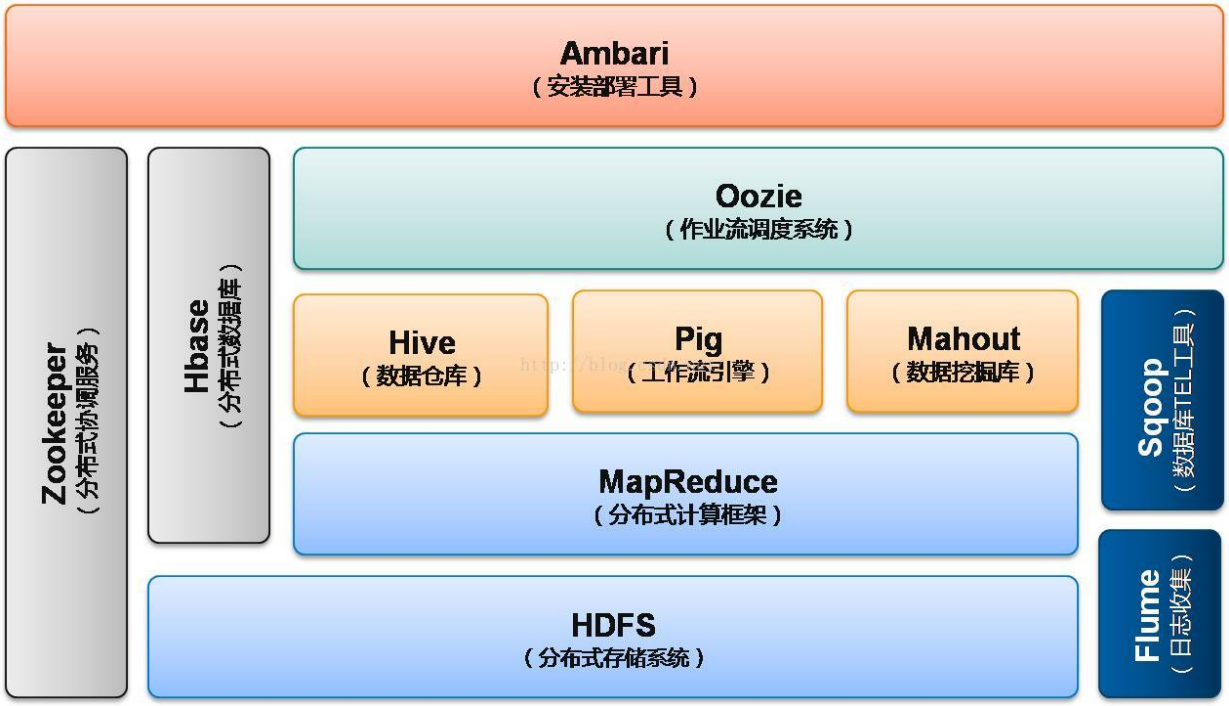
Mapreduce

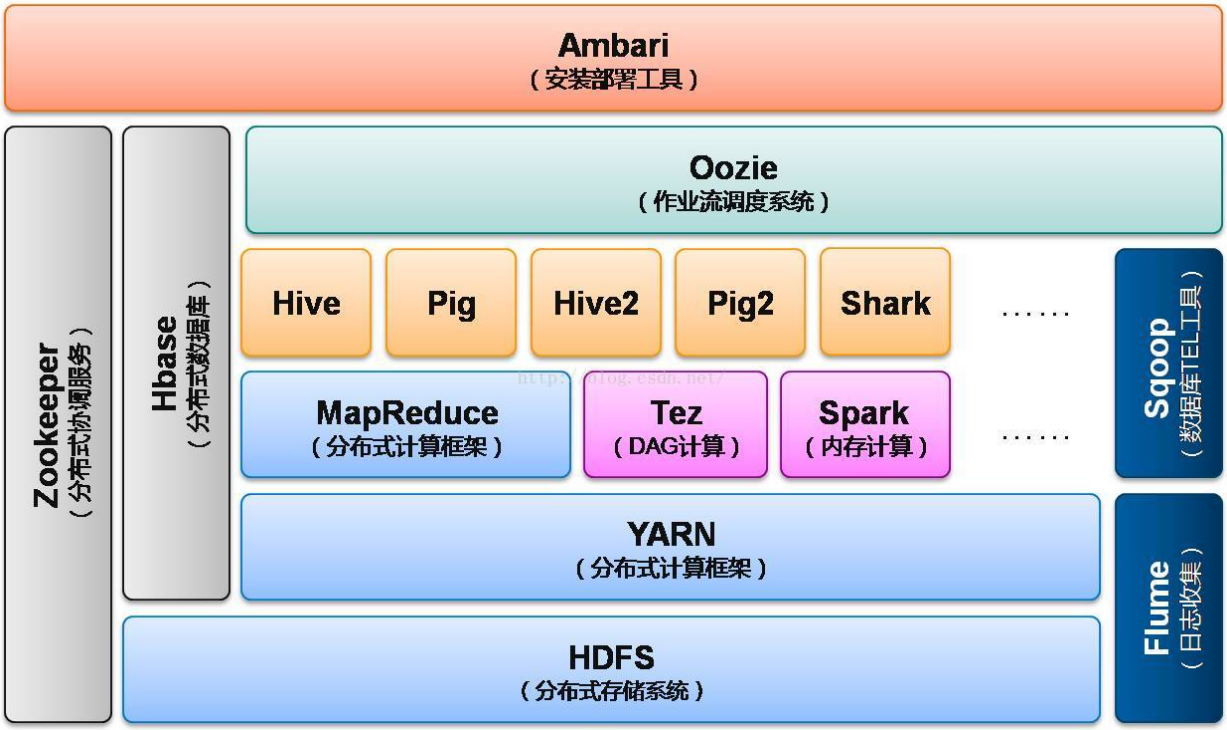
MapReduce特点：

- 易于编程
- 良好的扩展性
- 高容错性
- 适合PB级海量数据的离线处理

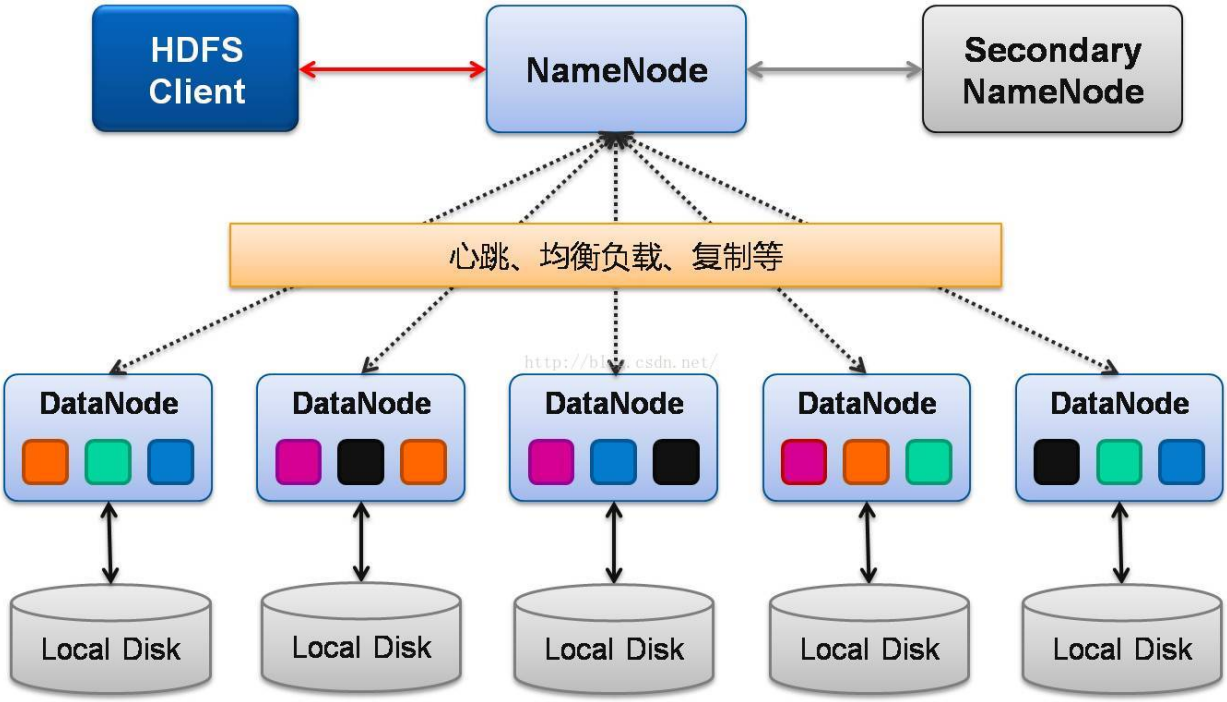
1、hadoop1.0时期架构



2、hadoop2.0时期架构



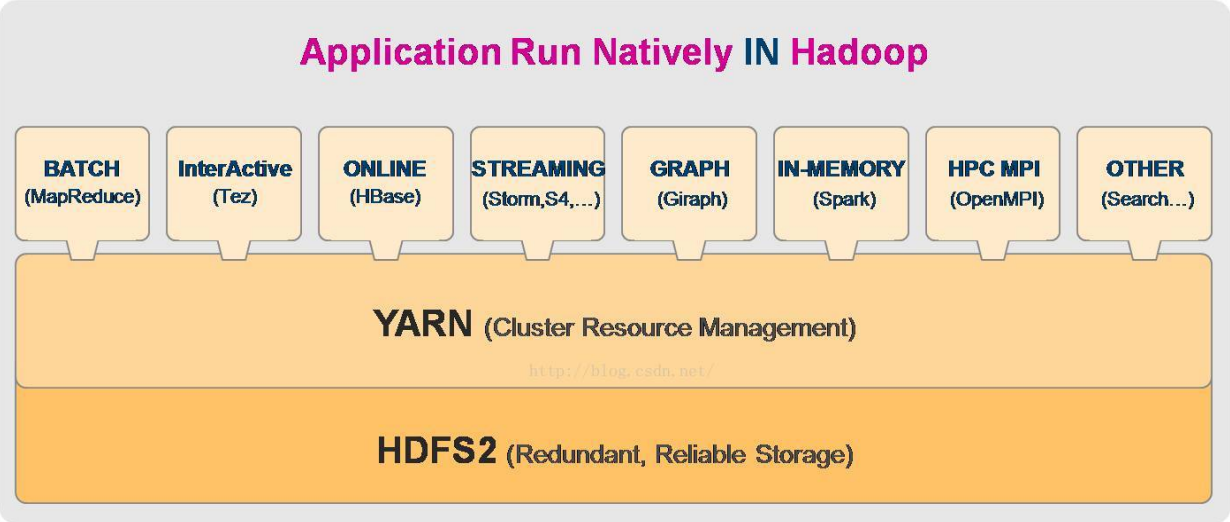
3、hdfs架构



Active Namenode主 Master（只有一个），管理 HDFS 的名称空间，管理数据块映射信息；配置副本策略；处理客户端读写请求 Secondary NameNode NameNode 的热备；定期合并 fsimage 和 fsedits，推送给 NameNode；当 Active NameNode 出现故障时，快速切换为新的 Active NameNode。 Datanode Slave（有多个）；存储实际的数据块；执行数据块读 / 写 Client 与 NameNode 交互，获取文件位置信息；与 DataNode 交互，读取或者写入数据；管理 HDFS、访问 HDFS。

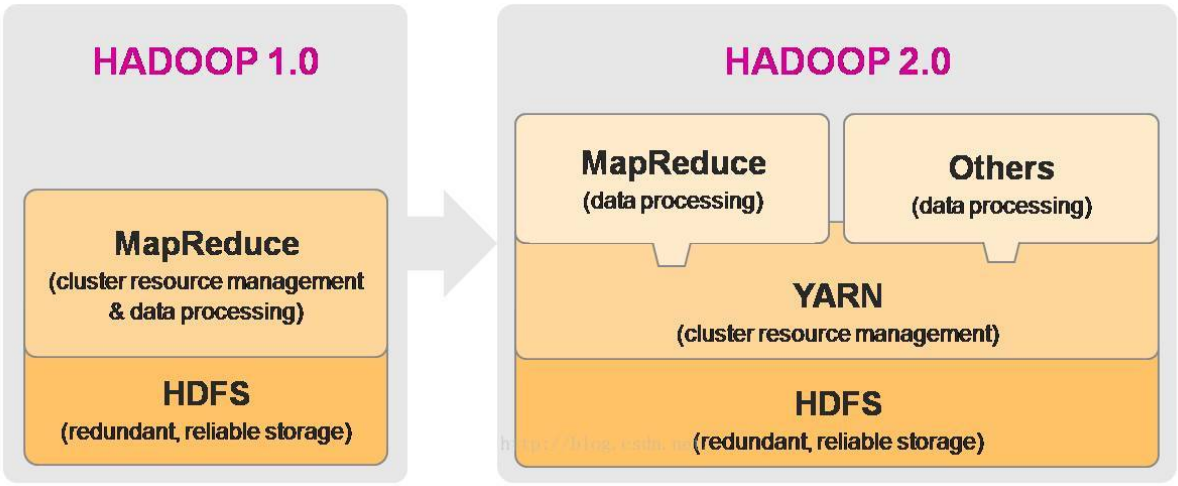
4、MapReduce 源自于 Google 的 MapReduce 论文发表于 2004 年 12 月Hadoop MapReduce 是 Google MapReduce 克隆版MapReduce特点良好的扩展性高容错性适合 PB 级以上海量数据的离线处理

5、yarn架构



- YARN是什么
  - Hadoop 2.0 新增系统
  - 负责集群的资源管理和调度
  - 使得多种计算框架可以运行在一个集群中
- YARN的特点
  - 良好的扩展性、高可用性
  - 对多种类型的应用程序进行统一管理和调度
  - 自带多种多用户调度器，适合共享集群环境

6、hadoop1.0与hadoop2.0比较图



- 分布式存储系统 HDFS（Hadoop Distributed File System）
  提供了高可靠性、高扩展性和高吞吐率的数据存储服务
- 分布式计算框架 MapReduce
  具有易于编程、高容错性和高扩展性等优点
- 资源管理系统 YARN（Yet Another Resource Negotiator）
  负责集群资源的统一管理和调度

7、Hive（基于MR的数据仓库）由Facebook开源，最初用于海量结构化日志数据统计；ETL（Extraction-Transformation-Loading）工具构建在Hadoop之上的数据仓库；数据计算使用MapReduce，数据存储使用HDFSHive 定义了一种类 SQL 查询语言——HQL类似SQL，但不完全相同通常用于进行离线数据处理（采用 MapReduce）；可认为是一个 HQL→MR 的语言翻译器

8、Hbase（分布式数据库）源自 Google 的 Bigtable 论文发表于 2006 年 11 月Hbase 是 Google Bigtable 克隆版

9、Hadoop 发行版（开源版）

- **Apache Hadoop**
  - 推荐使用最新的2.x.x版本，比如2.4.0
  - 下载地址：http://hadoop.apache.org/releases.html
  - SVN：http://svn.apache.org/repos/asf/hadoop/common/branches/
- **CDH（Cloudera Distributed Hadoop）**
  - 推荐使用最新的CDH5 版本，比如CDH 5.0.0
  - 下载地址： http://archive.cloudera.com/cdh5/cdh/
- **HDP（Hortonworks Data Platform）**
  - 推荐使用最新的HDP 2.x 版本，比如 HDP 2.1 版本
  - 下载地址：http://zh.hortonworks.com/hdp/downloads/

MapReduce的编程模型

1、用户编写完MapReduce 程序后，按照一定的规则指定程序的输入和输出目录，并提交到Hadoop集群中。作业在Hadoop 中的执行过程如图1所示。Hadoop 将输入数据切分 成若干个输入分片（input split，后面简称split），并将每个split 交给一个Map Task 处理；Map Task 不断地从对应的split 中解析出一个个key/value，并调用m a p ( ) 函数处理，处理完 之后根据Reduce Task 个数将结果分成若干个分片（partition）写到本地磁盘；同时，每个Reduce Ta s k 从每个M a p Ta s k 上读取属于自己的那个partition，然后使用基于排序的方法将 key 相同的数据聚集在一起，调用Reduce ( ) 函数处理，并将结果输出到文件中。

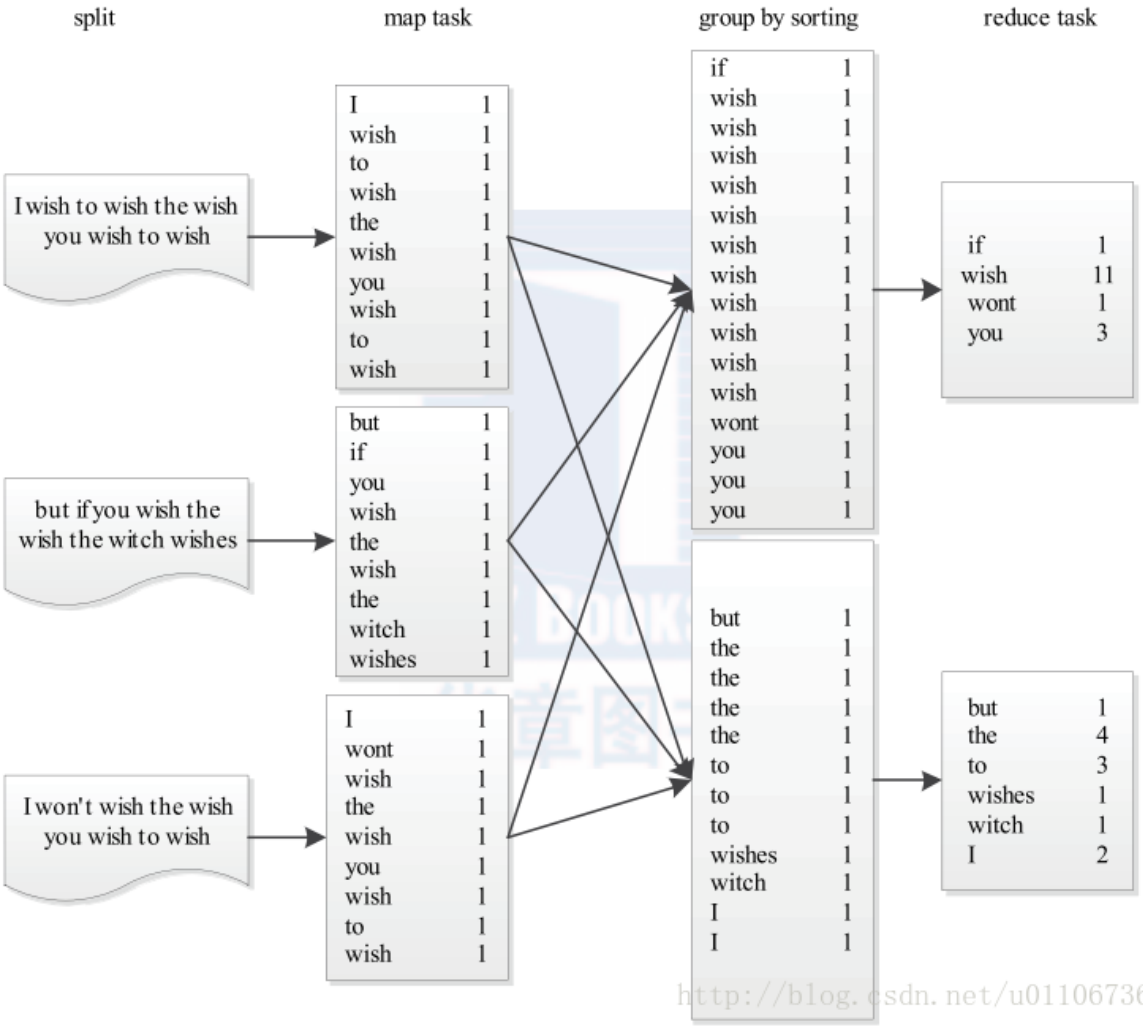


图1 Word Count 程序运行过程 2、上面的程序还缺少三个基本的组件，功能分别是：

①指定输入文件格式。将输入数据切分成若干个s p l i t ，且将每个s p l i t 中的数据解析成一个个m a p ( ) 函数要求的k e y / v a l u e 对。

②确定m a p ( ) 函数产生的每个k e y / v a l u e 对发给哪个R e d u c e Ta s k 函数处理。

③指定输出文件格式，即每个k e y / v a l u e 对以何种形式保存到输出文件中。

在Hadoop MapReduce 中，这三个组件分别是InputFormat 、Partitioner 和OutputFormat ，它们均需要用户根据自己的应用需求配置。而对于上面的Wo r d C o u n t 例子，默认情况下Hadoop 采用的默认实现正好可以满足要求，因而不必再提供。

综上所述，Hadoop MapReduce 对外提供了5 个可编程组件，分别是InputFormat 、M a p p e r 、Partitioner 、Reducer 和OutputFormat 。

三、Hadoop MapReduce 作业的生命周期 本节主要讲解Hadoop MapReduce 作业的生命周期，即作业从提交到运行结束经历的整个过程。本节只是概要性地介绍MapReduce 作业的生命周期；

假设用户编写了一个MapReduce 程序，并将其打包成x x x . j a r 文件，然后使用以下命 令提交作业：

[java] view plain copy

```
1. $HADOOP_HOME/bin/hadoop jar xxx.jar \
2.     -D mapred.job.name="xxx" \
```

3. -D mapred.map.tasks=3 \
4. -D mapred.reduce.tasks=2 \
5. -D input=/test/input \
6. -D output=/test/output

则该作业的运行过程如图2所示。

这个过程分为以下5 个步骤： 步骤1　作业提交与初始化。用户提交作业后，首先由JobClient 实例将作业相关信息，比如将程序jar 包、作业配置文件、分片元信息文件等上传到分布式文件系统（一般为HDFS）上，其中，分片元信息文件记录了每个输入分片的逻辑位置信息。然后JobClient通过RPC通知JobTracker 。JobTracker 收到新作业提交请求后，由作业调度模块对作业进行初始化：为作业创建一个JobInProgress 对象以跟踪作业运行状况，而JobInProgress 则会为每个Task 创建一个TaskInProgress 对象以跟踪每个任务的运行状态，TaskInProgress 可能需要管理多个“Task 运行尝试”（称为“TaskAttempt”）。

步骤2　任务调度与监控。前面提到，任务调度和监控的功能均由JobTracker 完成。TaskTracker 周期性地通过Heartbeat 向JobTracker 汇报本节点的资源使用情况，一旦出现 空闲资源，JobTracker 会按照一定的策略选择一个合适的任务使用该空闲资源，这由任务调度器完成。任务调度器是一个可插拔的独立模块，且为双层架构，即首先选择作业，然后 从该作业中选择任务，其中，选择任务时需要重点考虑数据本地性。此外，JobTracker 跟踪作业的整个运行过程，并为作业的成功运行提供全方位的保障。首先，当TaskTracker 或者Task 失败时，转移计算任务；其次，当某个Task 执行进度远落后于同一作业的其他Task 时，为之启动一个相同Task ，并选取计算快的Task 结果作为最终结果。

步骤3　任务运行环境准备。运行环境准备包括JVM 启动和资源隔离，均由TaskTracker 实现。TaskTracker 为每个Task 启动一个独立的JVM 以避免不同Task 在运行过程中相互影响；同时，TaskTracker 使用了操作系统进程实现资源隔离以防止Task 滥用资源。 步骤4　任务执行。TaskTracker 为Task 准备好运行环境后，便会启动Task 。在运行过程中，每个Task 的最新进度首先由Task 通过RPC 汇报给TaskTracker ，再由TaskTracker汇报给JobTracker 。

步骤5　作业完成。待所有Task 执行完毕后，整个作业执行成功。

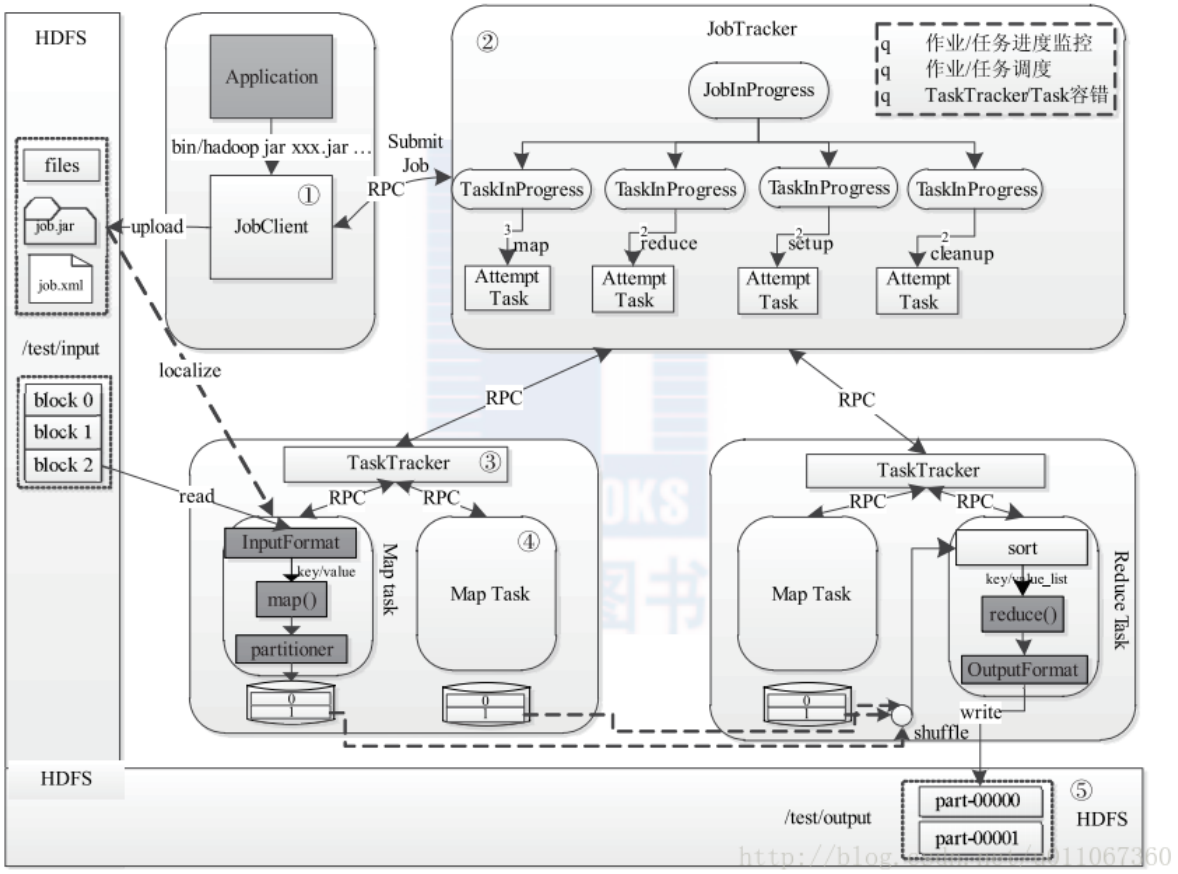


图2 Hadoop MapReduce 作业的生命周期

四、MapReduce 编程模型的实现

1、MapReduce 编程模型给出了其分布式编程方法，共分5 个步骤： 1 ）迭代（iteration ）。遍历输入数据，并将之解析成key/value 对。 2 ）将输入key/value 对映射（m a p ）成另外一些key/value 对。 3 ）依据k e y 对中间数据进行分组（grouping ）。 4 ）以组为单位对数据进行归约（reduce ）。 5 ）迭代。将最终产生的key/value 对保存到输出文件中。 MapReduce 将计算过程分解成以上5 个步骤带来的最大好处是组件化与并行化。为了实现MapReduce 编程模型，Hadoop 设计了一系列对外编程接口。用户可通过实现这些接口完成应用程序的开发。

2、MapReduce 编程接口体系结构 MapReduce 编程模型对外提供的编程接口体系结构如图3 所示，整个编程模型位于应用程序层和MapReduce 执行器之间，可以分为两层。第一层是最基本的J a v a A P I ，主要有5 个可编程组件，分别是InputFormat 、Mapper 、Partitioner 、Reduce r 和OutputFormat 。 Hadoop 自带了很多直接可用的InputFormat 、Partitioner 和OutputFormat ，大部分情况下，用户只需编写Mapper 和Reducer 即可。第二层是工具层，位于基本J a v a A P I 之上，主要是为了方便用户编写复杂的MapReduce 程序和利用其他编程语言增加MapReduce 计算平台的兼容性而提出来的。在该层中，主要提供了4 个编程工具包。

J o b C o n t r o l ： 方便用户编写有依赖关系的作业，这些作业往往构成一个有向图，所以 通常称为DAG （Directed Acyclic Graph ）作业，如第2 章中的朴素贝叶斯分类算法实现便是4 个有依赖关系的作业构成的DAG 。 C h a i n Mapper / Chain Reduce r ： 方便用户编写链式作业，即在M a p 或者Reduce 阶段存在多个Mapper ， 形式如下： [MAPPER+ REDUCER MAPPER*] Hadoop Streaming ： 方便用户采用非J a v a 语言编写作业，允许用户指定可执行文件或者脚本作为Mapper / Reduce r 。 Hadoop Pipes ： 专门为C / C + + 程序员编写MapReduce 程序提供的工具包。

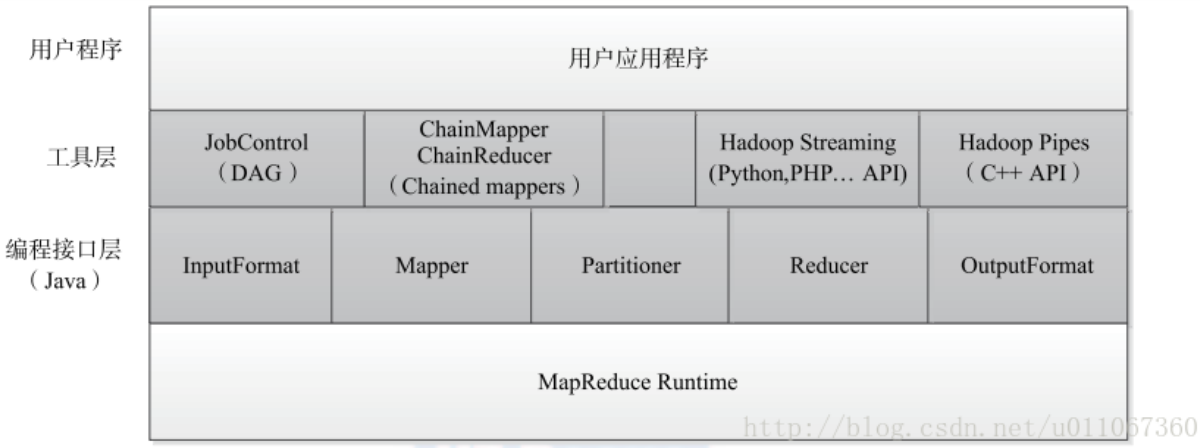


图3 MapReduce 编程接口体系结构

五、小结：

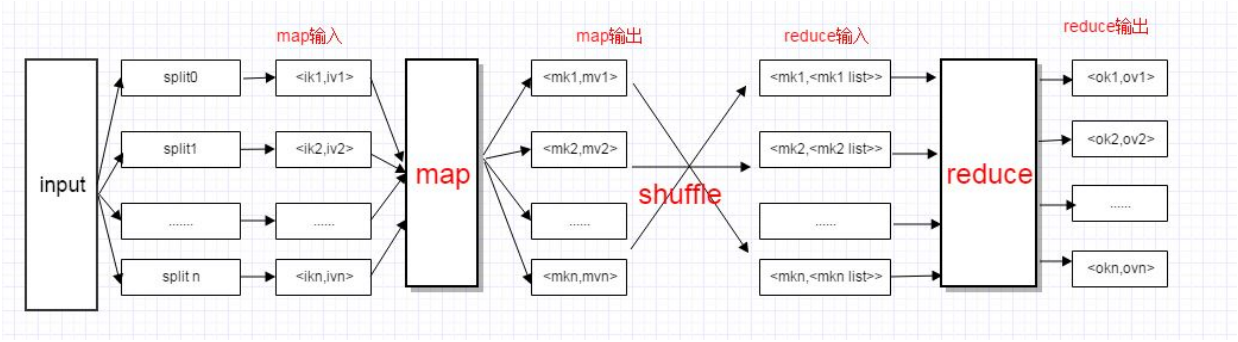
1、Hadoop MapReduce 直接诞生于搜索领域，以易于编程、良好的扩展性和高容错性为设计目标。它主要由两部分组成：编程模型和运行时环境。其中，编程模型为用户提供了5 个可编程组件，分别是InputFormat 、Mapper 、Partitioner 、Reduce r 和OutputFormat ； 运行时环境则将用户的MapReduce 程序部署到集群的各个节点上，并通过各种机制保证其成功 运行。 2、Hadoop MapReduce 处理的数据一般位于底层分布式文件系统中。该系统往往将用户的文件切分成若干个固定大小的block 存储到不同节点上。默认情况下，MapReduce 的每 个Task 处理一个block 。MapReduce 主要由四个组件构成，分别是C l i e n t 、JobTracker 、TaskTracker 和Ta s k ，它们共同保障一个作业的成功运行。一个MapReduce 作业的运行周期是，先在C l i e n t 端被提交JobTracker 上，然后由JobTracker 将作业分解成若干个Ta s k ，并对这些Ta s k 进行调度和监控，以保障这些程序运行成功，而TaskTracker 则启动JobTracker 发来的Ta s k ，并向JobTracker 汇报这些Task 的运行状态和本节点上资源的使用情况。

Mapreduce 的数据模型

MapReduce的数据模型：

- `<key, value>`
- 数据由一条一条的记录组成
- 记录之间是无序的
- 每一条记录有一个key，和一个value
- key: 可以不唯一
- key与value的具体类型和内部结构由程序员决定，系统基本上把它们看作黑匣

图解：



下面以wordcount为例说明MapReduce计算过程： 输入文本：

```
hello world hadoop hdfs hadoop hello hadoop hdfs1
```

map输出：

```
<hello,1>
<world,1>
<hadoop,1>
<hdfs,1>
<hadoop,1>
<hello,1>
<hadoop,1>
<hdfs,1>12345678
```

shuffle(洗牌)过程把key值相同的value合并成list作为reduce输入：

```
<hello,<1,1>>
<world,1>
<hadoop,<1, 1, 1>>
<hdfs,<1,1>>1234
```

reduce输出：

```
<hello,2>
<world,1>
<hadoop,3>
<hdfs,1>1234
```

关于Wordcount运行例子可以参考hadoop helloworld(wordcount),代码解读博客园上有一篇很详细的文章Hadoop集群（第6期）_WordCount运行详解.

MapReduce守护进程：

MapReduce框架主要有两个守护进程，jobtracker和tasktracker。jobtraker是管理者，taskertracker是被管理者。

jobtracker:

- 负责接收用户提交的作业,负责启动跟踪任务执行
- 管理所有作业(job:用户的一个计算请求)
- 将作业分成一系列任务(task:由job拆分出来的执行单元)进行调度
- 将任务指派给tasktracker
- 作业/任务监控,错误处理等

tasktracker:

- 负责执行由jobtracker分配的任务，管理各个任务在每个节点执行情况
- 运行MapTask和ReduceTask
- 与Jobtracker进行交互，执行命令,并汇报任务状态

MapReduce相关概念

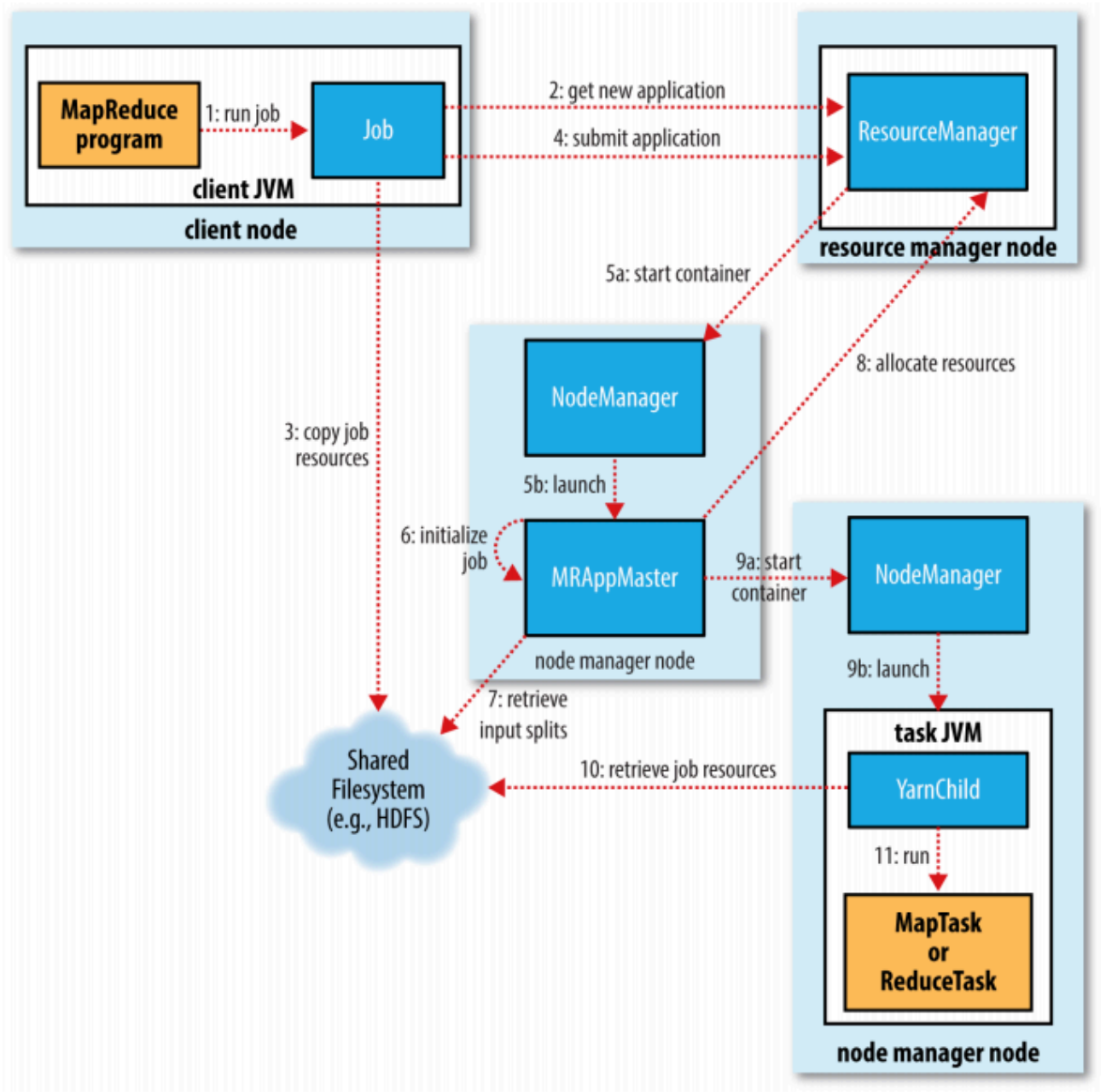
MapTask

- Map引擎
- 分析每条数据记录,将数据解析传递给用户自定义的map()函数
- 将map()函数输出写到本地磁盘（如果是map-only情况，直接输出到HDFS中）

ReduceTask

- Reduce引擎
- 从MapTask上远程读取输入数据
- 对数据进行排序
- 将数据按照分组传递给用户编写的reduce()函数

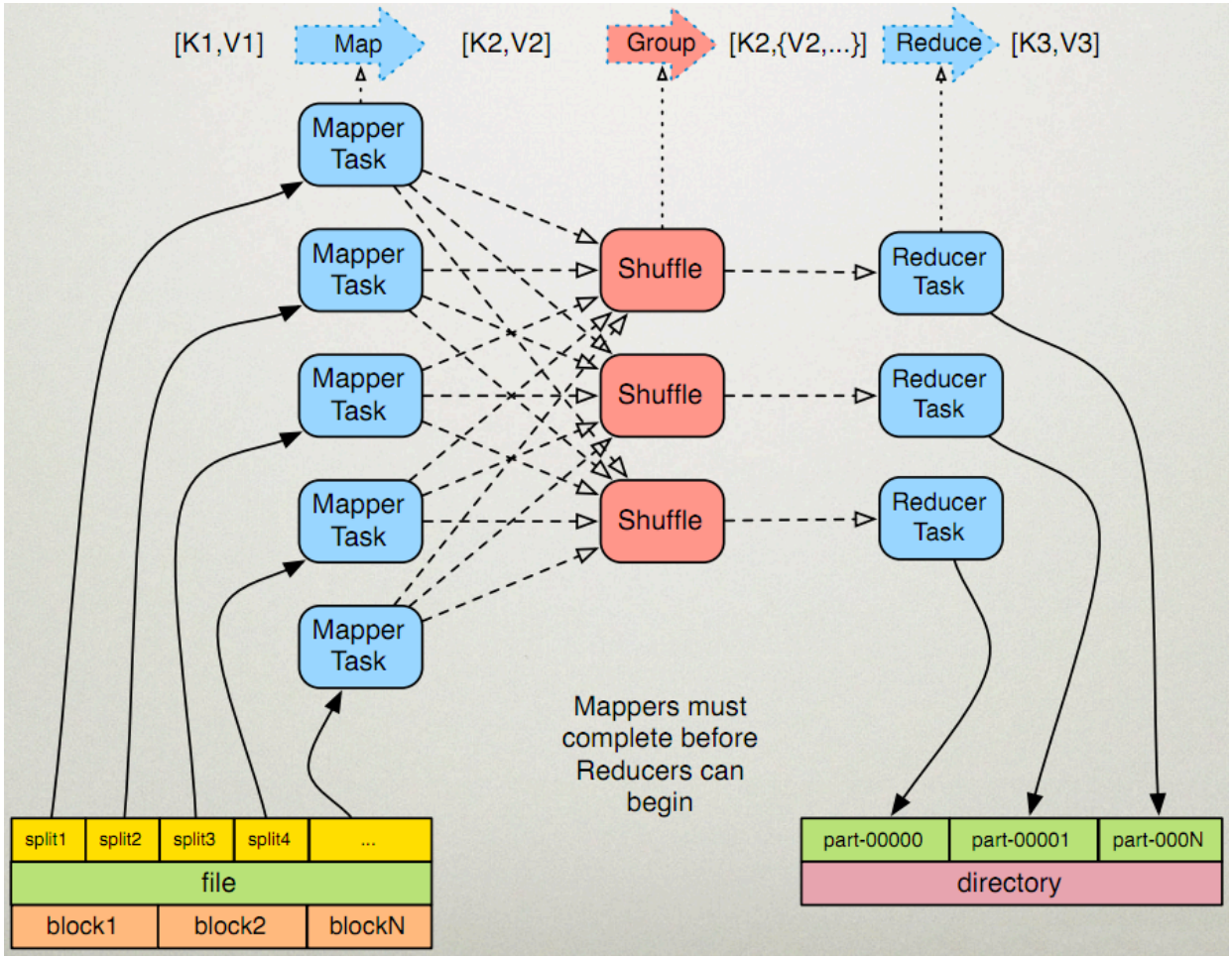
MapReduce运行流程

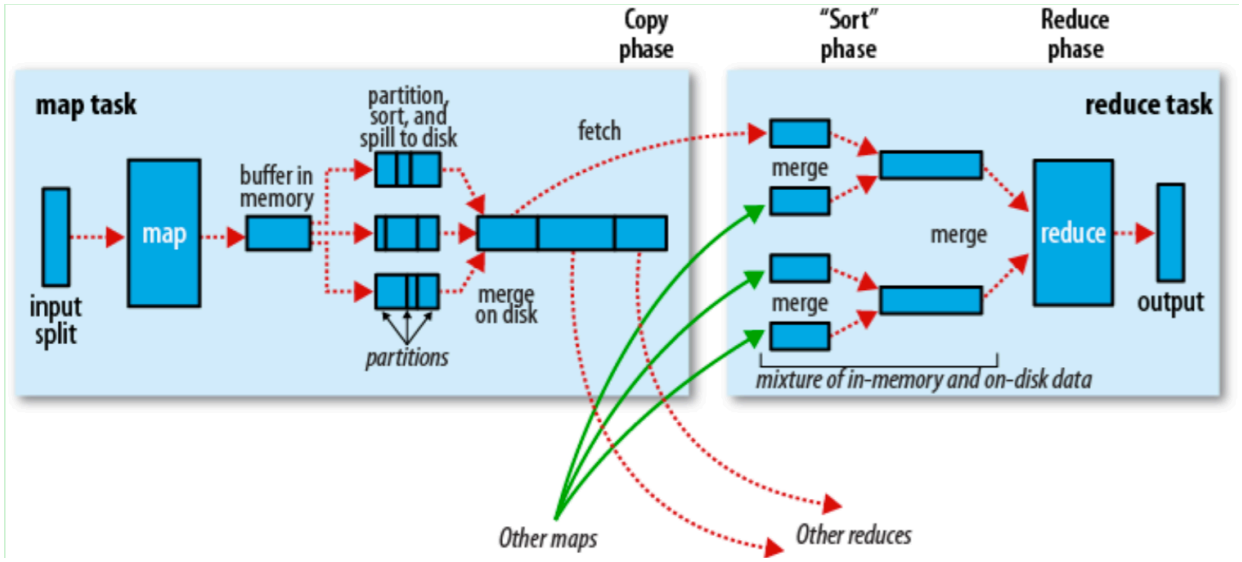


- 1.在客户端启动一个作业
- 2.客户端向JobTracker请求作业号
- 3.客户端向HDFS复制作业的资源文件，这些文件包括打包jar文件，配置文件,以及由客户端计算所得到的输入划分信息。这些文件都存在jobtracker专门为这个job创建的一个文件夹中，以JobID命名。输入划分信息告诉JobTracker应该为这个作业启动多少个map任务等信息
- 4.客户端向JobTracker提交作业,JobTracker接收到作业以后，把它加入到作业队列，然后JobTracker根据自己的调度算法调度到当前作业时，根据输入划分信息,开始为每个划分新建1个task任务，并把task任务分配给tasktracker执行。这里的分配不是随便分配的,而是遵循数据本地化原则的。(数据本地化Data-Local, 就是将map任务分配给拥有该map所要处理数据的DataNode节点，并将jar拷贝到这个节点，这个叫做移动计算，不是移动数据。)
- 5.TaskTracker每个一段时间向JobTracker发送心跳,告诉他自己仍然在运行。同时心跳中还带着其他的一些信息，比如当前map任务完成的进度。当jobtracker接收到最后一个map任务发来的信息的时候,便把作业设置为"成功", 当jobclient查询时，将成功信息返回给用户。

shuffle过程：

shuffle是洗牌或者弄乱的意思，在MapReduce中是指从map task输出到reduce task输入这段过程。


[K1,V1]
Map
[K2,V2]
Group
[K2,{V2,...}]
Reduce
[K3,V3]
Mapper Task
Mapper Task
Mapper Task
Mapper Task
Mapper Task
Shuffle
Shuffle
Shuffle
Reducer Task
Reducer Task
Reducer Task
Mappers must complete before Reducers can begin
split1
split2
split3
split4
...
file
block1
block2
blockN
part-00000
part-00001
part-000N
directory


Copy phase
"Sort" phase
Reduce phase
map task
reduce task
partition, sort, and spill to disk
fetch
buffer in memory
map
input split
merge on disk
partitions
merge
merge
merge
reduce
output
mixture of in-memory and on-disk data
Other maps
Other reduces