念

- 面向批处理的分布式计算框架
- 一种编程模型：MapReduce程序被分为Map（映射）阶段和Reduce（化简）阶段

➤ 核心思想

- 分而治之，并行计算
- 移动计算，而非移动数据



➤ 特点

- 计算跟着数据走
- 良好的扩展性：计算能力随着节点数增加，近似线性递增
- 高容错
- 状态监控
- 适合海量数据的离线批处理
- 降低了分布式编程的门槛

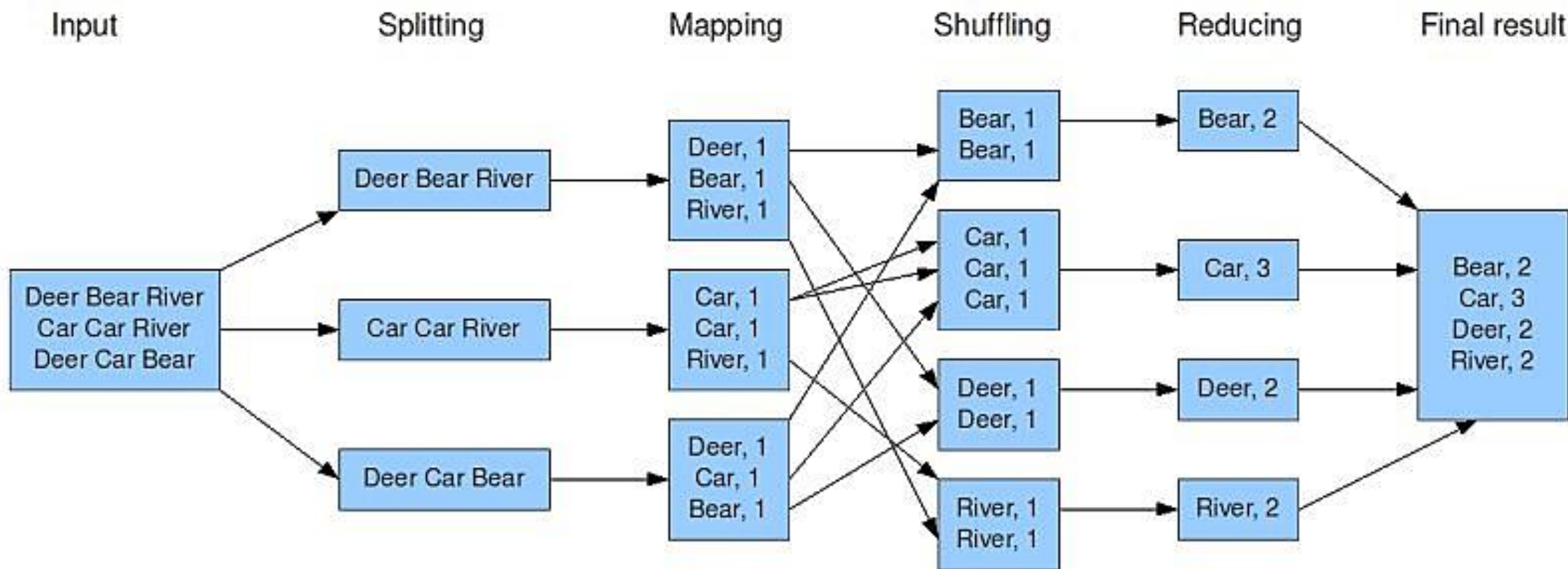
➤ 适用场景

- 数据统计，如：网站的PV、UV统计
- 搜索引擎构建索引
- 海量数据查询
- 复杂数据分析算法实现

➤ 不适用场景

- OLAP
 - 要求毫秒或秒级返回结果
- 流计算
 - 流计算的输入数据集是动态的，而MapReduce是静态的
- DAG计算
 - 多个作业存在依赖关系，后一个的输入是前一个的输出，构成有向无环图DAG
 - 每个MapReduce作业的输出结果都会落盘，造成大量磁盘IO，导致性能非常低下

➤ 示例：WordCount



➤ Job & Task (作业与任务)

- 作业是客户端请求执行的一个工作单元
 - 包括输入数据、MapReduce程序、配置信息
- 任务是将作业分解后得到的细分工作单元
 - 分为Map任务和Reduce任务

➤ Split (切片)

- 输入数据被划分成等长的小数据块，称为输入切片（Input Split），简称切片
- Split是逻辑概念，仅包含元数据信息，如：数据的起始位置、长度、所在节点等
- 每个Split交给一个Map任务处理，Split的数量决定Map任务的数量
- Split的划分方式由程序设定，Split与HDFS Block没有严格的对应关系
- Split的大小默认等于Block大小
- Split越小，负载越均衡，但集群的开销越大

➤ Map阶段（映射）

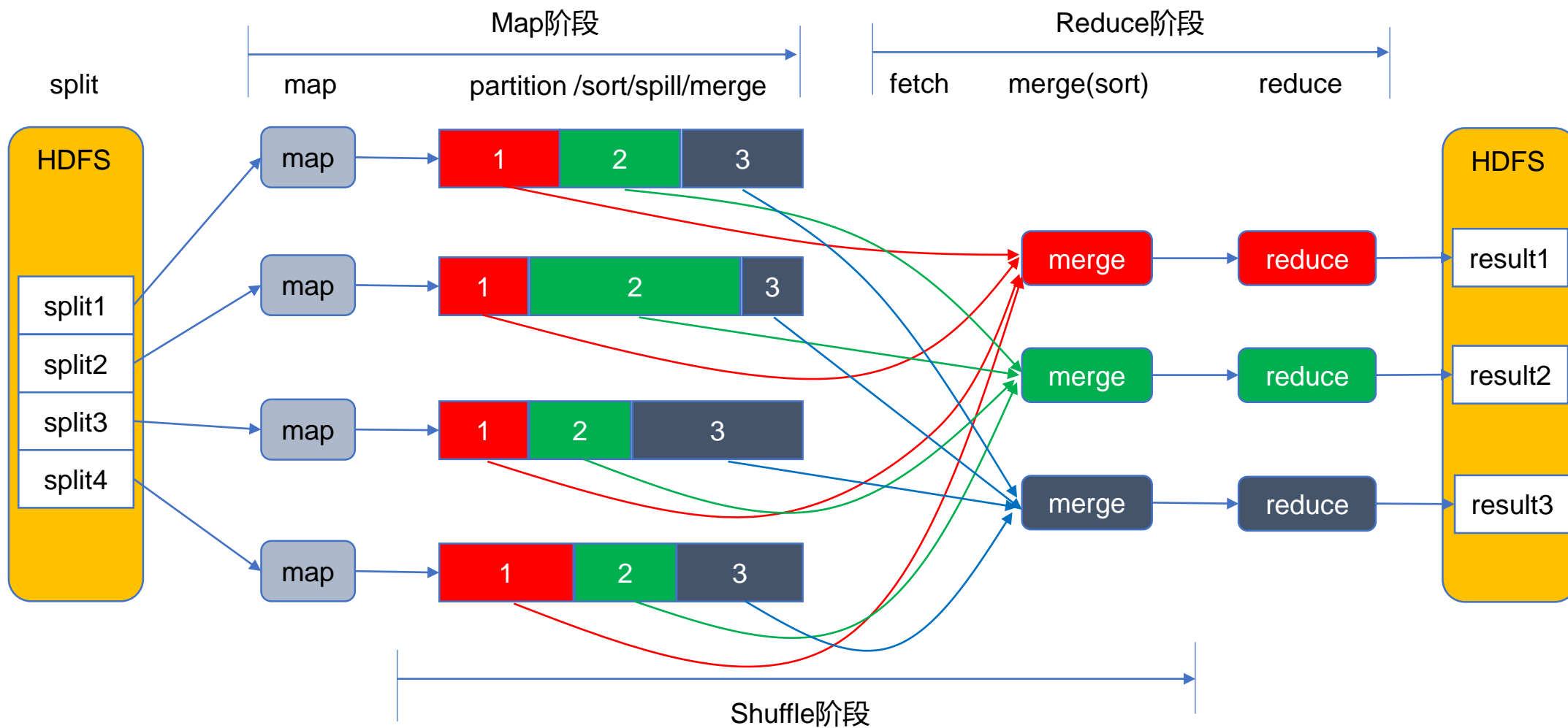
- 由若干Map任务组成，任务数量由Split数量决定
- 输入：Split切片（key-value），输出：中间计算结果（key-value）

➤ Reduce阶段（化简）

- 由若干Reduce任务组成，任务数量由程序指定
- 输入：Map阶段输出的中间结果（key-value），输出：最终结果（key-value）

➤ Shuffle阶段（洗牌）

- Map、Reduce阶段的中间环节，负责执行Partition（分区）、Sort（排序）、Spill（溢写）、Merge（合并）、抓取（Fetch）等工作
- Partition决定了Map任务输出的每条数据放入哪个分区，交给哪个Reduce任务处理
- Reduce任务的数量决定了Partition数量
- Partition编号 = Reduce任务编号 = “key hashCode % reduce task number”
- 避免和减少Shuffle是MapReduce程序调优的重点



➤ Shuffle详解

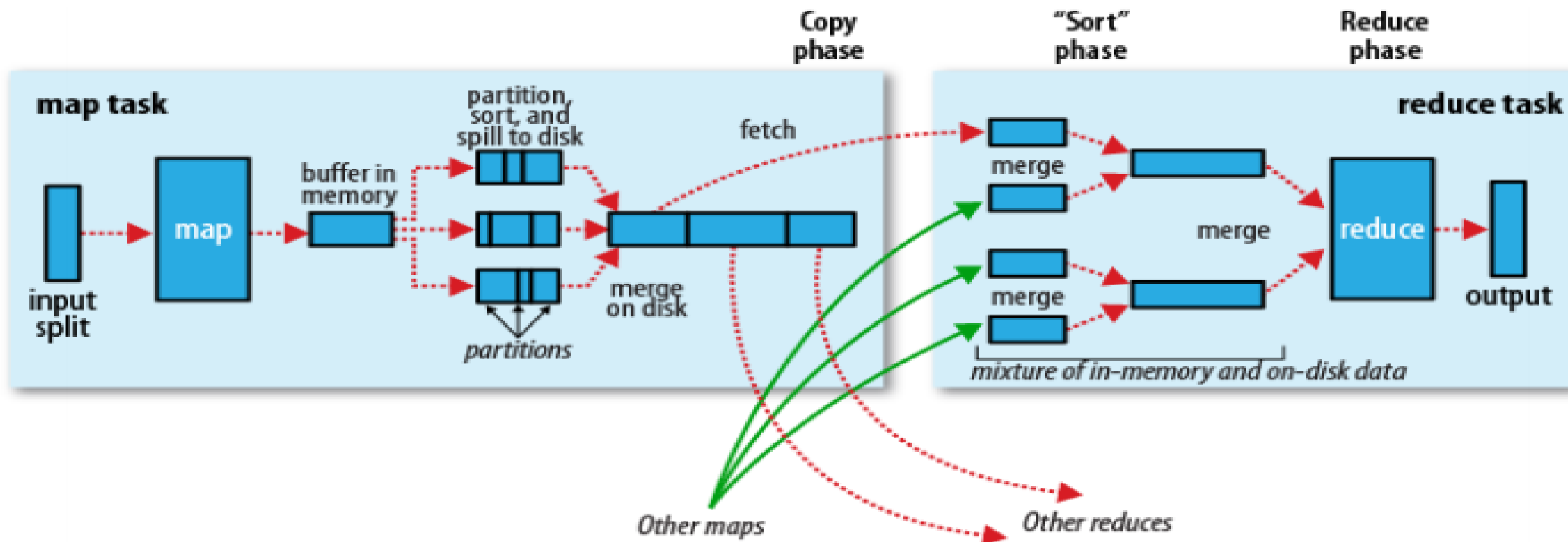
• Map端

- Map任务将中间结果写入专用内存缓冲区Buffer（默认100M），同时进行Partition和Sort（先按“key hashCode % reduce task number”对数据进行分区，分区内再按key排序）
- 当Buffer的数据量达到阈值（默认80%）时，将数据溢写（Spill）到磁盘的一个临时文件中，文件内数据先分区后排序
- Map任务结束前，将多个临时文件合并（Merge）为一个Map输出文件，文件内数据先分区后排序

• Reduce端

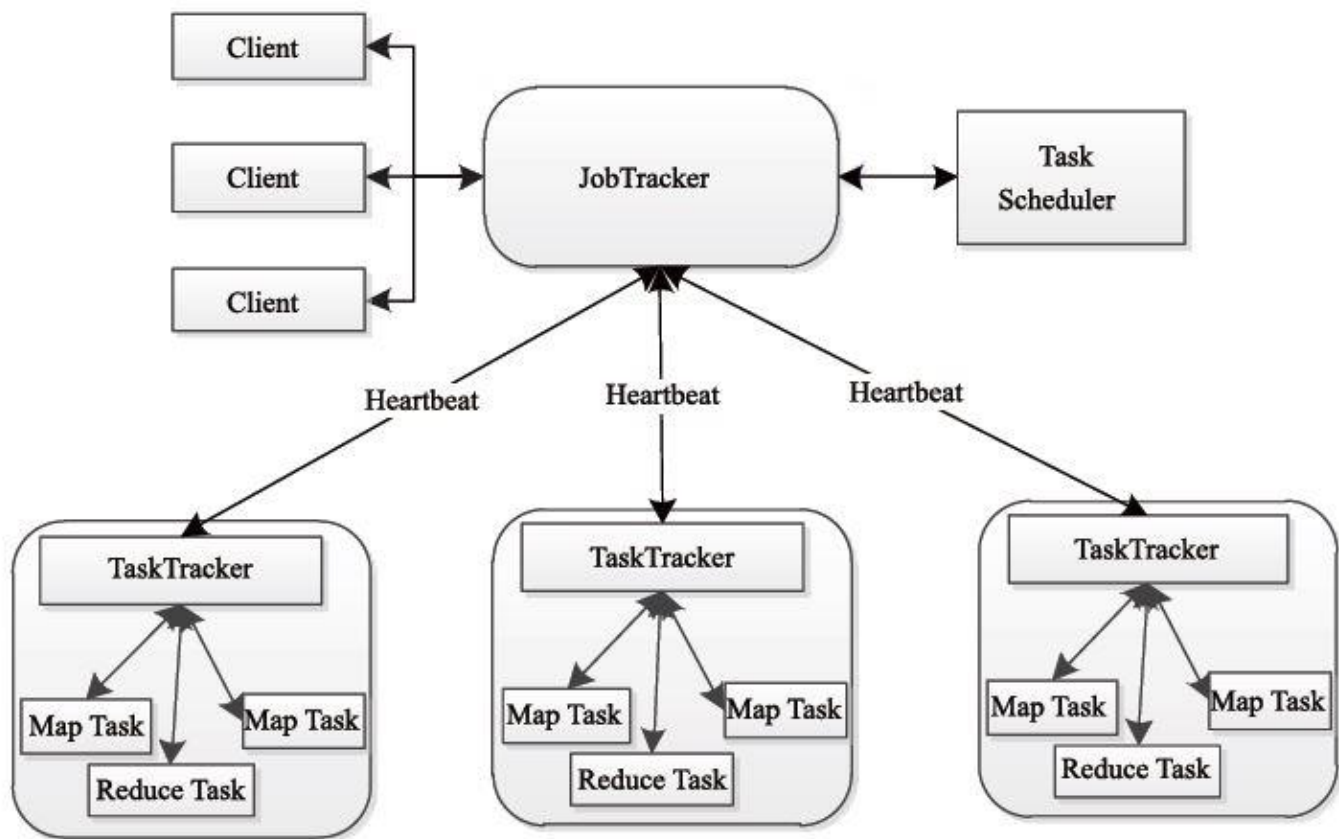
- Reduce任务从多个Map输出文件中主动抓取（Fetch）属于自己的分区数据，先写入Buffer，数据量达到阈值后，溢写到磁盘的一个临时文件中
- 数据抓取完成后，将多个临时文件合并为一个Reduce输入文件，文件内数据按key排序

➤ Shuffle详解

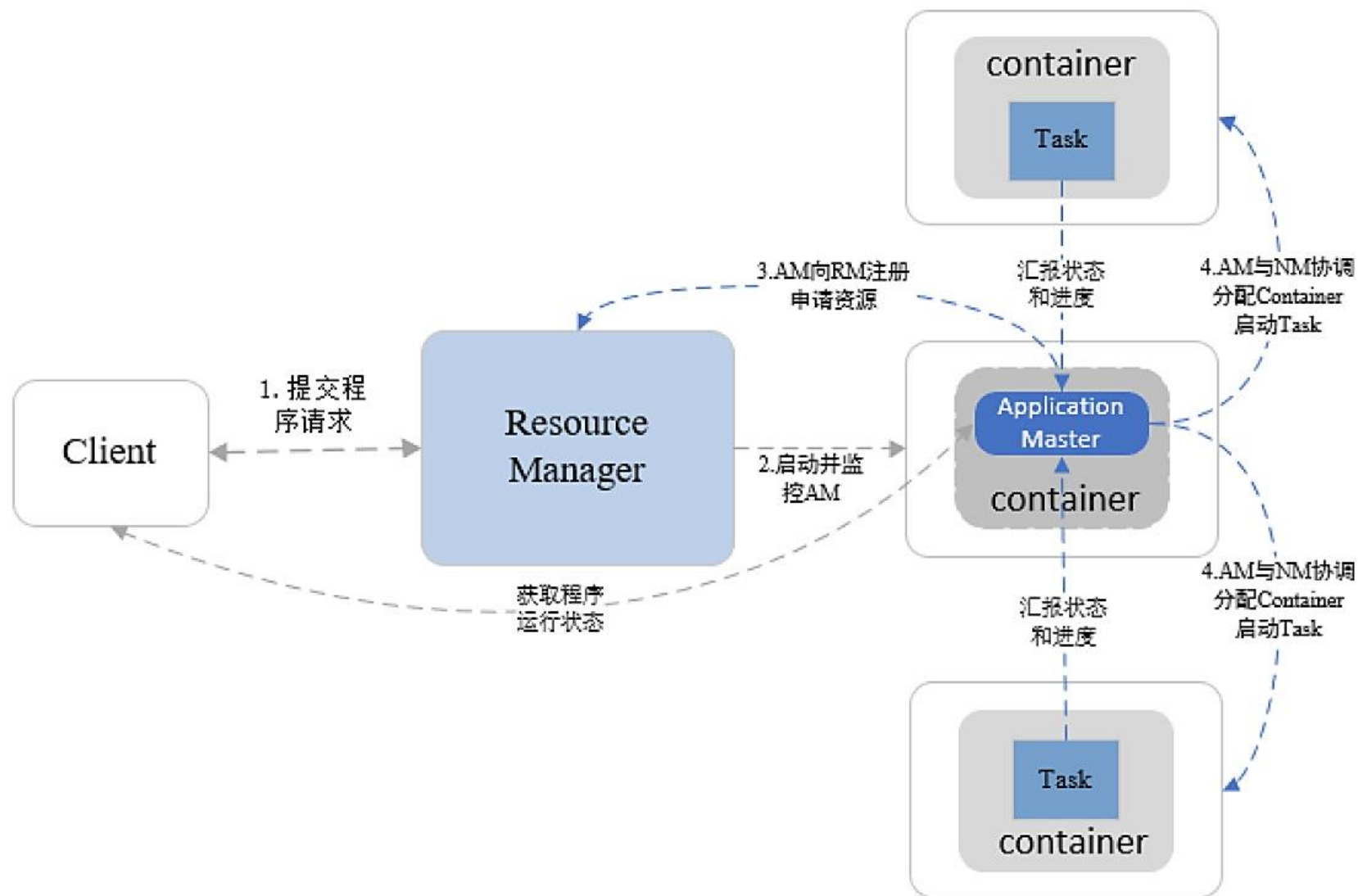


➤ JobTracker/TaskTracker模式 (Hadoop 1.X)

- JobTracker节点 (Master)
 - 调度任务在TaskTracker上运行
 - 若任务失败，指定新TaskTracker重新运行
- TaskTracker节点 (Slave)
 - 执行任务，发送进度报告
- 存在的问题
 - JobTracker存在单点故障
 - JobTracker负载太重（上限4000节点）
 - JobTracker缺少对资源的全面管理
 - TaskTracker对资源的描述过于简单
 - 源码很难理解



➤ YARN模式 (Hadoop 2.X)



➤ 提交作业

```
# hadoop jar {jarFile} [mainClass] args
```

- jarFile: MapReduce运行程序的jar包
- mainClass: jar包中main函数所在类的类名
- args: 程序调用需要的参数，如：输入输出路径

➤ 查看作业

```
# sudo -u yarn application -list
```

➤ 终止作业

```
# sudo -u yarn application -kill {application_id}
```

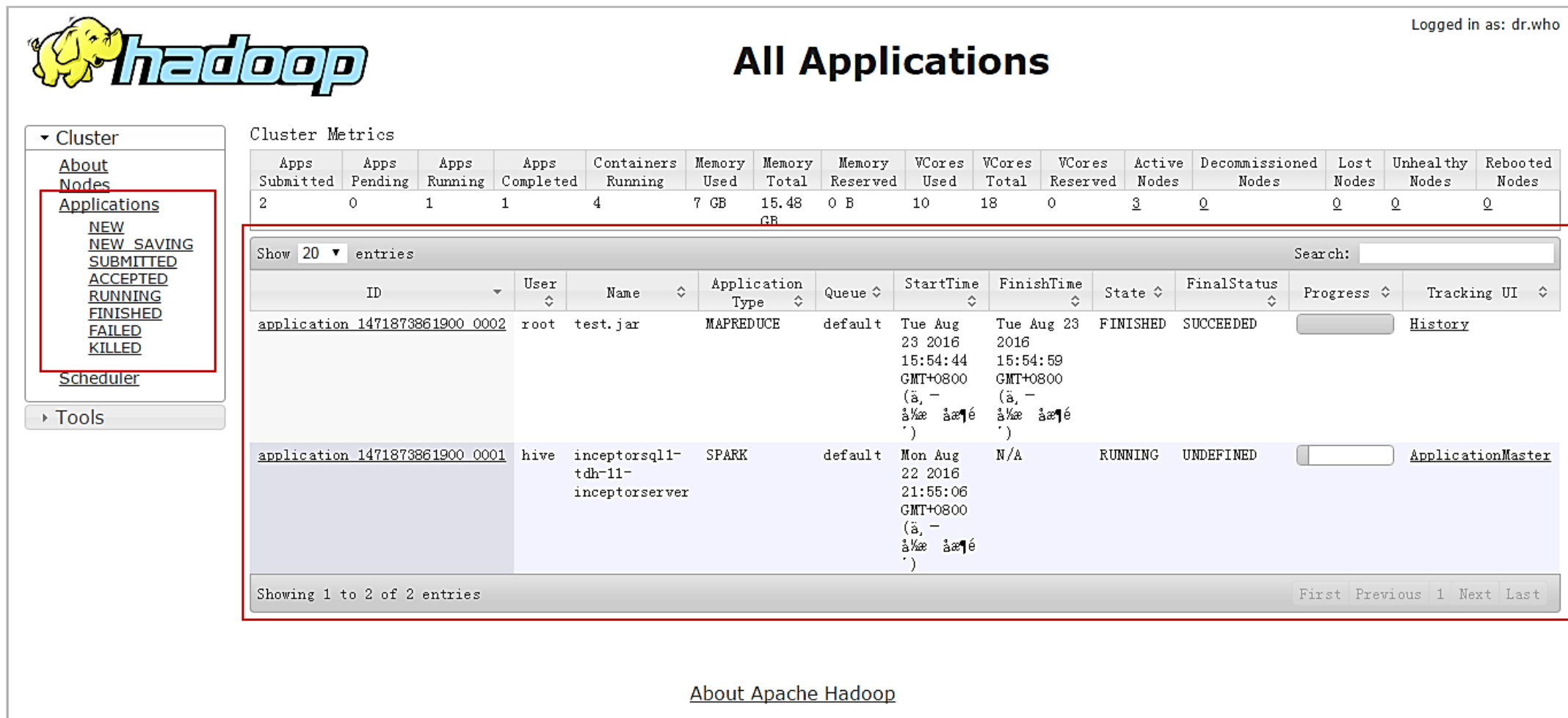
➤ 示例：提交作业

```
# hadoop jar /usr/lib/hadoop-mapreduce/hadoop-mapreduce-example.jar pi 10 10
```

```
t3126poc4:~ # hadoop jar /usr/lib/hadoop-mapreduce/hadoop-mapreduce-examples.jar pi 10 10
Number of Maps = 10
Samples per Map = 10
Wrote input for Map #0
Wrote input for Map #1
Wrote input for Map #2
Wrote input for Map #3
Wrote input for Map #4
Wrote input for Map #5
Wrote input for Map #6
Wrote input for Map #7
Wrote input for Map #8
Wrote input for Map #9
Starting Job
2016-05-10 14:09:58,250 INFO client.RMPProxy: Connecting to ResourceManager at t3126poc5/172.16.2.85:8032
2016-05-10 14:09:58,834 INFO input.FileInputFormat: Total input paths to process : 10
2016-05-10 14:09:58,915 INFO mapreduce.JobSubmitter: number of splits:10
2016-05-10 14:09:59,188 INFO mapreduce.JobSubmitter: Submitting tokens for job: job_1462786145119_0002
2016-05-10 14:09:59,453 INFO impl.YarnClientImpl: Submitted application application_1462786145119_0002
2016-05-10 14:09:59,498 INFO mapreduce.Job: The url to track the job: http://t3126poc5:8088/proxy/application_1462786145119_0002/
2016-05-10 14:09:59,499 INFO mapreduce.Job: Running job: job_1462786145119_0002
2016-05-10 14:10:05,641 INFO mapreduce.Job: Job job_1462786145119_0002 running in uber mode : false
2016-05-10 14:10:05,644 INFO mapreduce.Job: map 0% reduce 0%
```

➤ 作业监控

- Web监控: `http://{AM_IP}:8088/proxy/{application_id}/`



Logged in as: dr.who

hadoop

All Applications

Cluster Metrics

Apps Submitted	Apps Pending	Apps Running	Apps Completed	Containers Running	Memory Used	Memory Total	Memory Reserved	VCores Used	VCores Total	VCores Reserved	Active Nodes	Decommissioned Nodes	Lost Nodes	Unhealthy Nodes	Rebooted Nodes
2	0	1	1	4	7 GB	15.48 GB	0 B	10	18	0	3	0	0	0	0

Cluster Metrics

Show 20 entries

Search:

ID	User	Name	Application Type	Queue	StartTime	FinishTime	State	FinalStatus	Progress	Tracking UI
application 1471873861900 0002	root	test.jar	MAPREDUCE	default	Tue Aug 23 2016 15:54:44 GMT+0800 (ä, - ä%æ äæ'é')	Tue Aug 23 2016 15:54:59 GMT+0800 (ä, - ä%æ äæ'é')	FINISHED	SUCCEEDED	<div></div>	History
application 1471873861900 0001	hive	inceptorserver	SPARK	default	Mon Aug 22 2016 21:55:06 GMT+0800 (ä, - ä%æ äæ'é')	N/A	RUNNING	UNDEFINED	<div></div>	ApplicationMaster

Showing 1 to 2 of 2 entries


First Previous 1 Next Last

[About Apache Hadoop](#)

➤ 作业诊断

- 配置参数: yarn.nodemanager.log-dirs
 - MapReduce运行日志目录
 - 默认值为/mnt/disk*/hadoop/yarn/
- 根据运行出错信息, 可以到指定节点下分析日志

```
t3126poc5:~ # ls /mnt/disk2/hadoop/yarn/logs/application_1462783245088_0002/container_1462783245088_0002_01_000002/  
stderr stdout syslog
```

2

chapter

Spark

- ✓ Spark简介
- ✓ Spark原理
- ✓ 任务监控

➤ MapReduce有较大的局限性

- 仅支持Map、Reduce两种语义操作
- 执行效率低，时间开销大
- 主要用于大规模离线批处理
- 不适合迭代计算、交互式计算、实时流处理等场景

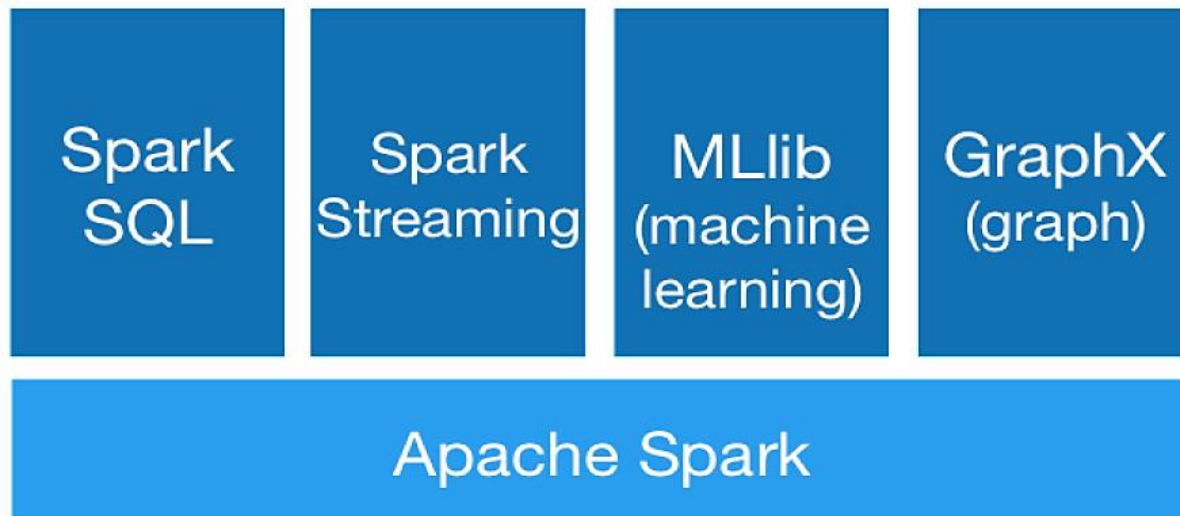
➤ 计算框架种类多，选型难，学习成本高

- 批处理：MapReduce
- 流处理：Storm、Flink
- 交互式计算：Impala、Presto
- 机器学习：Mahout

➤ 统一计算框架，简化技术选型

- 在一个统一框架下，实现批处理、流处理、交互式计算、机器学习

- 由加州大学伯克利分校的AMP实验室开源
- 大规模分布式通用计算引擎
 - Spark Core：核心计算框架
 - Spark SQL：结构化数据查询
 - Spark Streaming：实时流处理
 - Spark MLib：机器学习
 - Spark GraphX：图计算
- 具有高吞吐、低延时、通用易扩展、高容错等特点
- 采用Scala语言开发
- 提供多种运行模式



➤ 计算高效

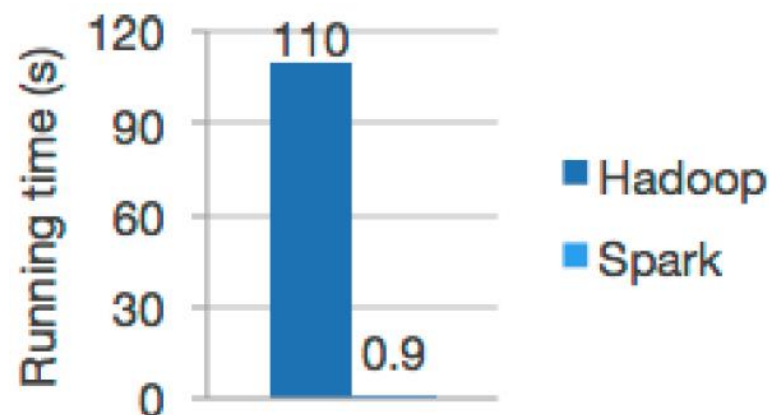
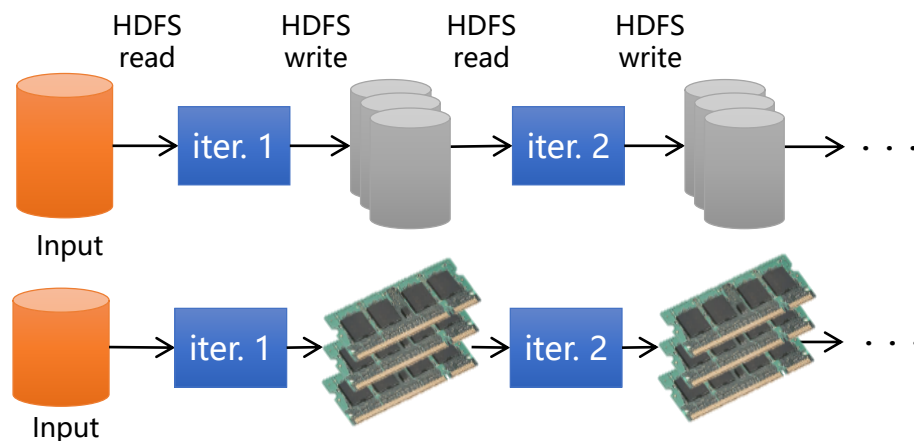
- 利用内存计算、Cache缓存机制，支持迭代计算和数据共享，减少数据读取的IO开销
- 利用DAG引擎，减少中间计算结果写入HDFS的开销
- 利用多线程池模型，减少任务启动开销，避免Shuffle中不必要的排序和磁盘IO操作

➤ 通用易用

- 适用于批处理、流处理、交互式计算、机器学习算法等场景
- 提供了丰富的开发API，支持Scala、Java、Python、R等

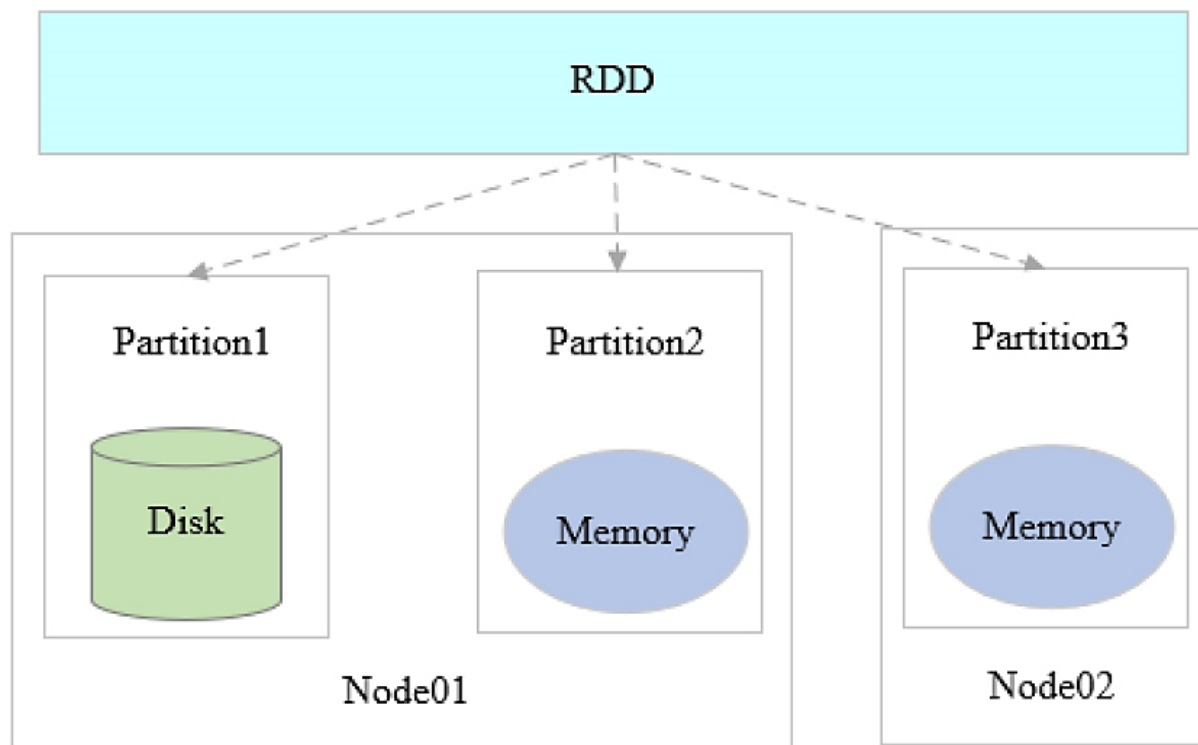
➤ 运行模式多样

- Local模式
- Standalone模式
- YARN/Mesos模式



➤ RDD

- 弹性分布式数据集（Resilient Distributed Datasets）
 - 分布在集群中的只读对象集合
 - 由多个Partition组成
 - 通过转换操作构造
 - 失效后自动重构（弹性）
 - 存储在内存或磁盘中
- Spark基于RDD进行计算



➤ RDD操作（Operator）

- Transformation（转换）

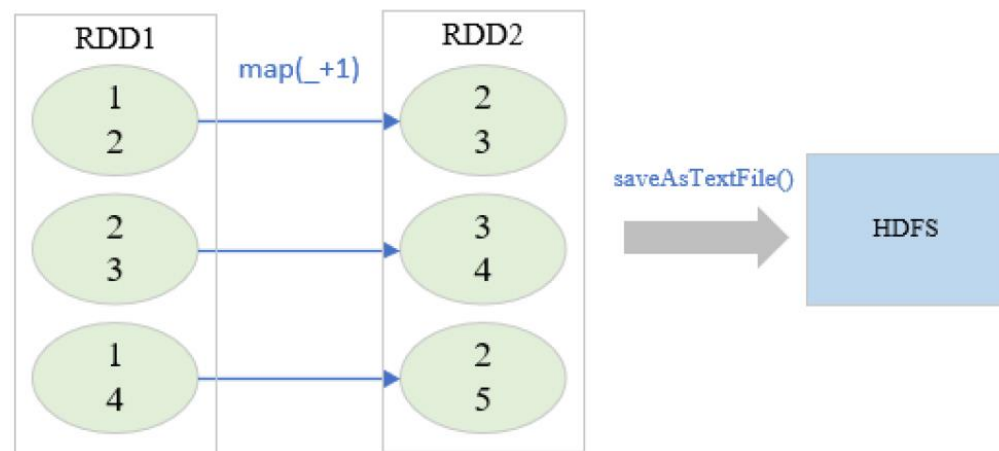
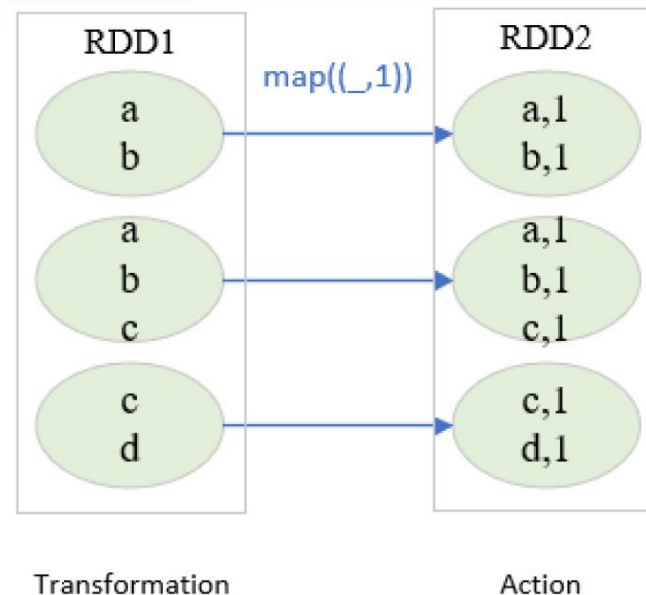
- 将Scala集合或Hadoop输入数据构造一个新RDD
- 通过已有的RDD产生新RDD
- 惰性执行：只记录转换关系，不触发计算
- 例如：map、filter、flatMap、union、distinct、sortByKey

- Action（动作）

- 通过RDD计算得到一个值或一组值
- 真正触发计算
- 例如：first、count、collect、foreach、saveAsTextFile

➤ 示例：RDD的两种操作

- `rdd1.map(_,+1).saveAsTextFile("hdfs://node01:9000")`



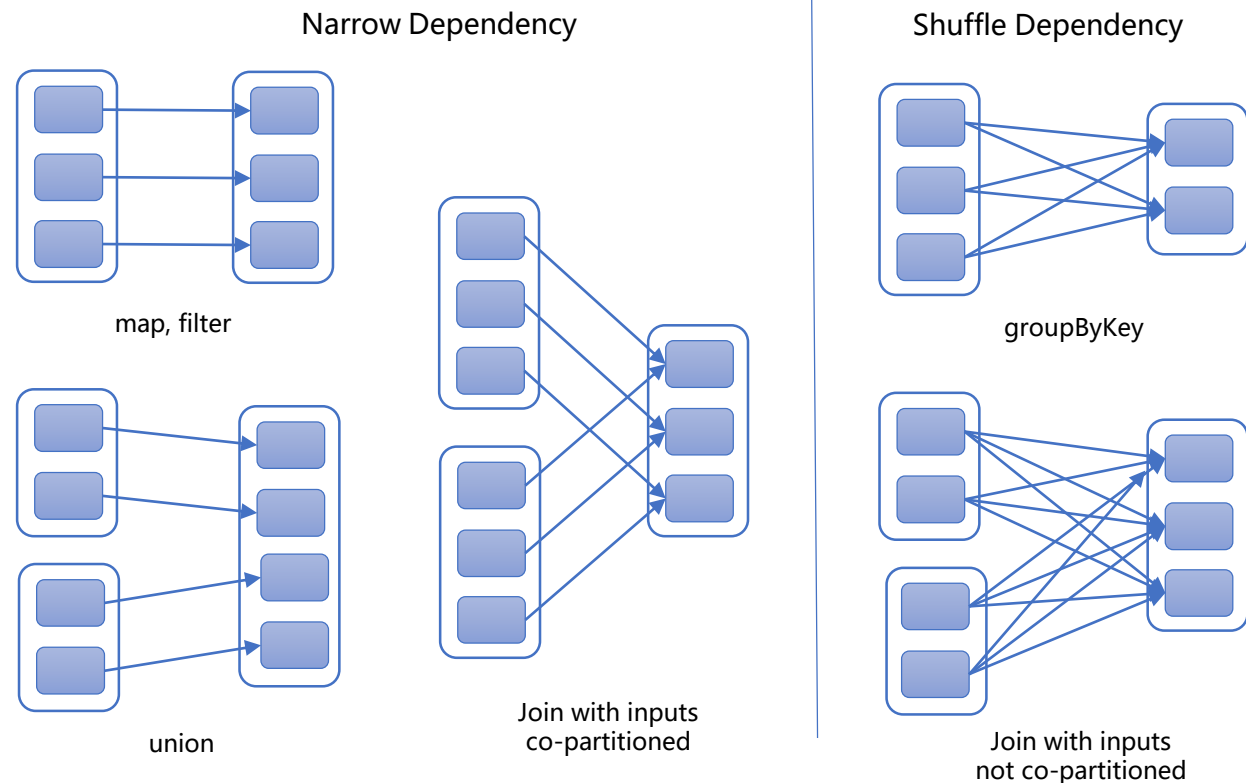
➤ RDD依赖 (Dependency)

- 窄依赖 (Narrow Dependency)

- 父RDD中的分区最多只能被一个子RDD的一个分区使用
- 子RDD如果有部分分区数据丢失或损坏，只需从对应的父RDD重新计算恢复
- 例如：map、filter、union

- 宽依赖 (Shuffle/Wide Dependency)

- 子RDD分区依赖父RDD的所有分区
- 子RDD如果部分或全部分区数据丢失或损坏，必须从所有父RDD分区重新计算
- 相对于窄依赖，宽依赖付出的代价要高很多，尽量避免使用
- 例如：groupByKey、reduceByKey、sortByKey



➤ 示例：WordCount

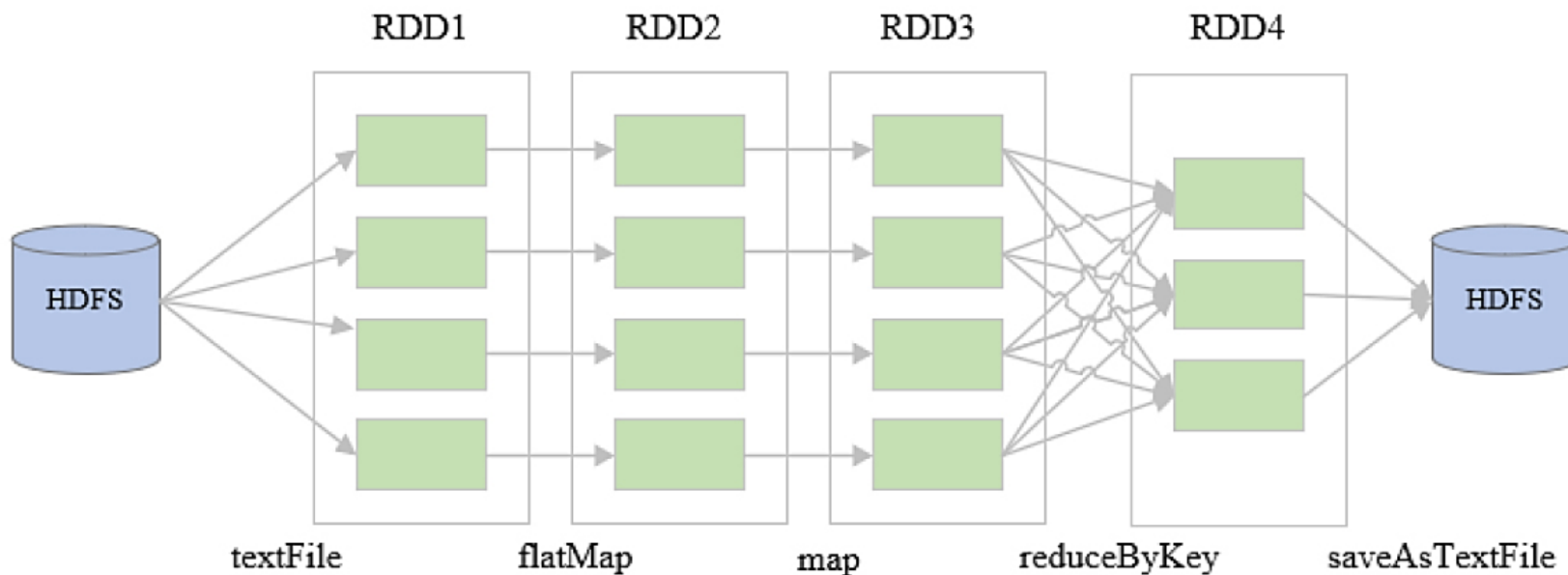
```
val rdd1 = sc.textFile("hdfs://node01:9000/data/wc/in")
```

```
val rdd2 = rdd1.flatMap(_.split("\t"))
```

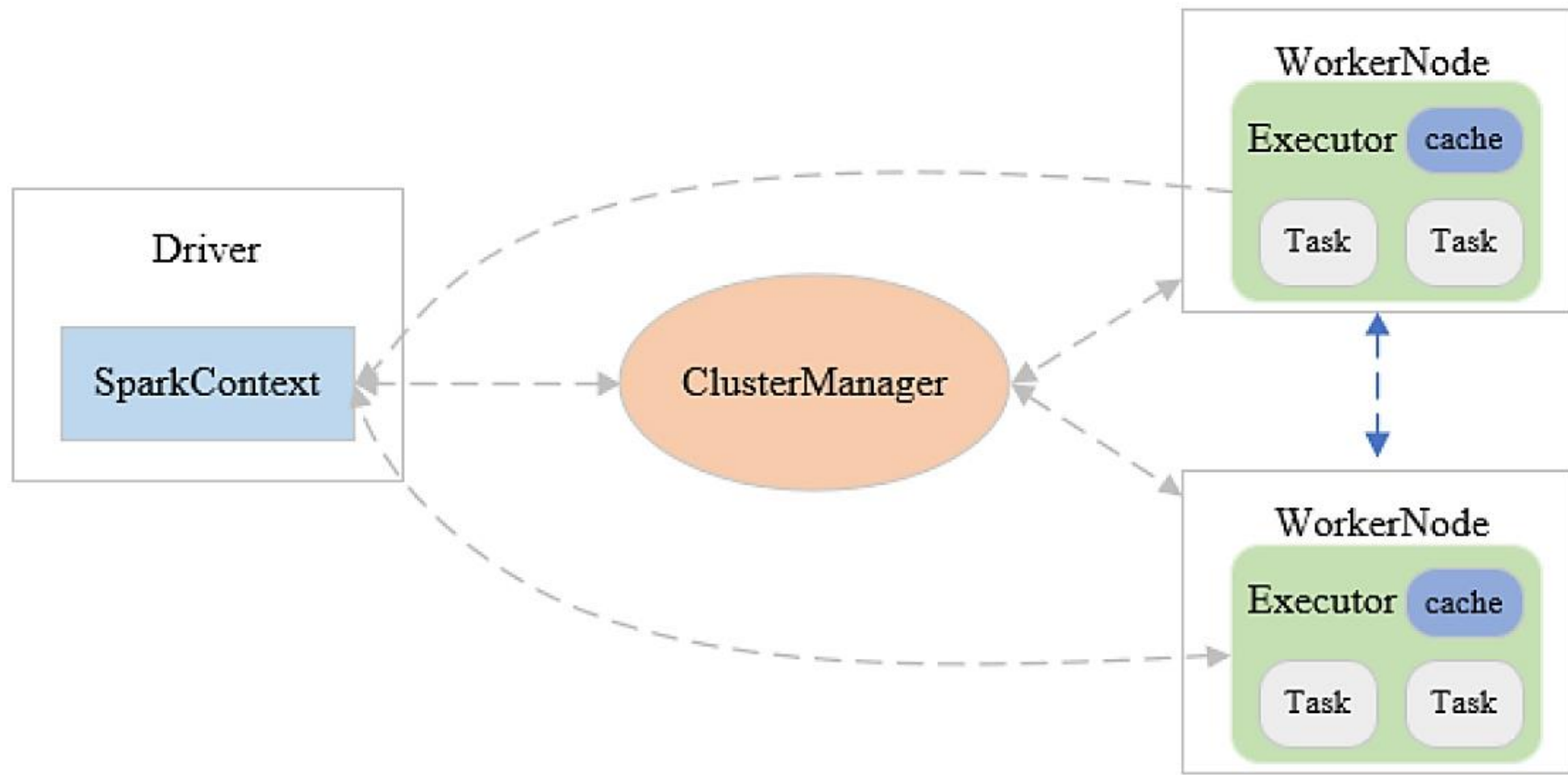
```
val rdd3 = rdd2.map((_,1))
```

```
val rdd4 = rdd3.reduceByKey((_+_))
```

```
rdd4.saveAsTextFile("hdfs://node01:9000/data/wc/out")
```



➤ 抽象模式



➤ 抽象模式

- Driver

- 一个Spark程序有一个Driver，一个Driver创建一个SparkContext，程序的main函数运行在Driver中
- 负责解析Spark程序、划分Stage、调度任务到Executor上执行

- SparkContext

- 负责加载配置信息，初始化运行环境，创建DAGScheduler和TaskScheduler

- Executor

- 负责执行Driver分发的任务，一个节点可以启动多个Executor，每个Executor通过多线程运行多个任务

- Task

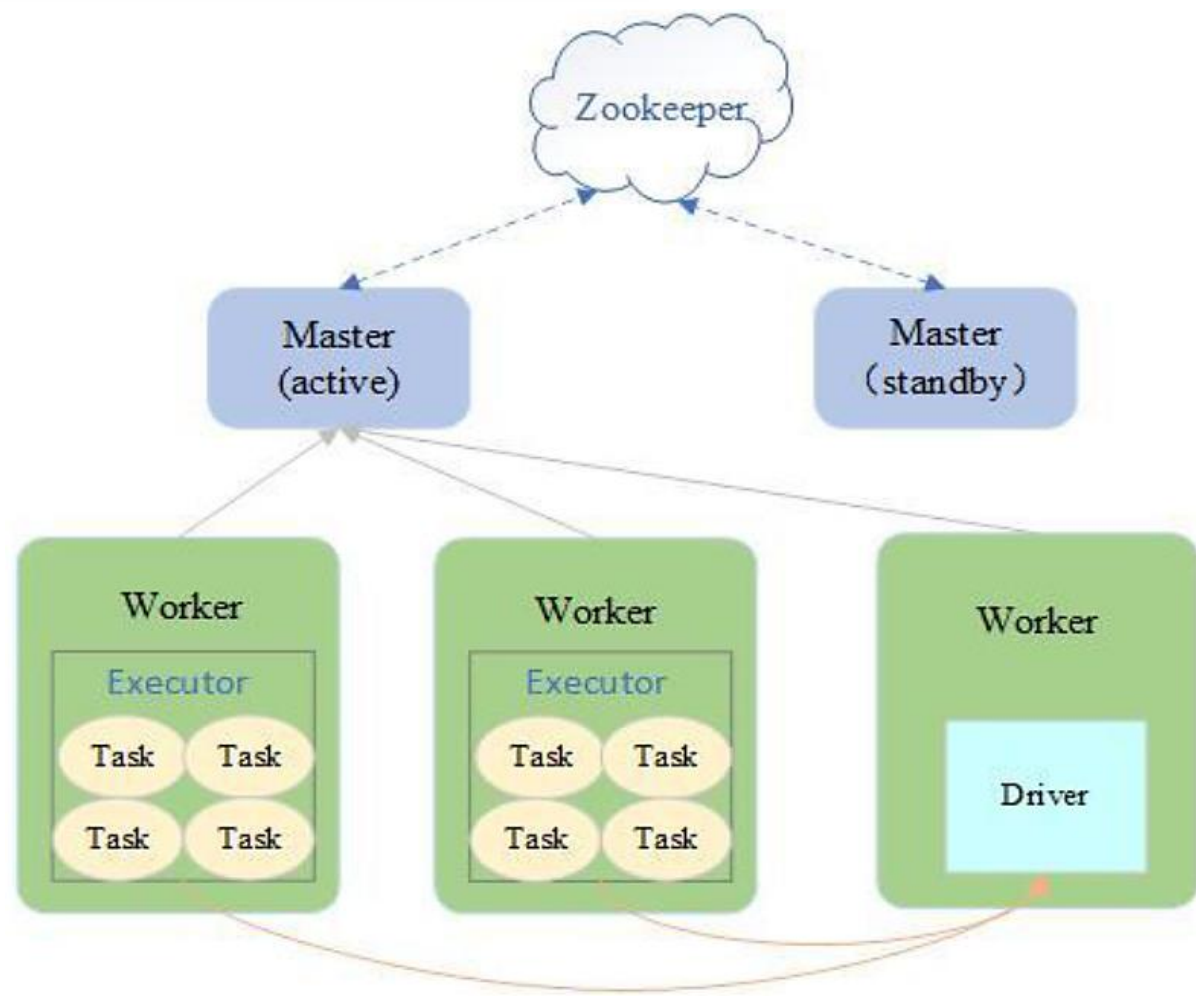
- Spark运行的基本单位，一个Task负责处理若干RDD分区的计算逻辑

➤ Local模式

- 单机运行，通常用于测试
- Spark程序以多线程方式直接运行在本地

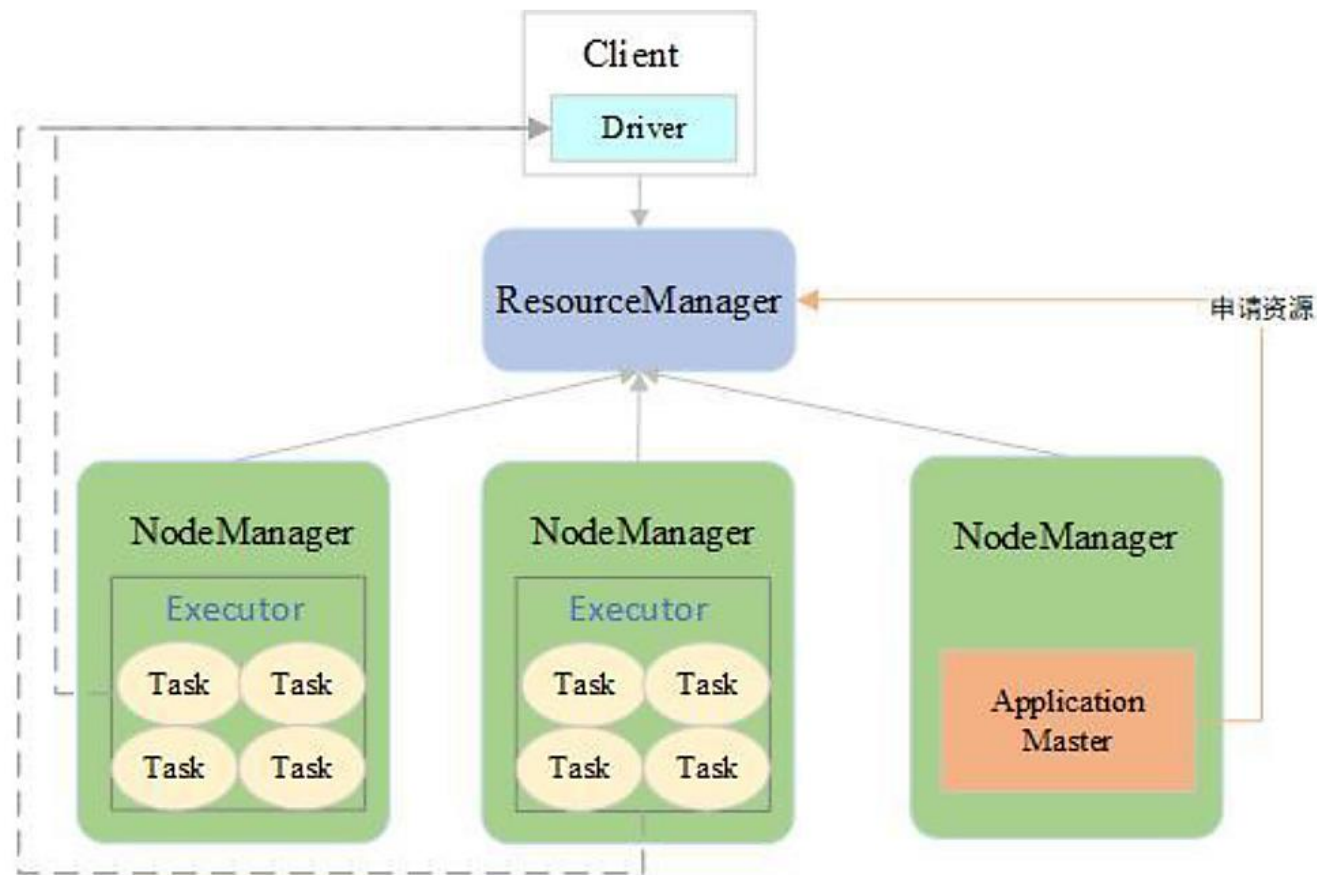
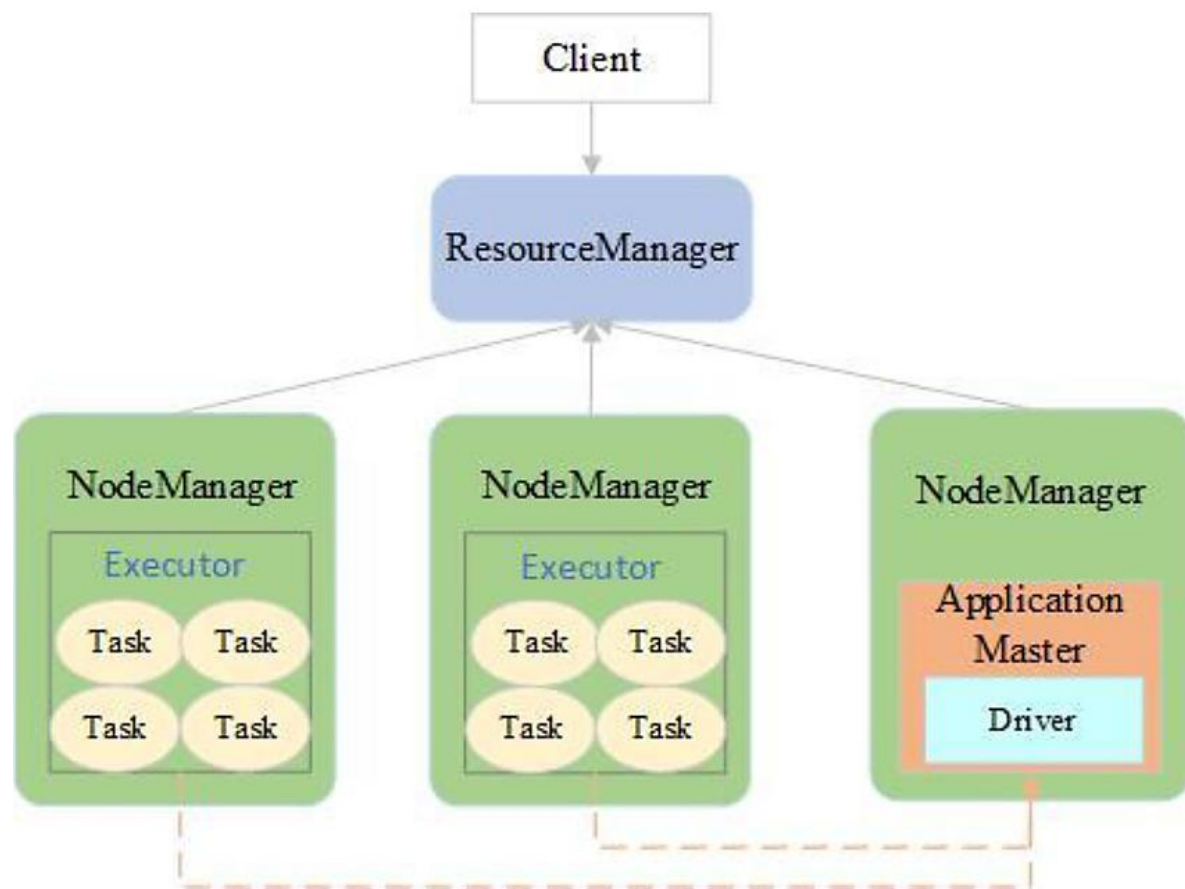
➤ Standalone模式

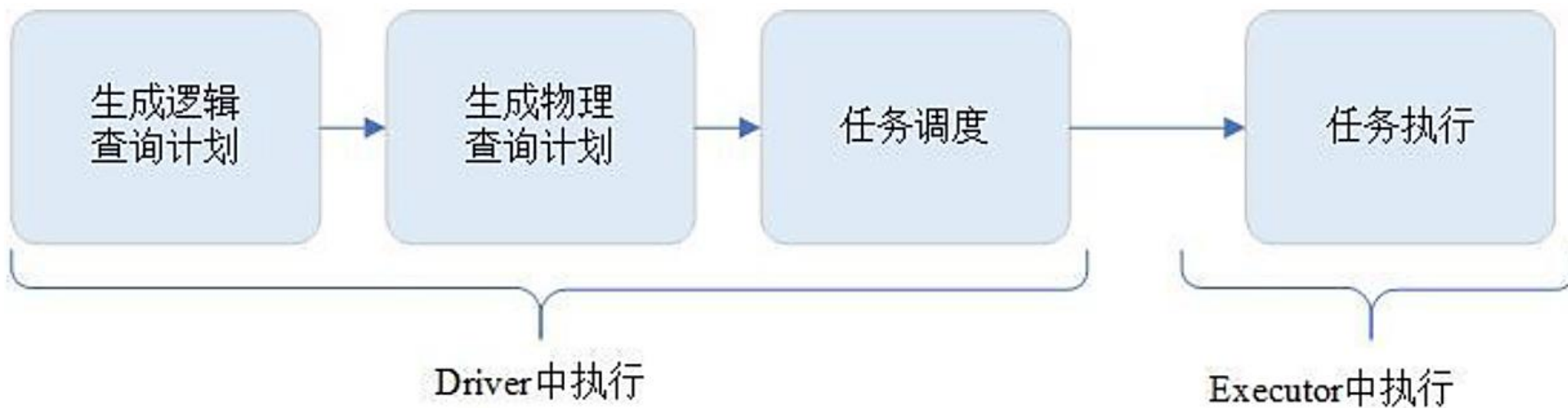
- Spark集群独立运行，不依赖于第三方资源管理系统，如：YARN、Mesos
- 采用Master/Slave架构
- Driver在Worker中运行，Master只负责集群管理
- ZooKeeper负责Master HA，避免单点故障
- 适用于集群规模不大，数据量不大的情况



➤ YARN模式

- YARN-Client模式：适用于交互和调试
- YARN-Cluster模式：适用于生产环境





➤ 生成逻辑查询计划

```
sc.textFile(inputArg)
```

RDD[String]

```
.flatMap(_ .split("\t"))
```

RDD[String]

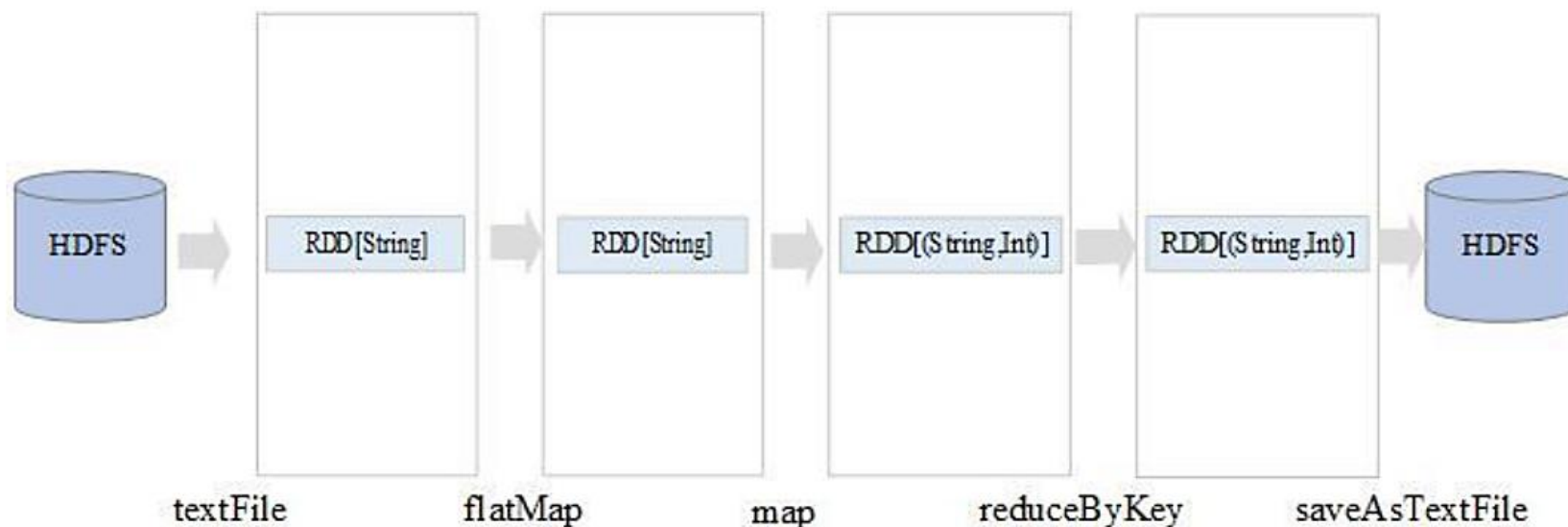
```
.map((_,1))
```

RDD[(String,Int)]

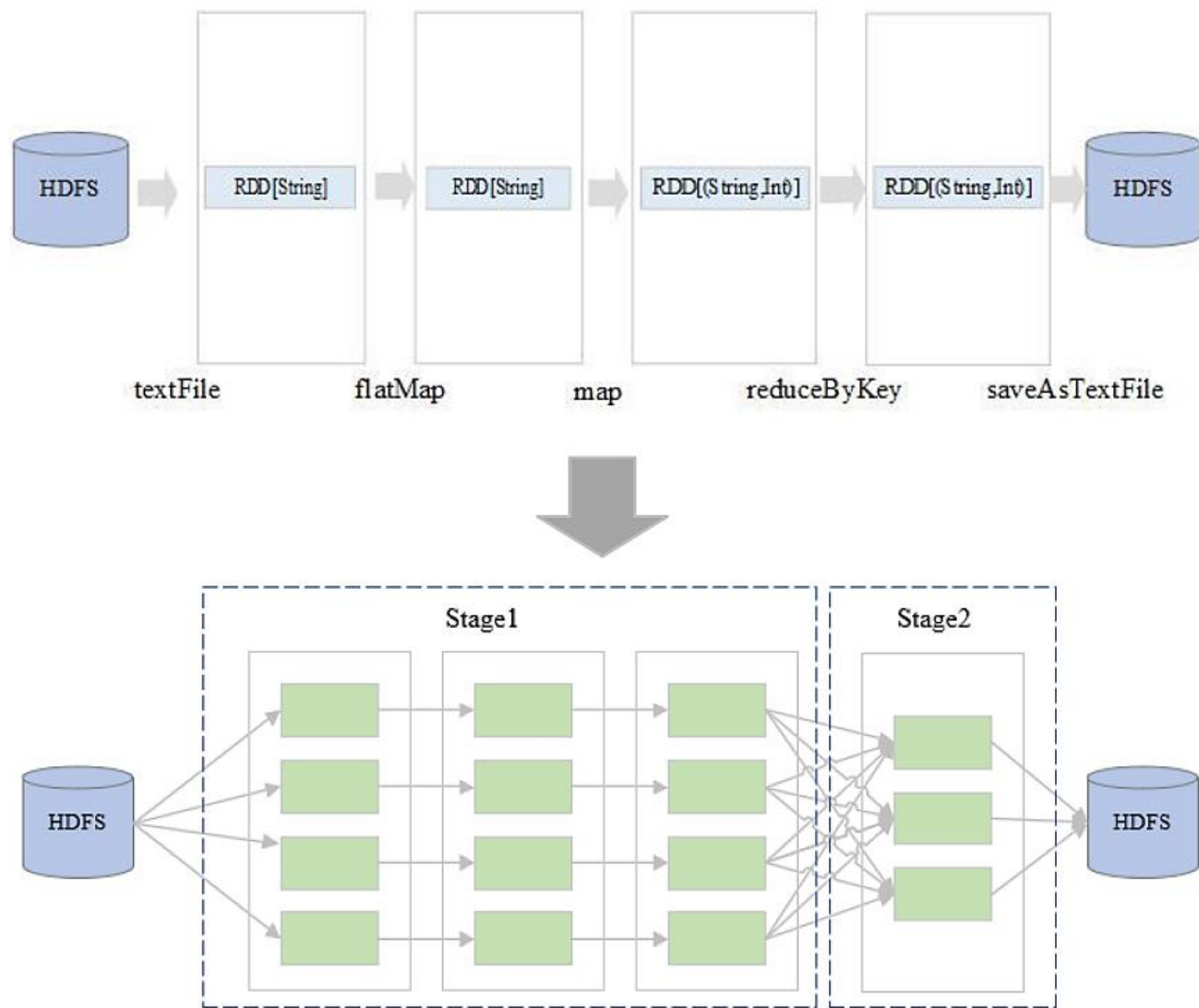
```
.reduceByKey(_ + _)
```

RDD[(String,Int)]

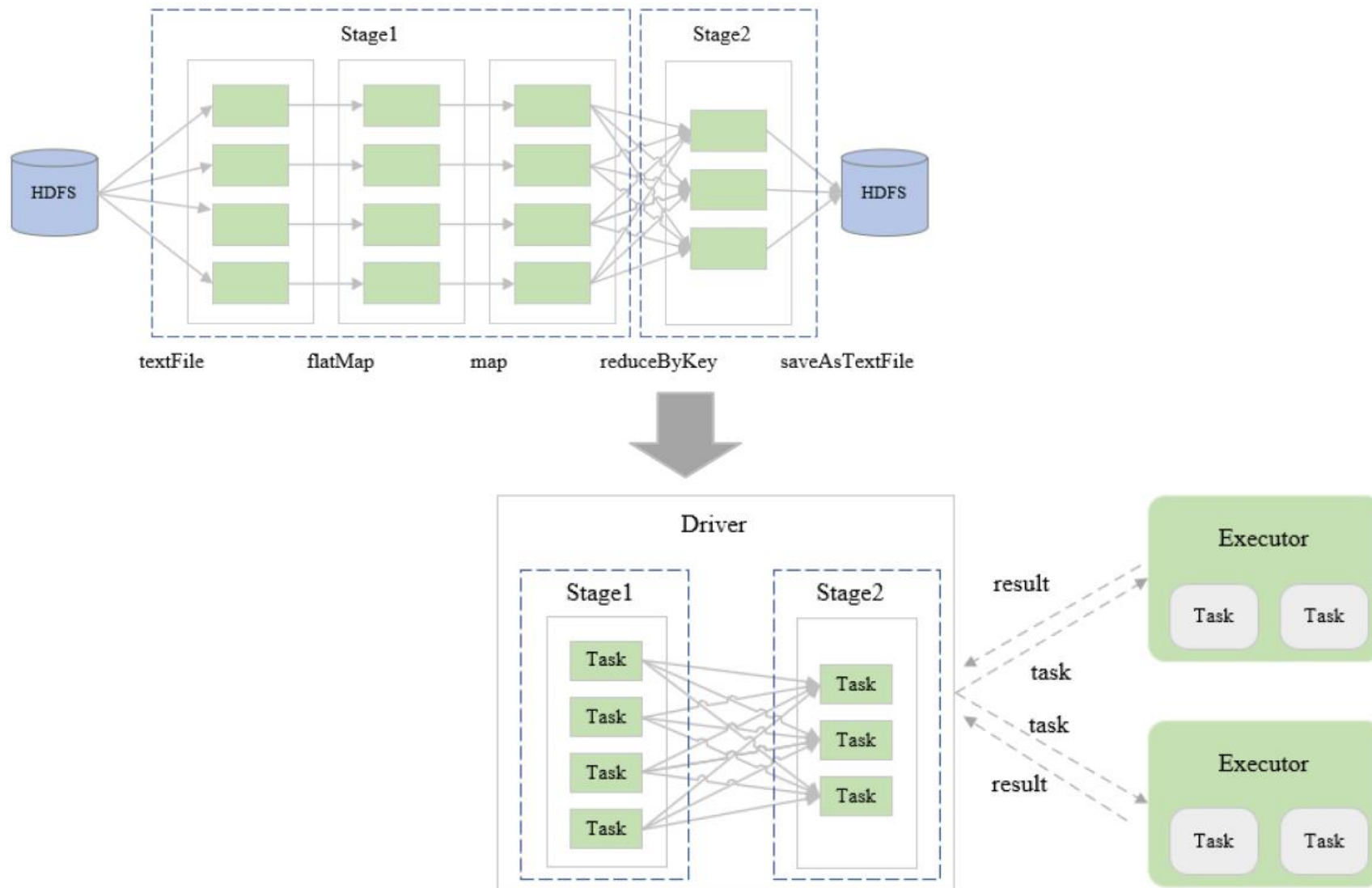
```
.saveAsTextFile(outArg)
```



➤ 生成物理查询计划



➤ 任务调度与执行



➤ DAG (Directed Acyclic Graph)

- 有向无环图：一个有向图无法从任意顶点出发经过若干条边回到该点
- 受制于某些任务必须比另一些任务较早执行的约束，可排序为一个队列的任务集合，该队列可由一个DAG图呈现
- Spark程序的内部执行逻辑可由DAG描述，顶点代表任务，边代表任务间的依赖约束

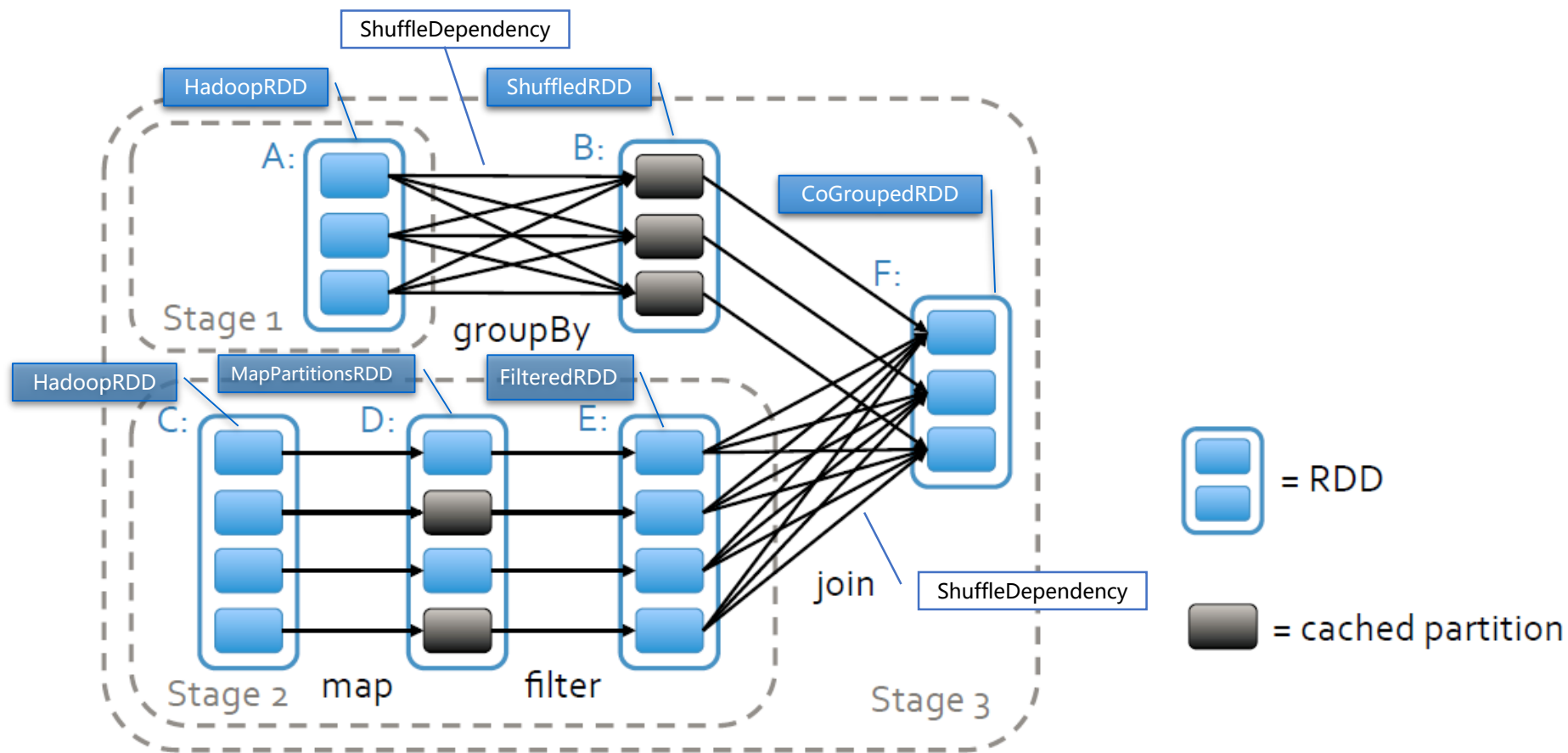
➤ DAGScheduler

- 根据任务的依赖关系建立DAG
- 根据依赖关系是否为宽依赖，即是否存在Shuffle，将DAG划分为不同的阶段（Stage）
- 将各阶段中的Task组成的TaskSet提交到TaskScheduler

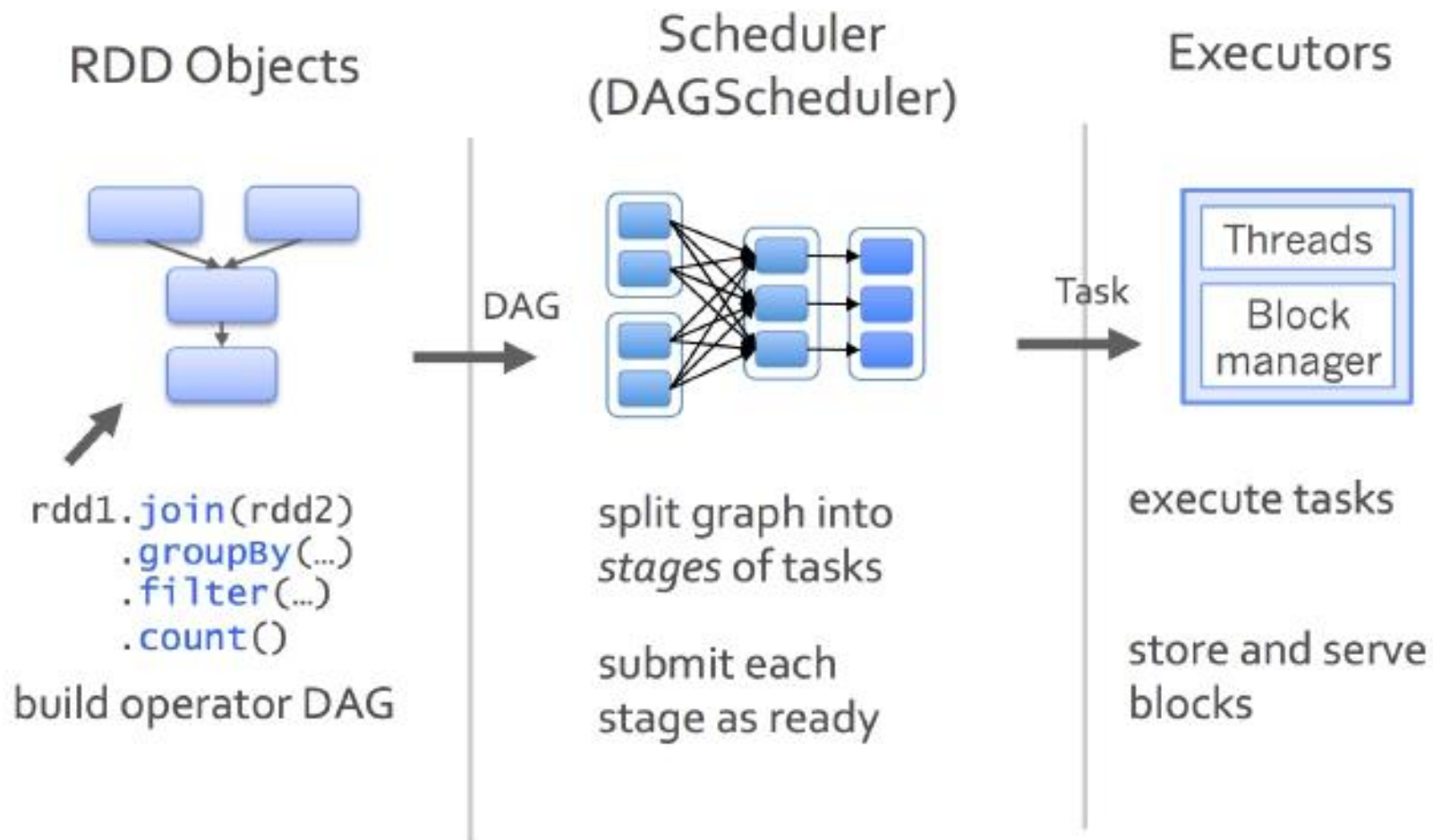
➤ TaskScheduler

- 负责Application的任务调度
- 重新提交失败的Task
- 为执行速度慢的Task启动备用Task

➤ 示例：DAG



➤ 示例：DAGScheduler



➤ Web监控：http://{inceptor_server_ip}:4040

TRANSWARP DATA HUB

Spark

JobsClusterLocalStorageHolodeskEnvironmentExecutors

Inceptor::tdh-31 application UI

Details for Job 92

Status: SUCCEEDED

Job Group: 0693da7c-705e-45ff-902e-d22ae21125cd

Completed Stages: 2

Event Timeline

DAG Visualization

Stage 104

Order by

Result output

Stage 105

Table reader columns_v

Filter

Data exchange

Completed Stages (2)

Stage Id	Description	Submitted	Duration	Tasks: Succeeded/Total	Input	Shuffle Read	Shuffle Write
104	<div><div>SELECT * FROM system.columns_v WHERE database_name='tpcds_orc_2' AND table_name='atomicity_table_src' ORDER BY column_id</div><div>runJob at FileSinkOperator.scala:234</div></div> <div>+details</div>	2016/07/16 05:47:00	0.3 s	<div>8/8</div>		89.0 B	
105	<div><div>SELECT * FROM system.columns_v WHERE database_name='tpcds_orc_2' AND table_name='atomicity_table_src' ORDER BY column_id</div><div>mapPartitionsWithContext at Operator.scala:562</div></div> <div>+sources</div> <div>+details</div>	2016/07/16 05:47:00	0.2 s	<div>1/1</div>			185.0 B



Q&A

TRANSWARP
星环科技