**大数据平台架构综述**

康驻关1），戴云鹏2），滕博文2），赵晋锋1）

（1）（北京理工大学计算机学院 北京 100081）

（2）（北京机械工业自动化研究所 北京 100081）

摘要：近几年随着信息技术的快速发展，大数据也成为近几年热门的东西。大数据平台架构把大数据开发中一些通用的，重复使用的基础代码、算法封装为类库，降低大数据的学习门槛，降低开发难度，提高大数据项目的开发效率。大数据平台架构自提出以来也在不断地发展，取得了一些进步。该文首先对大数据平台架构的背景进行介绍，其次对三个常见的系统架构进行了介绍，包括Hadoop架构、Spark架构和Flink架构；最后以腾讯和美团点评大数据平台为例介绍了其应用，以及总结了大数据架构平台的已取得的突破和未来的展望。

关键词：大数据、大数据平台、数据架构、平台架构

**Overview of Big Data Platform Architecture**

Kang ZhuGuan1), Teng BoWen 2), Dai YunPeng2),Zhao JinFeng1)

(1)(School of Computer Science, Beijing Institute of Technology, Beijing 100081, China)  
(2) (Beijing Institute of Machinery Industry Automation, Beijing 100081)

Abstract：In recent years, with the rapid development of information technology, big data has become a hot thing in recent years. The big data platform architecture encapsulates some common and reused basic codes and algorithms in big data development into class libraries, which can lower the learning threshold of big data, reduce the development difficulty and improve the development efficiency of big data projects. Since the big data platform architecture was proposed, it has been developing and making some progress. This paper first introduces the background of big data platform architecture, and then introduces three common system architectures, including Hadoop architecture, Spark architecture and Flink architecture. Finally, Tencent and Meituan comment big data platform as an example to introduce its application，and summarize the breakthroughs and future prospects of the big data architecture platform.

Key words：Big data, Big data platform, Data architecture, Platform architecture

# 任务分工

本文主要由四名同学共同完成，具体的分工情况由以下所述：

康驻关：负责大第二章节和第三章节的编写，主要负责查询大数据架构的研究背景、架构的研究现状以及hadoop架构的相关知识体系。

戴云鹏：负责第四章节的编写，主要负责查询Spark架构的研究现状以及相关知识体系，并对架构的具体应用进行详细的说明。

滕博文：负责第五章节的编写，主要负责查询Flink架构的研究现状以及相关知识体系，并对架构的具体应用进行详细的说明。

赵晋锋：负责第六章节的编写，主要负责对以上相关架构体系进行整理，同时针对上述几种框架列举几个详细的相关应用，方便更加深入的对大数据的架构进行深入研究。

# 2．引言

## 2.1 发展背景

随着电子信息产业的飞速发展，互联网的数据量呈几何倍数增长，现在正处于大数据时代。有数据显示，2014年，我国生产的数据量就已经达到了ZB级别。在未来几年内，中国的数据总数量将呈现翻倍上涨的趋势。面对规模达PB乃至EB级以上的数据，不单单是在数据存储上，也在数据处理上给硬件带来了很大的压力，因此急需引入一项促进大数据发展的架构平台。

## 2.2 发展阶段

萌芽阶段：20世纪90年代到21世纪的样子，数据库技术成熟，数据挖掘理论成熟，也称数据挖掘阶段。

突破阶段：2003——2006年，非结构化的数据大量出现，传统的数据库处理难以应对，也称非结构化数据阶段。

成熟阶段：2006——2009年，谷歌公开发表两篇论文《谷歌文件系统》和《基于集群的简单数据处理:MapReduce》，其核心的技术包括分布式文件系统GFS，分布式计算系统框架MapReduce，分布式锁Chubby，及分布式数据库BigTable，这期间大数据研究的焦点是性能，云计算，大规模的数据集并行运算算法，以及开源分布式架构（Hadoop)。

应用阶段：2009年至今，大数据基础技术成熟之后，学术界及及企业界纷纷开始转向应用研究，2013年大数据技术开始向商业、科技、医疗、政府、教育、经济、交通、物流及社会的各个领域渗透，因此2013年也被称为大数据元年。

根据IBM前首席执行官郭士纳的观点，IT领域每隔十五年就会迎来一次重大变革，以下是近年来IT领域的发展浪潮，如表2.1

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **信息化浪潮** | **发生时间** | **标志** | **解决问题** | **代表企业** |
| 第一次浪潮 | 1980年前后 | 个人计算机 | 信息处理 | Intel、AMD、IBM、苹果、微软、联想、戴尔、惠普等 |
| 第二次浪潮 | 1995年前后 | 互联网 | 信息传输 | 雅虎、谷歌、阿里巴巴、百度、腾讯等 |
| 第三次浪潮 | 2010年前后 | 物联网、云计算和大数据 | 信息爆炸 | 将涌现出一批新的市场标杆企业 |

表2.1 三次信息化浪潮

## 2.3 大数据的特征

大数据主要有四个重要特征，如下图2.1所示：



图2.1 大数据的四个特征

## 2.4 基础技术架构

大数据主要的技术架构分为数据访问层、数据应用层、数据计算平台、数据四个层次，每个层次有主要的架构模块，每层包含的主要模块如下如2.2所示：

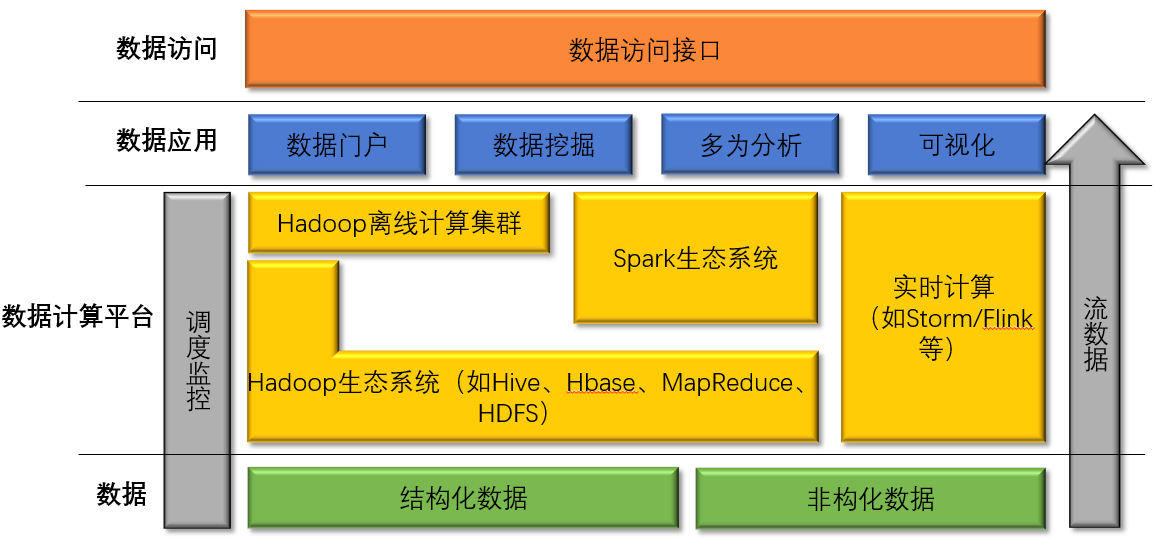


图2.2 大数据的基础技术架构

## 2.5 常见的技术架构

在现实的互联网中有各种各样式的大数据架构，不同公司也构建了自己的技术架构，但是归根到底，这些底层的架构都是基于图1.2所示的基础技术架构进行的深度定制，这里我们列举两个互联网上比较流行的大数据架构，如图2.3和图2.4所示：

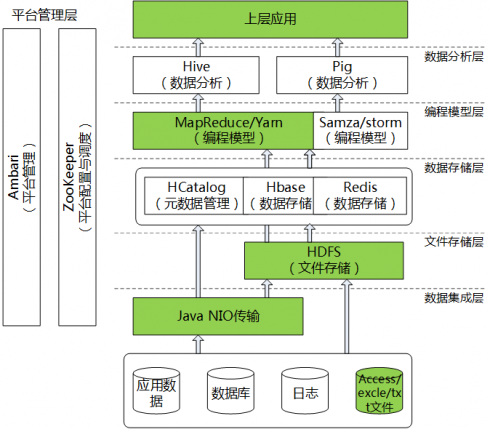
****

图2.3 数据分析架构

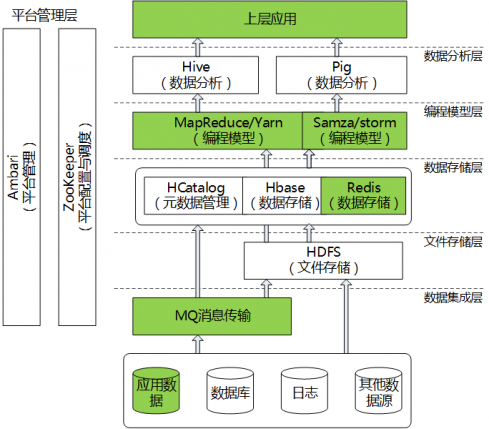
****

图2.4 京东基于Samza的流式计算实践架构

本文首先对大数据平台架构的研究历程和发展现状进行了详细的阐述；其次分别详细的介绍了Hadoop架构、Spark架构、Flink架构和应用场景；最后对大数据平台架构的发展进行了总结和展望。

# 3．Hadoop架构介绍

## 3.1 hadoop生态系统

### 3.1.1 hadoop组成架构

重点组件：

HDFS：分布式文件系统。

YARN：运算资源调度系统。

MAPREDUCE：分布式运算程序开发框架。

HIVE：基于大数据技术（文件系统+运算框架）的SQL数据仓库工具。

HBASE：基于HADOOP的分布式海量数据库。

ZOOKEEPER：分布式协调服务基础组件。

Mahout：基于mapreduce/spark/flink等分布式运算框架的机器学习算法库。

Sqoop：数据导入导出工具。

Flume：日志数据采集框架。

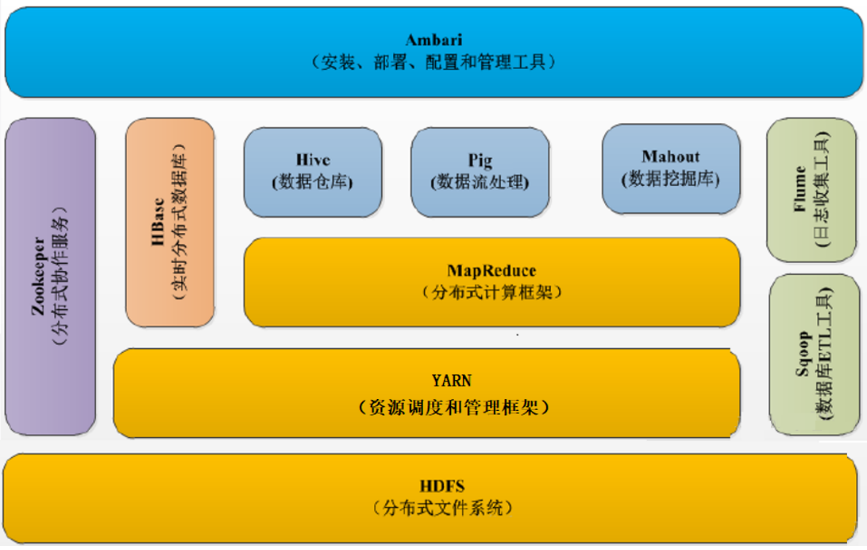


图3.1 hadoop生态系统

### 3.1.2 hadoop三大核心设计

HADOOP的核心组件有:

HDFS（分布式文件系统）

YARN（运算资源调度系统）

MAPREDUCE（分布式运算编程框架）

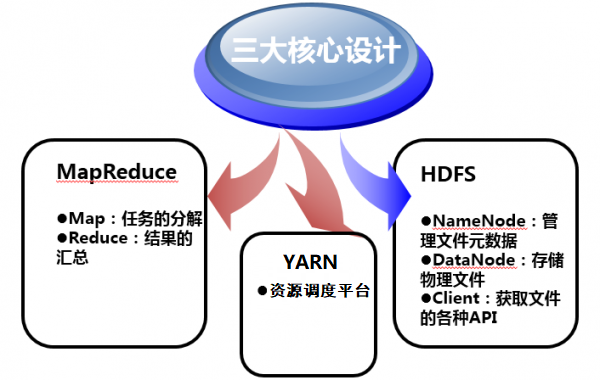


图3.2 hadoop三大核心模块

## 3.2 HDFS架构介绍

### 3.2.1 HDFS架构设计

* 设计思想

分而治之：将大文件、大批量文件，分布式存放在大量服务器上，**以便于采取分而治之的方式对海量数据进行运算分析;**

**首先，它是一个文件系统**，用于存储文件，通过统一的命名空间——目录树来定位文件;

**其次，它是分布式的**，由很多服务器联合起来实现其功能，集群中的服务器有各自的角色;

* 重要特性

1、HDFS中的文件在物理上是**分块存储（block）**，块的大小可以通过配置参数来规定;

2、HDFS文件系统会给客户端提供一个**统一的抽象目录树**，客户端通过路径来访问文件，形如：hdfs://namenode:port/dir-a/dir-b/dir-c/file.data;

3、**目录结构及文件分块信息(元数据)**的管理由namenode节点承担;

4、文件的各个block的存储管理由datanode节点承担;

5、HDFS是设计成适应一次写入，多次读出的场景，且不支持文件的修改;

### 3.2.2 HDFS工作流程

1、HDFS集群分为两大角色：NameNode、DataNode。

2、NameNode负责管理整个文件系统的元数据。

3、DataNode 负责管理用户的文件数据块。

4、文件会按照固定的大小（blocksize）切成若干块后分布式存储在若干台datanode上。

5、每一个文件块可以有多个副本，并存放在不同的datanode上。

6、Datanode会定期向Namenode汇报自身所保存的文件block信息，而namenode则会负责保持文件的副本数量。

7、HDFS的内部工作机制对客户端保持透明，客户端请求访问HDFS都是通过向namenode申请来进行。

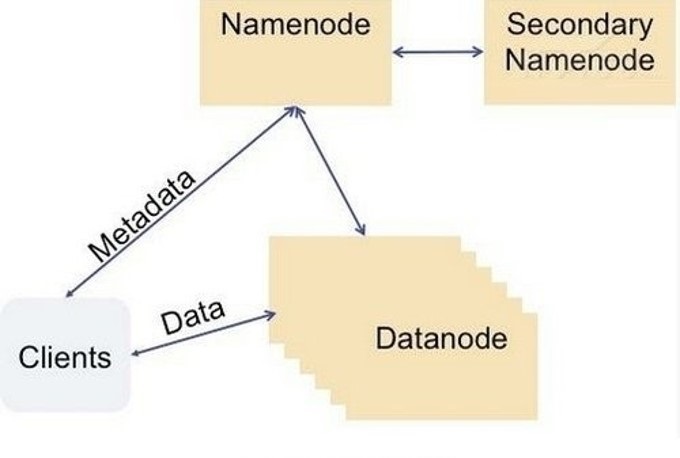


图3.3 HDFS工作流程图

### 3.2.3 HDFS使用案例

**1、HDFS文件数据上传：**客户端要向HDFS写数据，首先要跟namenode通信以确认可以写文件并获得接收文件block的datanode，然后，客户端按顺序将文件逐个block传递给相应datanode，并由接收到block的datanode负责向其他datanode复制block的副本。

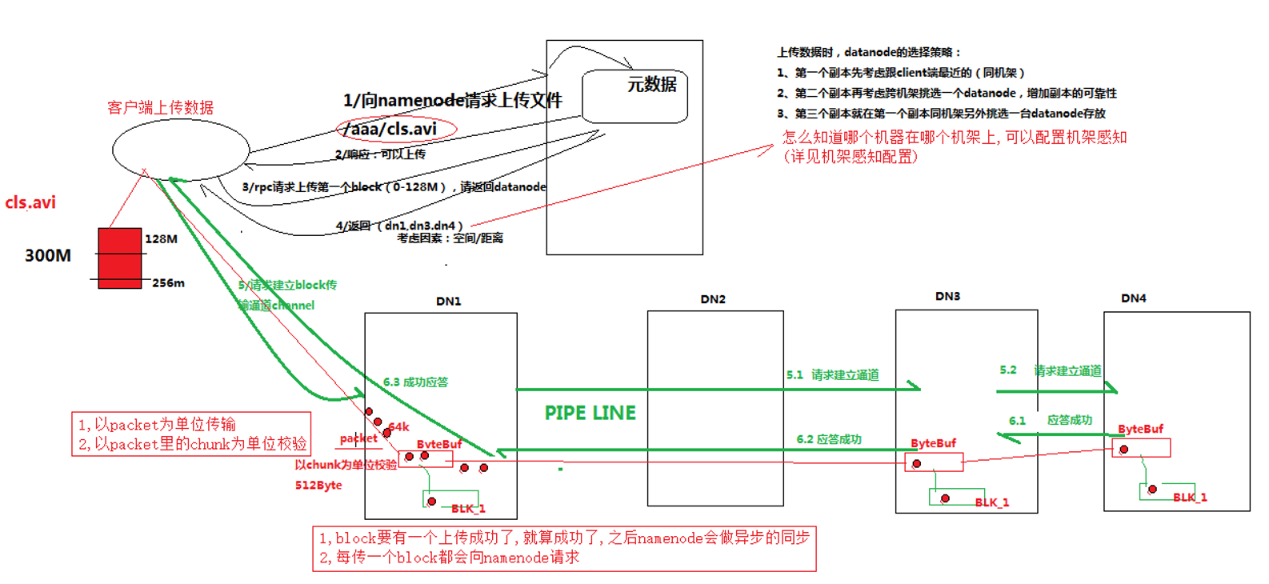


图3.4 HDFS上传数据流程图

1、根namenode通信请求上传文件，namenode检查目标文件是否已存在，父目录是否存在。

2、namenode返回是否可以上传。

3、client请求第一个 block该传输到哪些datanode服务器上。

4、namenode返回3个datanode服务器DN1\DN2\DN3。

5、client请求3台dn中的一台DN1上传数据（本质上是一个RPC调用，建立pipeline），DN1收到请求会继续调用DN2，然后DN2调用DN3，将真个pipeline建立完成，逐级返回客户端。

6、client开始往DN1上传第一个block（先从磁盘读取数据放到一个本地内存缓存），以packet为单位，DN1收到一个packet就会传给DN2，DN2传给DN3；DN1每传一个packet会放入一个应答队列等待应答。

7、当一个block传输完成之后，client再次请求namenode上传第二个block的服务器。

**2、HDFS文件数据读取：**客户端将要读取的文件路径发送给namenode，namenode获取文件的元信息（主要是block的存放位置信息）返回给客户端，客户端根据返回的信息找到相应datanode逐个获取文件的block并在客户端本地进行数据追加合并从而获得整个文件。

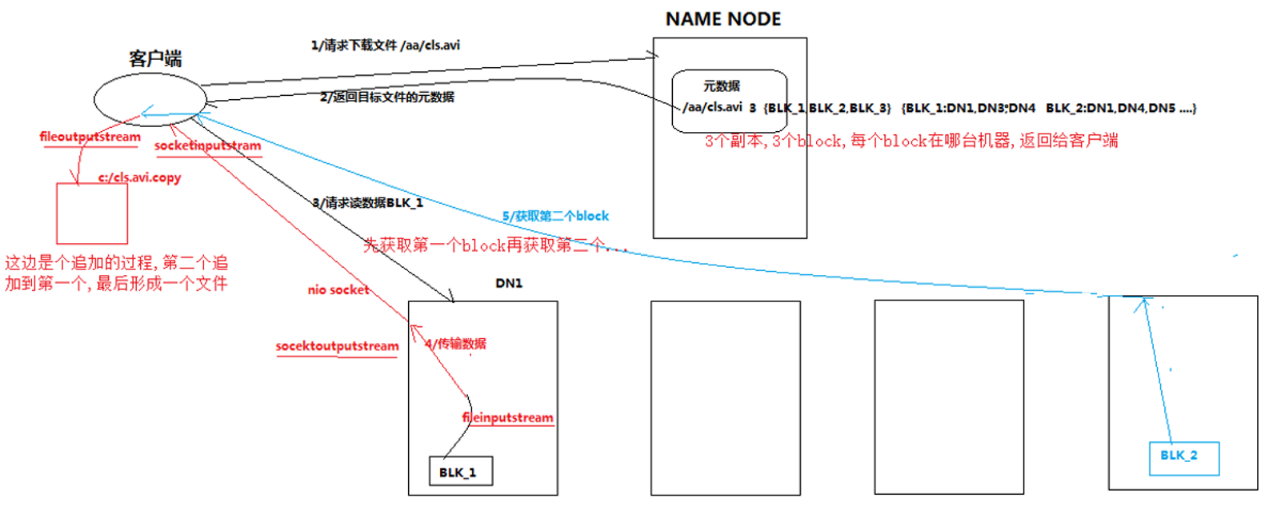


图3.5 HDFS读取数据流程图

**详细流程：**

1、跟namenode通信查询元数据，找到文件块所在的datanode服务器。

2、挑选一台datanode（就近原则，然后随机）服务器，请求建立socket流。

3、datanode开始发送数据（从磁盘里面读取数据放入流，以packet为单位来做校验）。

4、客户端以packet为单位接收，现在本地缓存，然后写入目标文件。

## 3.3 MapReduce架构介绍

### 3.3.1 MapReduce介绍

MapReduce将复杂的、运行于大规模集群上的并行计算过程高度地抽象到了两个函数：Map和Reduce。

MapReduce编程容易，不需要掌握分布式并行编程细节，也可以很容易把自己的程序运行在分布式系统上，完成海量数据的计算。

MapReduce采用“**分而治之**”策略，一个存储在分布式文件系统中的大规模数据集，会被切分成许多独立的分片（split），这些分片可以被多个Map任务并行处理。

**为什么要引入MapReduce？**

（1）海量数据在单机上处理因为硬件资源限制，无法胜任。

（2）而一旦将单机版程序扩展到集群来分布式运行，将极大增加程序的复杂度和开发难度。

（3）引入mapreduce框架后，开发人员可以将绝大部分工作集中在业务逻辑的开发上，而将分布式计算中的复杂性交由框架来处理。

### 3.3.2 MapReduce工作流程

一个完整的mapreduce程序在分布式运行时有三类实例进程：

1、MRAppMaster：负责整个程序的过程调度及状态协调。

2、mapTask：负责map阶段的整个数据处理流程。

3、ReduceTask：负责reduce阶段的整个数据处理流程。



图3.6 MapReduce工作流程

### 3.3.3 MapReduce使用案例

现有多个文件，这些文件分布在不同的服务器节点中，每个文件中分别存储了大量的单词，现在需要统计其中每一个单词出现的总次数。如下图3.7所示：



图3.7 任务描述

解决方案如下图3.8所示：

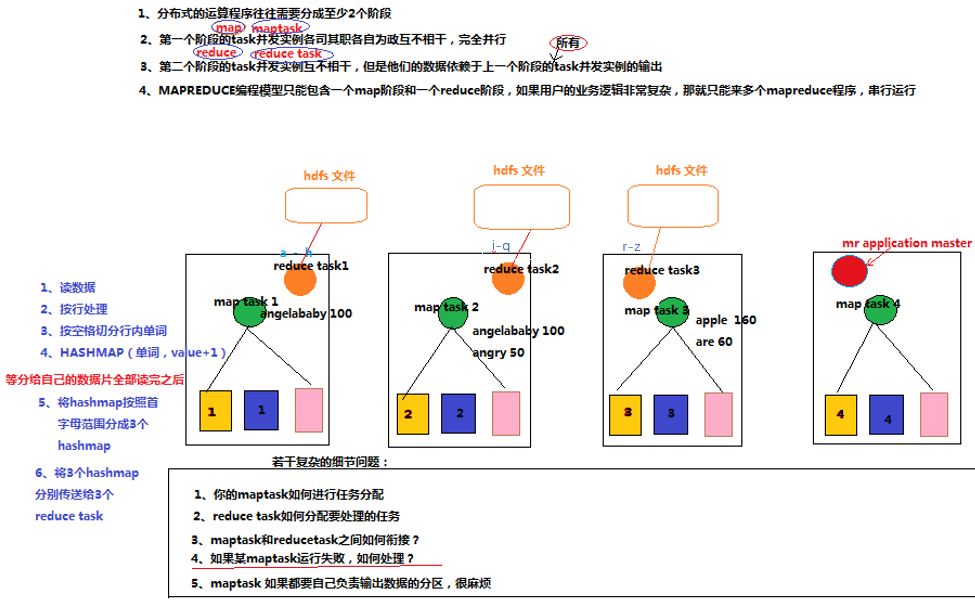


图3.8 MapReduce设计思想

**详细流程：**

1、一个mr程序启动的时候，最先启动的是MRAppMaster，MRAppMaster启动后根据本次job的描述信息，计算出需要的maptask实例数量，然后向集群申请机器启动相应数量的maptask进程。

2、maptask进程启动之后，根据给定的数据切片范围进行数据处理，主体流程为：

a). 利用客户指定的inputformat来获取RecordReader读取数据，形成输入KV对。

b). 将输入KV对传递给客户定义的map()方法，做逻辑运算，并将map()方法输出的KV对收集到缓存。

c). 将缓存中的KV对按照K分区排序后不断溢写到磁盘文件。

3、MRAppMaster监控到所有maptask进程任务完成之后，会根据客户指定的参数启动相应数量的reducetask进程，并告知reducetask进程要处理的数据范围（数据分区）。

4、Reducetask进程启动之后，根据MRAppMaster告知的待处理数据所在位置，从若干台maptask运行所在机器上获取到若干个maptask输出结果文件，并在本地进行重新归并排序，然后按照相同key的KV为一个组，调用客户定义的reduce()方法进行逻辑运算，并收集运算输出的结果KV，然后调用客户指定的outputformat将结果数据输出到外部存储。

## 3.4 YARN架构介绍

Yarn是一个资源调度平台，负责为运算程序提供服务器运算资源，相当于一个分布式的操作系统平台，而mapreduce等运算程序则相当于运行于操作系统之上的应用程序。

### 3.4.1 YARN概述

1. yarn并不清楚用户提交的程序的运行机制
2. yarn只提供运算资源的调度（用户程序向yarn申请资源，yarn就负责分配资源）
3. yarn中的主管角色叫ResourceManager
4. yarn中具体提供运算资源的角色叫NodeManager
5. 这样一来，yarn其实就与运行的用户程序完全解耦，就意味着yarn上可以运行各种类型的分布式运算程序（mapreduce只是其中的一种），比如mapreduce、storm程序，spark程序，tez ……
6. 所以，spark、storm等运算框架都可以整合在yarn上运行，只要他们各自的框架中有符合yarn规范的资源请求机制即可
7. Yarn就成为一个通用的资源调度平台，从此，企业中以前存在的各种运算集群都可以整合在一个物理集群上，提高资源利用率，方便数据共享

### 3.4.2 Yarn中运行运算程序的示例

MAPREDUCE程序的调度过程，如下图3.9所示：

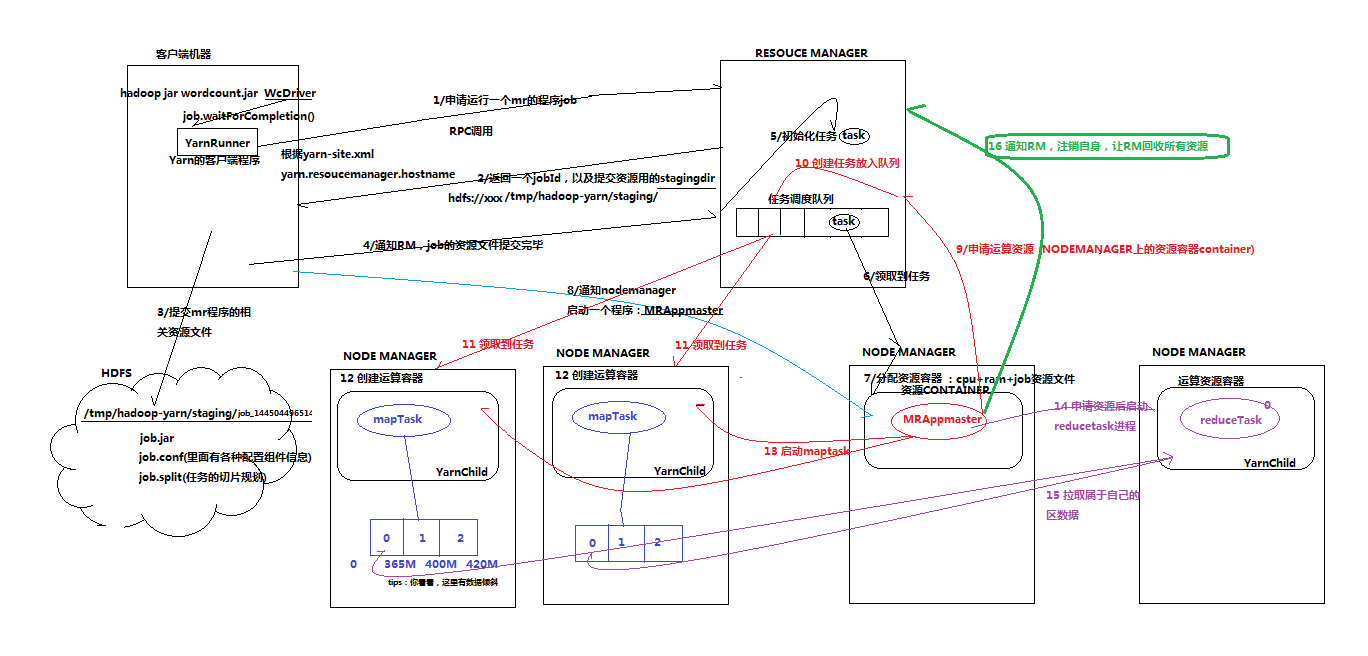


图3.9 程序调动过程

# 4. Spark架构介绍

## 4.1 Spark简介

### 4.1.1 Spark发展历程

Spark是一种快速、通用、可扩展的大数据分析引擎，2009年诞生于加州大学伯克利分校AMPLab，2010年开源，2013年6月成为Apache孵化项目，2014年2月成为Apache顶级项目。目前，Spark生态系统已经发展成为一个包含多个子项目的集合，其中包含SparkSQL、Spark Streaming、GraphX、MLlib等子项目，Spark是基于内存计算的大数据并行计算框架。Spark基于内存计算，提高了在大数据环境下数据处理的实时性，同时保证了高容错性和高可伸缩性，允许用户将Spark部署在大量廉价硬件之上，形成集群。Spark得到了众多大数据公司的支持，这些公司包括Hortonworks、IBM、Intel、Cloudera、MapR、Pivotal、百度、阿里、腾讯、京东、携程、优酷土豆。当前百度的Spark已应用于凤巢、大搜索、直达号、百度大数据等业务；阿里利用GraphX构建了大规模的图计算和图挖掘系统，实现了很多生产系统的推荐算法；腾讯Spark集群达到8000台的规模，是当前已知的世界上最大的Spark集群。

### 4.1.2 Spark特点

Spark提供了一个全面、统一的框架用于管理各种有着不同性质（文本数据、图表数据等）的数据集和数据源（批量数据或实时的流数据）的大数据处理的需求。Spark具有以下四个特点：

1、速度快：与Hadoop的MapReduce相比，Spark基于内存的运算要快100倍以上，基于硬盘的运算也要快10倍以上。Spark实现了高效的DAG执行引擎，可以通过基于内存来高效处理数据流。

2、易用：Spark支持Java、Python和Scala的API，还支持超过80种高级算法，使用户可以快速构建不同的应用。而且Spark支持交互式的Python和Scala的shell，可以非常方便地在这些shell中使用Spark集群来验证解决问题的方法。

3、通用：Spark提供了统一的解决方案。Spark可以用于批处理、交互式查询（Spark SQL）、实时流处理（Spark Streaming）、机器学习（Spark MLlib）和图计算（GraphX）。这些不同类型的处理都可以在同一个应用中无缝使用。Spark统一的解决方案非常具有吸引力，毕竟任何公司都想用统一的平台去处理遇到的问题，减少开发和维护的人力成本和部署平台的物力成本。

4、兼容性： Spark可以非常方便地与其他的开源产品进行融合。比如，Spark可以使用Hadoop的YARN和Apache Mesos作为它的资源管理和调度器，器，并且可以处理所有Hadoop支持的数据，包括HDFS、HBase和Cassandra等。这对于已经部署Hadoop集群的用户特别重要，因为不需要做任何数据迁移就可以使用Spark的强大处理能力。Spark也可以不依赖于第三方的资源管理和调度器，它实现了Standalone作为其内置的资源管理和调度框架，这样进一步降低了Spark的使用门槛，使得所有人都可以非常容易地部署和使用Spark。此外，Spark还提供了在EC2上部署Standalone的Spark集群的工具。

## 4.2 Spark的体系架构



图4.1 Spark的体系架构

1、Apache Spark被称为Spark core，是其最核心的部分，包含了Spark最基本、最核心的功能和基本分布式算子。

2、Spark SQL：Spark Streaming类似于Apache Storm，用于流式数据的处理。根据其官方文档介绍，Spark Streaming有高吞吐量和容错能力强等特点。Spark Streaming支持的数据输入源很多，例如：Kafka、Flume、Twitter、ZeroMQ和简单的TCP套接字等等。数据输入后可以用Spark的高度抽象原语如：map、reduce、join、window等进行运算。而结果也能保存在很多地方，如HDFS，数据库等。另外Spark Streaming也能和MLlib（机器学习）以及Graphx完美融合。

3、Spark Streaming是一种构建在Spark上的实时计算框架，它扩展了Spark处理大规模流式数据的能力。

首先，Spark Streaming把实时输入数据流以时间片Δt （如1秒）为单位切分成块，Spark Streaming会把每块数据作为一个RDD，并使用RDD操作处理每一小块数据，每个块都会生成一个Spark Job处理，最终结果也返回多块。 在Spark Streaming中，则通过操作DStream（表示数据流的RDD序列）提供的接口，这些接口和RDD提供的接口类似。

正如Spark Streaming最初的目标一样，它通过丰富的API和基于内存的高速计算引擎让用户可以结合流式处理，批处理和交互查询等应用。因此Spark Streaming适合一些需要历史数据和实时数据结合分析的应用场合。当然，对于实时性要求不是特别高的应用也能完全胜任，另外通过RDD的数据重用机制可以得到更高效的容错处理。

4、MLlib是一些常用机器学习算法和工具中非常突出的，分类，回归，聚类，协同过滤，降维，以及相关的优化原语。许多标准的机器学习方法可以归结为一个凸优化问题，例如一项任务，去找到一个凸面函数f的最小值，这个凸面函数依赖于可变的Vectorw向量，这个向量在一个Node 中叫做Weights权值，我们可以把这个在客观函数里写成最优化问题。MLlib 包括的评价指标有：精确度，回收，F值，ROC，精密回收曲线，曲线下面积(AUC)，AUC常用来比较模型，而精确度，回收，F值，ROC用来确定阀值。

5、GraphX是构建在Spark之上的图计算框架，它使用RDD来存储图数据，并提供了实用的图操作方法。由于RDD的与生俱来的特性，GraphX高效地实现了图的分布式存储和处理，可以应用于社交网络等大规模的图计算场景。GraphX在图顶点信息和边信息存储上做了优化，使得图计算框架性能相对于原生RDD实现得以较大提升，接近或到达GraphLab等专业图计算平台的性能。GraphX最大的贡献是，在Spark之上提供一栈式数据解决方案，可以方便且高效地完成图计算的一整套流水作业。

图结构中最基本的要素是顶点和边。GraphX描述的是拥有顶点属性和边属性的有向图。GraphX提供顶点（Vertex）、边（Edge）、边三元组（EdgeTriplet）三种视图。GraphX的各种图操作也是在这三种视图上完成的。顶点包含顶点ID和顶点数据（VD）；边包含源顶点ID（srcId）、目的顶点ID（dstId）和边数据（ED）。边三元组是边的扩展，它在边的基础上提供了边的源顶点数据、目的顶点数据。在许多图计算操作中，需要将边数据以及边所连接的顶点数据一起组成边三元组，然后在边三元组上进行操作。

## 4.3 Spark计算模型

### 4.3.1 RDD简介

RDD（Resilient Distributed Dataset）叫做分布式数据集，是Spark中最基本的数据抽象，它代表一个不可变、可分区、里面的元素可并行计算的集合。RDD具有数据流模型的特点：自动容错、位置感知性调度和可伸缩性。RDD允许用户在执行多个查询时显式地将工作集缓存在内存中，后续的查询能够重用工作集，这极大地提升了查询速度。

### 4.3.2 RDD的属性

1、一组分片（Partition），即数据集的基本组成单位。对于RDD来说，每个分片都会被一个计算任务处理，并决定并行计算的粒度。用户可以在创建RDD时指定RDD的分片个数，如果没有指定，那么就会采用默认值。默认值就是程序所分配到的CPU Core的数目。

2、一个计算每个分区的函数。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。compute函数会对迭代器进行复合，不需要保存每次计算的结果。

3、RDD之间的依赖关系。RDD的每次转换都会生成一个新的RDD，所以RDD之间就会形成类似于流水线一样的前后依赖关系。在部分分区数据丢失时，Spark可以通过这个依赖关系重新计算丢失的分区数据，而不是对RDD的所有分区进行重新计算。

4、一个Partitioner，即RDD的分片函数。当前Spark中实现了两种类型的分片函数，一个是基于哈希的HashPartitioner，另外一个是基于范围的RangePartitioner。只有对于于key-value的RDD，才会有Partitioner，非key-value的RDD的Parititioner的值是None。Partitioner函数不但决定了RDD本身的分片数量，也决定了parent RDD Shuffle输出时的分片数量。

5、一个列表，存储存取每个Partition的优先位置（preferred location）。对于一个HDFS文件来说，这个列表保存的就是每个Partition所在的块的位置。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。

### 4.3.3 RDD操作类型

RDD支持的两种类型的操作：

1、转换（Transformation）：现有的RDD通过转换生成一个新的RDD。lazy模式，延迟执行。转换的函数：map，filter，flatmap，groupByKey，reduceBykey，aggregateByKey，union，join，coalesce等。

2、动作（Action）：在RDD上运行计算，并返回结果给驱动程序（Drive）或写入文件系统。动作操作函数：reduce,collect,count,frist,take,countByKey以及foreach等等。collect该方法把数据收集到driver端 Array数组类型。所有的transfromation只有遇到action才能执行。当触发执行action之后，数据类型就不再是RDD了，数据就会存到指定的文件系统中，或者直接打印结果或者收集起来。



图4.2 RDD操作流程

### 4.3.4 RDD的运行逻辑：

在Spark应用中，整个执行流程在逻辑上运算之间会形成有向无环图。Action算子触发之后会将所有累积的算子形成一个有向无环图，然后由调度器调度该图上的任务进行运算。

Spark的调度方式与MapReduce有所不同。Spark根据RDD之间不同的依赖关系切分形成不同的阶段（Stage），一个阶段包含一系列函数进行流水线执行。如下图所示：

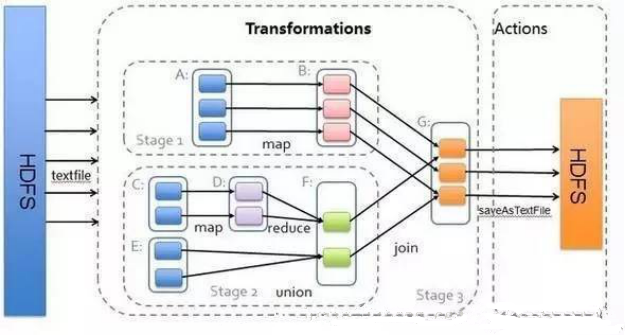


图4.3 RDD运行逻辑

图中的A、B、C、D、E、F、G，分别代表不同的RDD，RDD内的一个方框代表一个数据块。数据从HDFS输入Spark，形成RDD A和RDD C，RDD C上执行map操作，转换为RDD D，RDD B和RDD F进行join操作转换为G，而在B到G的过程中又会进行Shuffle。最后RDD G通过函数saveAsSequenceFile输出保存到HDFS中。

## 4.4 spark运行流程介绍

### 4.4.1 spark运行流程

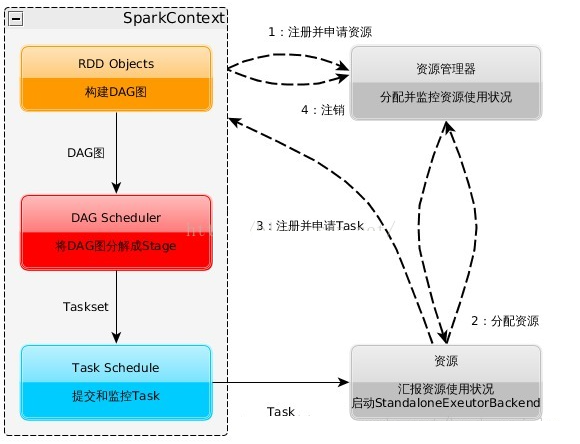


图4.4 spark运行流程

1、构建Spark Application的运行环境，启动SparkContext。

2、SparkContext向资源管理器（可以是Standalone，Mesos，Yarn）申请运Executor资源，并启动StandaloneExecutorbackend构建Spark Application的运行环境，启动SparkContext。

3、Executor向SparkContext申请Task。

4、SparkContext将应用程序分发给Executor。

5、SparkContext构建成DAG图，将DAG图分解成Stage、将Taskset发送给Task Scheduler，最后由Task Scheduler将Task发送给Executor运行。

6、Task在Executor上运行，运行完释放所有资源。

常用术语：

Application: Appliction都是指用户编写的Spark应用程序，其中包括一个Driver功能的代码和分布在集群中多个节点上运行的Executor代码。

Driver:Spark中的Driver即运行上述Application的main函数并创建SparkContext，创建SparkContext的目的是为了准备Spark应用程序的运行环境，在Spark中有SparkContext负责与ClusterManager通信，进行资源申请、任务的分配和监控等，当Executor部分运行完毕后，Driver同时负责将SparkContext关闭，通常用SparkContext代表Driver。

Executor:某个Application运行在worker节点上的一个进程，该进程负责运行某些Task， 并且负责将数据存到内存或磁盘上，每个Application都有各自独立的一批Executor，在Spark on Yarn模式下，其进程名称为CoarseGrainedExecutor Backend。一个CoarseGrainedExecutor Backend有且仅有一个Executor对象，负责将Task包装成taskRunner,并从线程池中抽取一个空闲线程运行Task， 这个每一个oarseGrained Executor Backend能并行运行Task的数量取决与分配给它的cpu个数。

Cluter Manager指的是在集群上获取资源的外部服务。目前有三种类型。

Standalone:spark原生的资源管理，由Master负责资源的分配。

Apache Mesos:与hadoop MR兼容性良好的一种资源调度框架。

Hadoop Yarn:主要是指Yarn中的ResourceManager。

Worker:集群中任何可以运行Application代码的节点，在Standalone模式中指的是通过slave文件配置的Worker节点，在Spark on Yarn模式下就是NoteManager节点。

Task:被送到某个Executor上的工作单元，但hadoopMR中的MapTask和ReduceTask概念一样，是运行Application的基本单位，多个Task组成一个Stage，而Task的调度和管理等是由TaskScheduler负责。

Job:包含多个Task组成的并行计算，往往由Spark Action触发生成， 一个Application中往往会产生多个Job。

Stage:每个Job会被拆分成多组Task，作为一个TaskSet，其名称为Stage，Stage的划分和调度是有DAGScheduler来负责的，Stage有非最终的Stage（Shuffle Map Stage）和最终的Stage（Result Stage）两种，Stage的边界就是发生shuffle的地方。

DAGScheduler:根据Job构建基于Stage的DAG（Directed Acyclic Graph有向无环图)，并提交Stage给TASkScheduler。 其划分Stage的依据是RDD之间的依赖的关系找出开销最小的调度方法。

TASKSedulter:将TaskSET提交给worker运行，每个Executor运行什么Task就是在此处分配的. TaskScheduler维护所有TaskSet，当Executor向Driver发生心跳时，TaskScheduler会根据资源剩余情况分配相应的Task。另外TaskScheduler还维护着所有Task的运行标签，重试失败的Task。

### 4.4.2 Spark运行特点

每个Application获取专属的executor进程，该进程在Application期间一直驻留，并以多线程方式运行Task。这种Application隔离机制是有优势的，无论是从调度角度看（每个Driver调度他自己的任务），还是从运行角度看（来自不同Application的Task运行在不同JVM中），当然这样意味着Spark Application不能跨应用程序共享数据，除非将数据写入外部存储系统Spark与资源管理器无关，只要能够获取executor进程，并能保持相互通信就可以了提交SparkContext的Client应该靠近Worker节点（运行Executor的节点），最好是在同一个Rack里，因为Spark Application运行过程中SparkContext和Executor之间有大量的信息交换。

## 4.5 Hadoop与Spark的对比

### 4.5.1 Hadoop与Spark的执行流程



图4.5 Hadoop与Spark的执行流程对比

### 4.5.2 Hadoop与Spark执行逻辑回归的时间对比

使用Hadoop进行迭代计算非常耗资源。Spark将数据载入内存后，之后的迭代计算都可以直接使用内存中的中间结果作运算，避免了从磁盘中频繁读取数据。



图4.6 Hadoop与Spark执行逻辑回归的时间对比

### 4.5.3 Spark与Hadoop相比具有的优势

**Spark优势：**

1、Spark的计算模式也属于MapReduce，但不局限于Map和Reduce操作，还提供了多种数据集操作类型，编程模型比Hadoop MapReduce更灵活。

2、Spark提供了内存计算，可将中间结果放到内存中，对于迭代运算效率更高。

3、Spark基于DAG的任务调度执行机制，要优于Hadoop MapReduce的迭代执行机制 。

**Hadoop缺点：**

1、表达能力有限

2、磁盘IO开销大

3、延迟高

4、任务之间的衔接涉及IO开销

5、在前一个任务执行完成之前，其他任务就无法开始，难以胜任复杂、多阶段的计算任务

# 5 Flink架构介绍

## 5.1 Flink简介

这几年大数据的飞速发展，出现了很多热门的开源社区，其中著名的有Hadoop、Storm，以及后来的Spark，他们都有着各自专注的应用场景。Spark掀开了内存计算的先河，也以内存为赌注，赢得了内存计算的飞速发展。Spark的火热或多或少的掩盖了其他分布式计算的系统身影。就像Flink，也就在这个时候默默的发展着。国外的一些人把大数据计算引擎分成了4代。

首先第一代的计算引擎，无疑就是Hadoop承载的MapReduce。这里大家应该都不会对MapReduce陌生，它将计算分为两个阶段，分别为Map和Reduce。对于上层应用来说，就不得不想方设法去拆分算法，甚至于不得不在上层应用实现多个Job的串联，以完成一个完整的算法，例如迭代计算。

其次，由于这样的弊端，催生了支持DAG框架的产生。因此，支持DAG的框架被划分为第二代计算引擎。如Tez以及更上层的Oozie。这里我们不去细究各种DAG实现之间的区别，不过对于当时的Tez和Oozie来说，大多还是批处理的任务。

接下来就是以Spark为代表的第三代的计算引擎。第三代计算引擎的特点主要是Job内部的DAG支持（不跨越Job），以及强调的实时计算。在这里，很多人也会认为第三代计算引擎也能够很好的运行批处理的Job。

随着第三代计算引擎的出现，促进了上层应用快速发展，例如各种迭代计算的性能以及对流计算和SQL等的支持。Flink的诞生就被归在了第四代。这应该主要表现在Flink对流计算的支持，以及更一步的实时性上面。当然Flink也可以支持Batch的任务，以及DAG的运算。

很多人可能都是在2015年才听到Flink这个词，其实早在2008年，Flink的前身已经是柏林理工大学一个研究性项目。在2014被Apache孵化器所接受，然后迅速地成为了ASF（Apache Software Foundation）的顶级项目之一。Flink的最新版本目前已经更新到了1.7.1了，在很多人感慨Spark的快速发展的同时，或许我们也该为Flink的发展速度点个赞。

Flink是一个针对流数据和批数据的分布式处理引擎。它主要是由Java代码实现。目前主要还是依靠开源社区的贡献而发展。对Flink而言，其所要处理的主要场景就是流数据，批数据只是流数据的一个极限特例而已。再换句话说，Flink会把所有任务当成流来处理，这也是其最大的特点。

Flink可以支持本地的快速迭代，以及一些环形的迭代任务。并且Flink可以定制化内存管理。在这点，如果要对比Flink和Spark的话，Flink并没有将内存完全交给应用层。这也是为什么Spark相对于Flink，更容易出现OOM的原因（out of memory）。就框架本身与应用场景来说，Flink更相似与Storm。如果之前了解过Storm或者Flume的读者，可能会更容易理解Flink的架构和很多概念。

## 5.2 基本架构

Flink系统的架构与Spark类似，是一个基于Master-Slave风格的架构，如下图5.1所示：

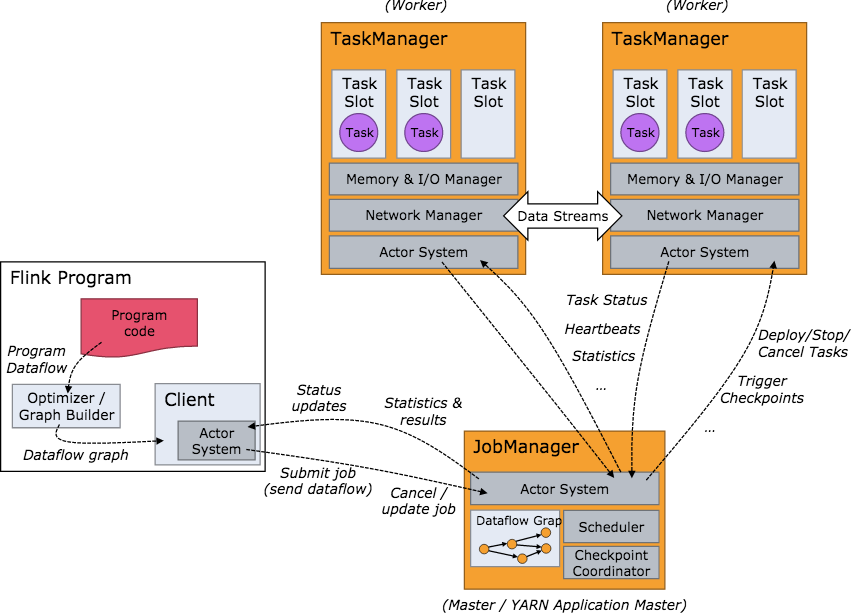


图5.1 基于Master-Slave风格的架构

如上图所示，Flink系统主要包含如下3个主要的进程：Client、JobManager和TaskManager。Client用来提交任务给JobManager，JobManager分发任务给TaskManager去执行，然后TaskManager会心跳的汇报任务状态。看到这里，有的人应该已经有种回到Hadoop一代的错觉。确实，从架构图去看，JobManager很像当年的JobTracker，TaskManager也很像当年的TaskTracker。然而有一个最重要的区别就是TaskManager之间是是流（Stream）。其次，Hadoop一代中，只有Map和Reduce之间的Shuffle，而对Flink而言，可能是很多级，并且在TaskManager内部和TaskManager之间都会有数据传递，而不像Hadoop，是固定的Map到Reduce。

### 5.2.1 三个进程

#### 5.2.1.1 JobManager

JobManager是Flink系统的协调者，它负责接收Flink Job，调度组成Job的多个Task的执行。同时，JobManager还负责收集Job的状态信息，并管理Flink集群中从节点TaskManager。JobManager所负责的各项管理功能，它接收到并处理的事件主要包括：

* RegisterTaskManager

在Flink集群启动的时候，TaskManager会向JobManager注册，如果注册成功，则JobManager会向TaskManager回复消息AcknowledgeRegistration。

* SubmitJob

Flink程序内部通过Client向JobManager提交Flink Job，其中在消息SubmitJob中以JobGraph形式描述了Job的基本信息。

* CancelJob

请求取消一个Flink Job的执行，CancelJob消息中包含了Job的ID，如果成功则返回消息CancellationSuccess，失败则返回消息CancellationFailure。

* UpdateTaskExecutionState

TaskManager会向JobManager请求更新ExecutionGraph中的ExecutionVertex的状态信息，更新成功则返回true。

* RequestNextInputSplit

运行在TaskManager上面的Task，请求获取下一个要处理的输入Split，成功则返回NextInputSplit。

* JobStatusChanged

ExecutionGraph向JobManager发送该消息，用来表示Flink Job的状态发生的变化，例如：RUNNING、CANCELING、FINISHED等。

#### 5.2.1.2 TaskManager

TaskManager也是一个Actor，它是实际负责执行计算的Worker，在其上执行Flink Job的一组Task。每个TaskManager负责管理其所在节点上的资源信息，如内存、磁盘、网络，在启动的时候将资源的状态向JobManager汇报。TaskManager端可以分成两个阶段：

* 注册阶段

TaskManager会向JobManager注册，发送RegisterTaskManager消息，等待JobManager返回AcknowledgeRegistration，然后TaskManager就可以进行初始化过程。

* 可操作阶段

该阶段TaskManager可以接收并处理与Task有关的消息，如SubmitTask、CancelTask、FailTask。如果TaskManager无法连接到JobManager，这是TaskManager就失去了与JobManager的联系，会自动进入“注册阶段”，只有完成注册才能继续处理Task相关的消息。

#### 5.2.1.3 Client

当用户提交一个Flink程序时，会首先创建一个Client，该Client首先会对用户提交的Flink程序进行预处理，并提交到Flink集群中处理，所以Client需要从用户提交的Flink程序配置中获取JobManager的地址，并建立到JobManager的连接，将Flink Job提交给JobManager。Client会将用户提交的Flink程序组装一个JobGraph， 并且是以JobGraph的形式提交的。一个JobGraph是一个Flink Dataflow，它由多个JobVertex组成的DAG。其中，一个JobGraph包含了一个Flink程序的如下信息：JobID、Job名称、配置信息、一组JobVertex等。

### 5.2.2 组件栈

Flink是一个分层架构的系统，每一层所包含的组件都提供了特定的抽象，用来服务于上层组件。Flink分层的组件栈，如下图2.2所示：

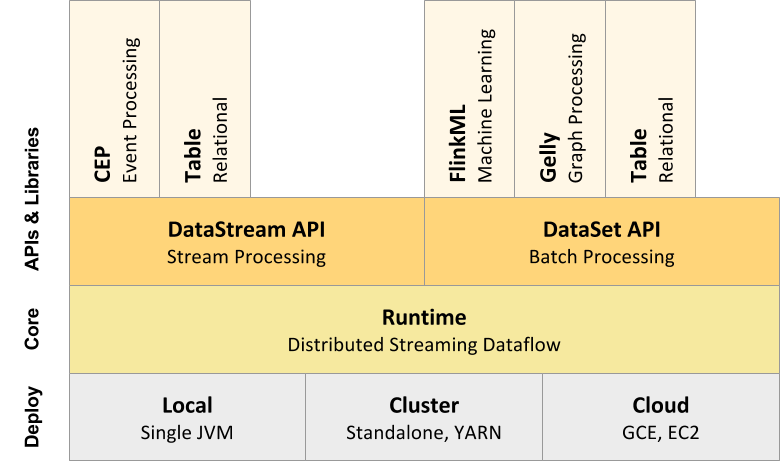


图5.2 Flink分层的组件栈

下面，我们自下而上，分别针对每一层进行解释说明：

#### 5.2.2.1 Deployment层

该层主要涉及了Flink的部署模式，Flink支持多种部署模式：本地、集群（Standalone /YARN）、云（GCE/EC2）。Standalone部署模式与Spark类似，这里，我们看一下Flink on YARN的部署模式，如下图5.3所示：

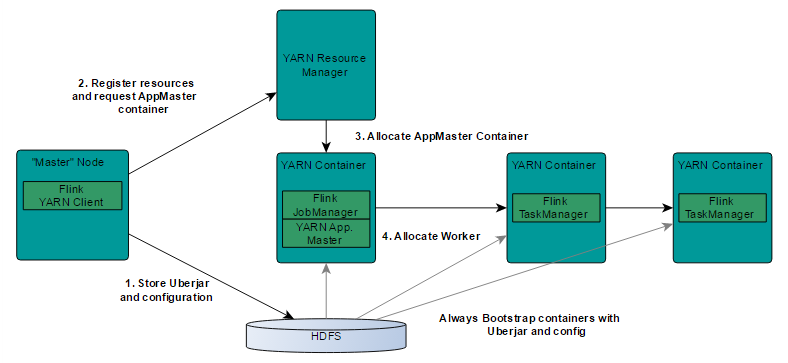


图5.3 Flink on YARN的部署模式

了解YARN的话，对上图的原理非常熟悉，实际Flink也实现了满足在YARN集群上运行的各个组件：Flink YARN Client负责与YARN RM通信协商资源请求，Flink JobManager和Flink TaskManager分别申请到Container去运行各自的进程。通过上图可以看到，YARN AM与Flink JobManager在同一个Container中，这样AM可以知道Flink JobManager的地址，从而AM可以申请Container去启动Flink TaskManager。待Flink成功运行在YARN集群上，Flink YARN Client就可以提交Flink Job到Flink JobManager，并进行后续的映射、调度和计算处理。

在图中可以看出，Flink与Yarn的关系与MapReduce和Yarn的关系是一样的。Flink通过Yarn的接口实现了自己的App Master。当在Yarn中部署了Flink，Yarn就会用自己的Container来启动Flink的JobManager（也就是App Master）和TaskManager。Flink在YARN上面运行流程具体如下：

（1）首先Flink Yarn Client会检验系统是否有足够的资源来启动YARN集群，如果资源足够，它就会将Jar包和配置文件上传到HDFS。

（2）Flink YARN CLient首先与Yarn ResourceManager进行通信，申请启动applicationMaster，在FLink Yarn的集群中，ApplicationMaster与Flink JobManager被封装在同一个container中。

（3）ApplicationMaster在启动的过程中，会和Yarn的ResourceManager进行交互，向ResourceManager申请所需要的TaskManager Container。当ApplicationMaster申请到TaskManager Container以后，它会在所对应的NodeManager节点上启动TaskManager进程。

（4）由于ApplicationMaster和Flink JobManager是封装在同一个Container中的，所以ApplicationMaster会将JobManager的IPC地址，通过HDFS共享的方式通知到各个TaskManager上。TaskManager启动成功以后，就会向JobManager进行注册。

（5）当所有的TaskManager都向JobManager注册成功以后，Flink基于Yarn的集群就启动成功了。Flink Yarn Client就可以提交FLink job到Flink JobManager上，然后进程后面的映射、调度、计算等处理。

#### 5.2.2.2 RunTime层

Runtime层提供了支持Flink计算的全部核心实现，比如：支持分布式Stream处理、JobGraph到ExecutionGraph的映射、调度等等，为上层API层提供基础服务。

#### 5.2.2.3 API层

API层主要实现了面向无界Stream的流处理和面向Batch的批处理API，其中面向流处理对应DataStream API，面向批处理对应DataSet API。

#### 5.2.2.4 Libraries层

该层也可以称为Flink应用框架层，根据API层的划分，在API层之上构建的满足特定应用的实现计算框架，也分别对应于面向流处理和面向批处理两类。面向流处理支持：CEP（复杂事件处理）、基于SQL-like的操作（基于Table的关系操作）；面向批处理支持：FlinkML（机器学习库）、Gelly（图处理）。

## 5.3 内部原理

### 5.3.1 容错机制

Flink基于Checkpoint机制实现容错，它的原理是不断地生成分布式Streaming数据流Snapshot。在流处理失败时，通过这些Snapshot可以恢复数据流处理。理解Flink的容错机制，首先需要了解一下Barrier这个概念：  
 Stream Barrier是Flink分布式Snapshotting中的核心元素，它会作为数据流的记录被同等看待，被插入到数据流中，将数据流中记录的进行分组，并沿着数据流的方向向前推进。每个Barrier会携带一个Snapshot ID，属于该Snapshot的记录会被推向该Barrier的前方。因为Barrier非常轻量，所以并不会中断数据流。带有Barrier的数据流，如下图5.3所示：

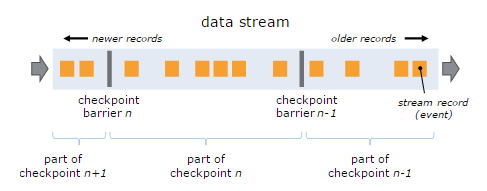


图5.5 带有Barrier的数据流

基于上图，我们通过如下要点来说明：

出现一个Barrier，在该Barrier之前出现的记录都属于该Barrier对应的Snapshot，在该Barrier之后出现的记录属于下一个Snapshot。

来自不同Snapshot多个Barrier可能同时出现在数据流中，也就是说同一个时刻可能并发生成多个Snapshot。

当一个中间（Intermediate）Operator接收到一个Barrier后，它会发送Barrier到属于该Barrier的Snapshot的数据流中，等到Sink Operator接收到该Barrier后会向Checkpoint Coordinator确认该Snapshot，直到所有的Sink Operator都确认了该Snapshot，才被认为完成了该Snapshot。

这里还需要强调的是，Snapshot并不仅仅是对数据流做了一个状态的Checkpoint，它也包含了一个Operator内部所持有的状态，这样才能够在保证在流处理系统失败时能够正确地恢复数据流处理。也就是说，如果一个Operator包含任何形式的状态，这种状态必须是Snapshot的一部分。

Operator的状态包含两种：一种是系统状态，一个Operator进行计算处理的时候需要对数据进行缓冲，所以数据缓冲区的状态是与Operator相关联的，以窗口操作的缓冲区为例，Flink系统会收集或聚合记录数据并放到缓冲区中，直到该缓冲区中的数据被处理完成；另一种是用户自定义状态（状态可以通过转换函数进行创建和修改），它可以是函数中的Java对象这样的简单变量，也可以是与函数相关的Key/Value状态。

对于具有轻微状态的Streaming应用，会生成非常轻量的Snapshot而且非常频繁，但并不会影响数据流处理性能。Streaming应用的状态会被存储到一个可配置的存储系统中，例如HDFS。在一个Checkpoint执行过程中，存储的状态信息及其交互过程，如下图5.7所示：

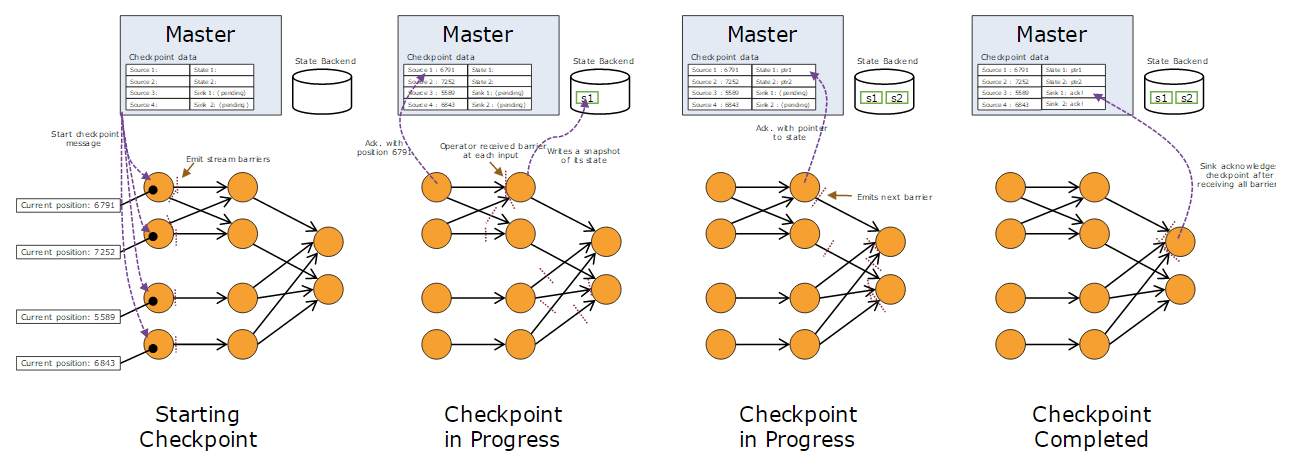


图5.6 存储的状态信息及其交互过程图带

在Checkpoint过程中，还有一个比较重要的操作——Stream Aligning。当Operator接收到多个输入的数据流时，需要在Snapshot Barrier中对数据流进行排列对齐，如下图5.7所示：

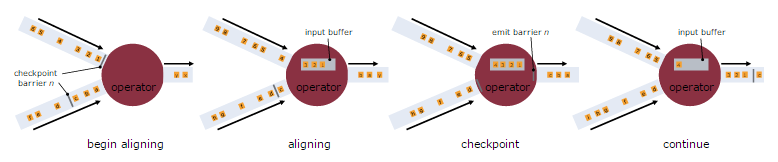


图5.7 数据流排列对齐图

具体排列过程如下：

1、Operator从一个incoming Stream接收到Snapshot Barrier n，然后暂停处理，直到其它的incoming Stream的Barrier n（否则属于2个Snapshot的记录就混在一起了）到达该Operator

2、接收到Barrier n的Stream被临时搁置，来自这些Stream的记录不会被处理，而是被放在一个Buffer中

3、一旦最后一个Stream接收到Barrier n，Operator会emit所有暂存在Buffer中的记录，然后向Checkpoint Coordinator发送Snapshot n

继续处理来自多个Stream的记录。

基于Stream Aligning操作能够实现Exactly Once语义，但是也会给流处理应用带来延迟，因为为了排列对齐Barrier，会暂时缓存一部分Stream的记录到Buffer中，尤其是在数据流并行度很高的场景下可能更加明显，通常以最迟对齐Barrier的一个Stream为处理Buffer中缓存记录的时刻点。在Flink中，提供了一个开关，选择是否使用Stream Aligning，如果关掉则Exactly Once会变成At least once。

### 5.3.2 调度机制

在JobManager端，会接收到Client提交的JobGraph形式的Flink Job，JobManager会将一个JobGraph转换映射为一个ExecutionGraph，如下图5.8所示：

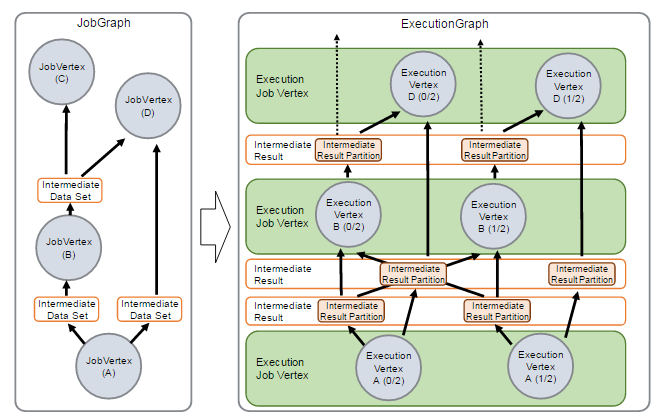


图5.8 调度机制示例

通过上图可以看出：

JobGraph是一个Job的用户逻辑视图表示，将一个用户要对数据流进行的处理表示为单个DAG图（对应于JobGraph），DAG图由顶点（JobVertex）和中间结果集（IntermediateDataSet）组成，其中JobVertex表示了对数据流进行的转换操作，比如map、flatMap、filter、keyBy等操作，而IntermediateDataSet是由上游的JobVertex所生成，同时作为下游的JobVertex的输入。

而ExecutionGraph是JobGraph的并行表示，也就是实际JobManager调度一个Job在TaskManager上运行的逻辑视图，它也是一个DAG图，是由ExecutionJobVertex、IntermediateResult（或IntermediateResultPartition）组成，ExecutionJobVertex实际对应于JobGraph图中的JobVertex，只不过在ExecutionJobVertex内部是一种并行表示，由多个并行的ExecutionVertex所组成。另外，这里还有一个重要的概念，就是Execution，它是一个ExecutionVertex的一次运行Attempt，也就是说，一个ExecutionVertex可能对应多个运行状态的Execution，比如，一个ExecutionVertex运行产生了一个失败的Execution，然后还会创建一个新的Execution来运行，这时就对应这个2次运行Attempt。每个Execution通过ExecutionAttemptID来唯一标识，在TaskManager和JobManager之间进行Task状态的交换都是通过ExecutionAttemptID来实现的。

### 5.3.3 迭代机制

机器学习和图计算应用，都会使用到迭代计算，Flink通过在迭代Operator中定义Step函数来实现迭代算法，这种迭代算法包括Iterate和Delta Iterate两种类型，在实现上它们反复地在当前迭代状态上调用Step函数，直到满足给定的条件才会停止迭代。下面，对Iterate和Delta Iterate两种类型的迭代算法原理进行说明：

#### 5.3.3.1 Iterate

Iterate Operator是一种简单的迭代形式：每一轮迭代，Step函数的输入或者是输入的整个数据集，或者是上一轮迭代的结果，通过该轮迭代计算出下一轮计算所需要的输入（也称为Next Partial Solution），满足迭代的终止条件后，会输出最终迭代结果，具体执行流程如下图5.9所示：

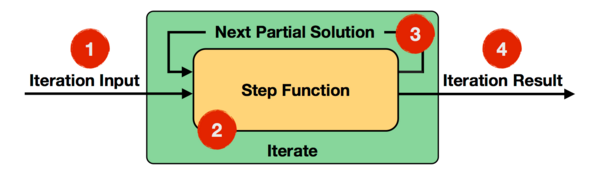


图5.9 Iterate Operator执行流程

Step函数在每一轮迭代中都会被执行，它可以是由map、reduce、join等Operator组成的数据流。下面通过官网给出的一个例子来说明Iterate Operator，非常简单直观，如下图所示：

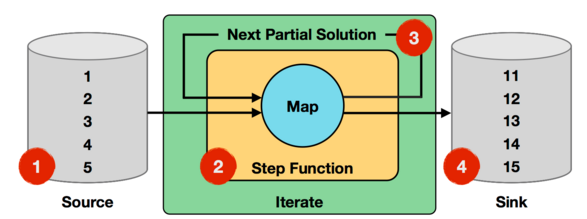


图5.10 Step函数运行示例图

上面迭代过程中，输入数据为1到5的数字，Step函数就是一个简单的map函数，会对每个输入的数字进行加1处理，而Next Partial Solution对应于经过map函数处理后的结果，比如第一轮迭代，对输入的数字1加1后结果为2，对输入的数字2加1后结果为3，直到对输入数字5加1后结果为变为6，这些新生成结果数字2~6会作为第二轮迭代的输入。迭代终止条件为进行10轮迭代，则最终的结果为11~15。

#### 5.3.3.2 Delta Iterate

Delta Iterate Operator实现了增量迭代，它的实现原理如下图所示：

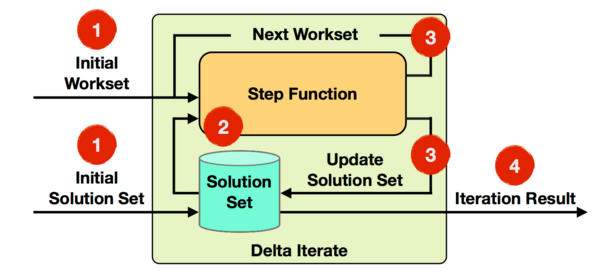


图5.11 Delta Iterate Operator原理图

基于Delta Iterate Operator实现增量迭代，它有2个输入，其中一个是初始Workset，表示输入待处理的增量Stream数据，另一个是初始Solution Set，它是经过Stream方向上Operator处理过的结果。第一轮迭代会将Step函数作用在初始Workset上，得到的计算结果Workset作为下一轮迭代的输入，同时还要增量更新初始Solution Set。如果反复迭代知道满足迭代终止条件，最后会根据Solution Set的结果，输出最终迭代结果。

比如，我们现在已知一个Solution集合中保存的是，已有的商品分类大类中购买量最多的商品，而Workset输入的是来自线上实时交易中最新达成购买的商品的人数，经过计算会生成新的商品分类大类中商品购买量最多的结果，如果某些大类中商品购买量突然增长，它需要更新Solution Set中的结果（原来购买量最多的商品，经过增量迭代计算，可能已经不是最多），最后会输出最终商品分类大类中购买量最多的商品结果集合。更详细的例子，可以参考官网给出的“Propagate Minimum in Graph”,这里不再累述。

## 5.4 技术特点

Flink 是一个开源的分布式，高性能，高可用，准确的流处理框架。支持实时流处理和批处理。

### 5.4.1 流处理特性

支持高吞吐、低延迟、高性能的流处理

支持带有事件时间的窗口（Window）操作

支持有状态计算的Exactly-once语义

支持高度灵活的窗口（Window）操作，支持基于time、count、session，以及data-driven的窗口操作

支持具有Backpressure功能的持续流模型

支持基于轻量级分布式快照（Snapshot）实现的容错

一个运行时同时支持Batch on Streaming处理和Streaming处理

Flink在JVM内部实现了自己的内存管理

支持迭代计算

支持程序自动优化：避免特定情况下Shuffle、排序等昂贵操作，中间结果有必要进行缓存

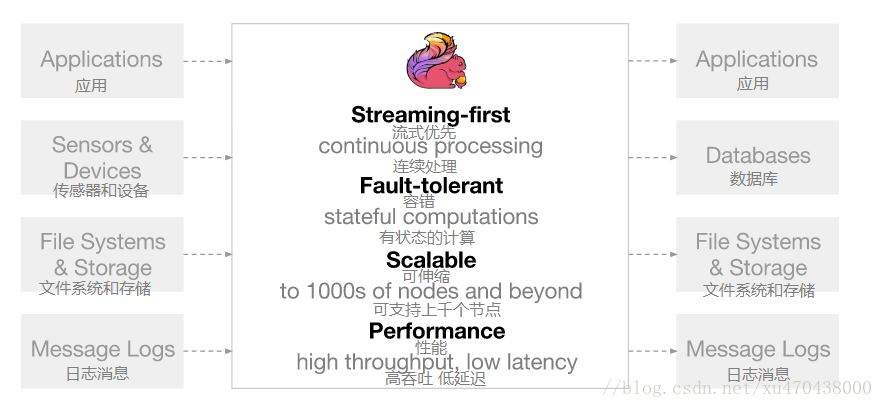


图5.12 流处理特性

### 5.4.2 API支持

对Streaming数据类应用，提供DataStream API。

对批处理类应用，提供DataSet API（支持Java/Scala）。

### 5.4.3 Libraries支持

支持机器学习（FlinkML）。

支持图分析（Gelly）。

支持关系数据处理（Table）。

支持复杂事件处理（CEP）。

### 5.4.4 整合支持

支持Flink on YARN

支持HDFS

支持来自Kafka的输入数据

支持Apache HBase

支持Hadoop程序

支持Tachyon

支持ElasticSearch

支持RabbitMQ

支持Apache Storm

支持S3

支持XtreemFS

### 5.4.5 Flink生态圈

一个计算框架要有长远的发展，必须打造一个完整的 Stack。不然就跟纸上谈兵一样，没有任何意义。只有上层有了具体的应用，并能很好的发挥计算框架本身的优势，那么这个计算框架才能吸引更多的资源，才会更快的进步。所以 Flink 也在努力构建自己的 Stack。

Flink 首先支持了 Scala 和 Java 的 API，Python 也正在测试中。Flink 通过 Gelly 支持了图操作，还有机器学习的FlinkML。Table 是一种接口化的 SQL 支持，也就是 API 支持，而不是文本化的 SQL 解析和执行。对于完整的 Stack我们可以参考下图。

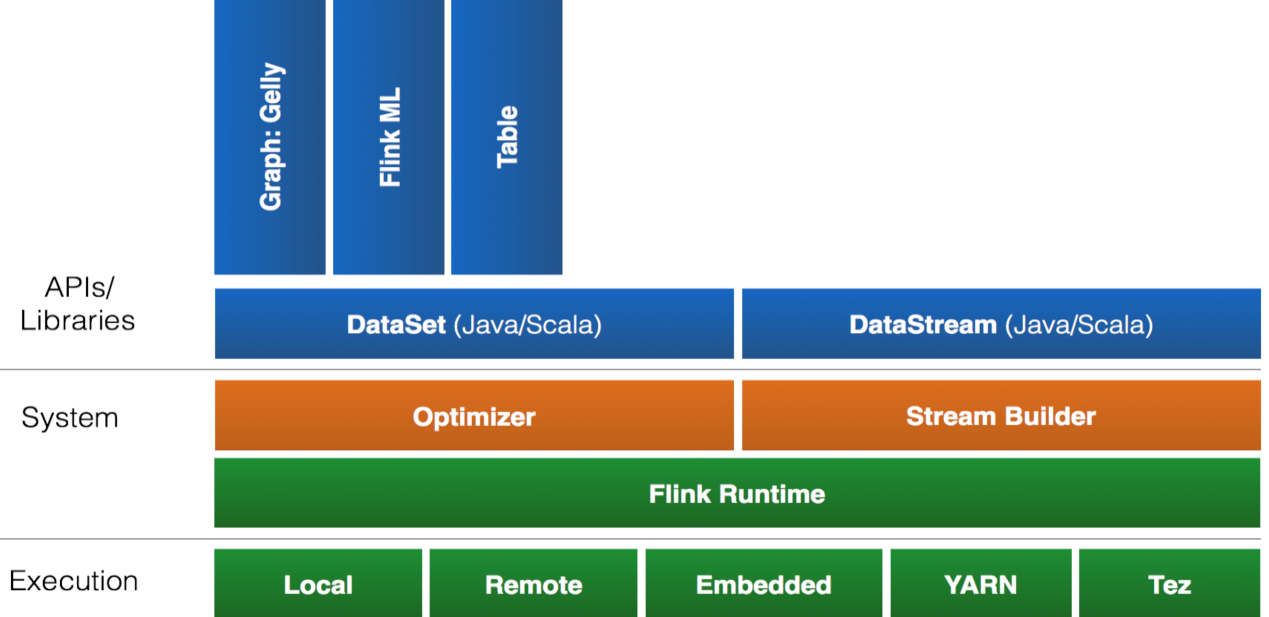


图5.13 Flink完整Stack层次图

Flink 为了更广泛的支持大数据的生态圈，其下也实现了很多 Connector 的子项目。最熟悉的，当然就是与Hadoop HDFS 集成。其次，Flink 也宣布支持了 Tachyon、S3 以及 MapRFS。不过对于 Tachyon 以及 S3 的支持，都是通过 Hadoop HDFS 这层包装实现的，也就是说要使用 Tachyon 和 S3，就必须有 Hadoop，而且要更改 Hadoop的配置（core-site.xml）。如果浏览 Flink 的代码目录，我们就会看到更多 Connector 项目，例如 Flume 和 Kafka。

## 5.5 Flink的应用场景

Flink最适合的应用场景是低延时的数据处理场景：高并发处理数据，实验毫秒级，且兼具可靠性。

Flink的典型应用场景有：

1、优化电子商务的实时搜索结果：阿里巴巴的所有基础设施团队使用flink实时更新产品细节和库存信息，为用户提供更高的关联性。

2、针对数据分析团队提供实时流处理服务：king通过flink-powered数据分析平台提供实时数据分析，从游戏数据中大幅缩短了观察时间

3、网络/传感器检测和错误检测：Bouygues电信公司，是法国最大的电信供应商之一，使用flin监控其有线和无线网络，实现快速故障响应。

4、商业智能分析ETL：Zalando使用flink转换数据以便于加载到数据仓库，将复杂的转换操作转化为相对简单的并确保分析终端用户可以更快的访问数据。

基于上面的案例分析，Flink非常适合于：

多种数据源(有时不可靠)：当数据是由数以百万计的不同用户或设备产生的，它是安全的假设数据会按照事件产生的顺序到达，和在上游数据失败的情况下，一些事件可能会比他们晚几个小时，迟到的数据也需要计算，这样的结果是准确的。

应用程序状态管理：当程序变得更加的复杂，比简单的过滤或者增强的数据结构，这个时候管理这些应用的状态将会变得比较难(例如：计数器，过去数据的窗口，状态机，内置数据库)。flink提供了工具，这些状态是有效的，容错的，和可控的，所以你不需要自己构建这些功能。

数据的快速处理：有一个焦点在实时或近实时用例场景中，从数据生成的那个时刻，数据就应该是可达的。在必要的时候，flink完全有能力满足这些延迟。

海量数据处理：这些程序需要分布在很多节点运行来支持所需的规模。flink可以在大型的集群中无缝运行，就像是在一个小集群一样。

# 6．大数据案例应用

目前我们正处于大数据时代，每天都会产生和处理海量数据。现在很多企业都已经构建了自己的大数据平台，本文主要以腾讯和美团公司的大数据平台为例，来展示大数据平台的一些应用场景。

## 6.1 腾讯大数据平台

据腾讯季度报告，显示腾讯QQ月活跃账户数达到8.03亿，QQ智能终端月活跃账户数达到7.09亿；微信和WeChat的合并月活跃账户数达到10.58亿；QQ空间月活跃账户数达到5.48亿；QQ空间智能终端月活跃账户数达到5.43亿…… 面对如此多的数据，腾讯的数据分析已经能做到始终“不落地”，即全部的实时处理。这主要是依靠腾讯的大数据平台。

腾讯大数据平台有如下核心模块：TDW、TRC、TDBank、TPR和Gaia。简单来 说，TDW用来做批量的离线计算，TRC负责做流式的实时计算，TPR负责精准推荐，TDBank则作为统一的数据采集入口，而底层的Gaia则负责整个集群的资源调度和管理。

腾讯的技术架构图如图6.1所示：

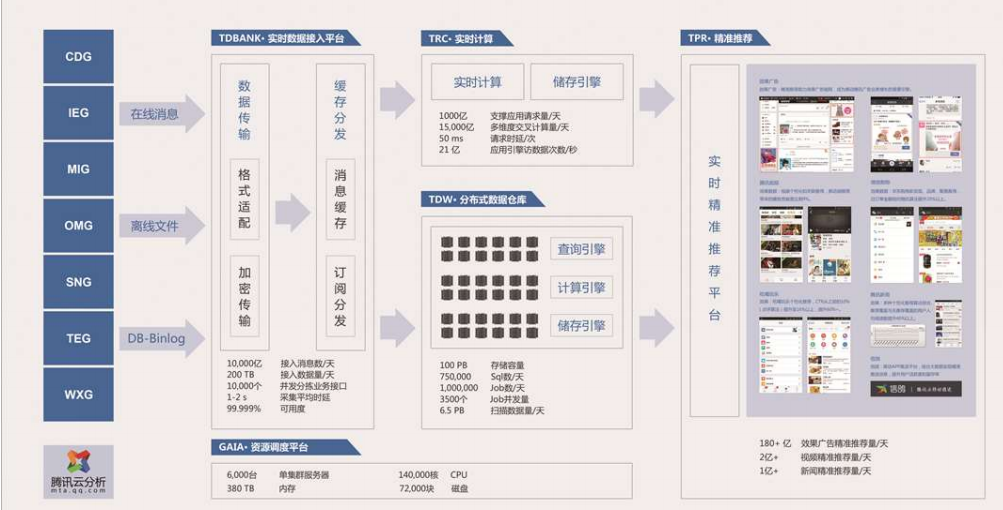


图6.1 技术架构图

腾讯在现在把公司的组织架构分成七大事业群（如图6.1左边所示）,其实有点类似于七个相对独立的公司一样，只不过事业群之间还是 共享很多资源，也有很多关联；

七个事业群分别是:

1、CDG企业发展事业群（有财付通部门，负责战略投资的部门，现在腾讯投资了很多公司，比如京东、搜狗、大众点评、58同城、滴滴打车、金山）

2、IEG互动娱乐事业群（腾讯游戏，文学，动漫）

3、MIG移动互联网事业群（qq浏览器、qq管家、腾讯地图、应用宝）

4、OMG网络媒体事业群（腾讯网、腾讯视频、腾讯微博、微视）

5、SNG社交网络事业群（QQ聊天工具,广点通、腾讯云、QQ音乐、QQ空间）

6、TEG技术工程事业群（没有直接的商业产品，是其他事业群的支撑平台）

7、WXG微信事业群（微信、qq邮箱、foxmail; 这是新发展起来相当有前景的部门）

整个平台会进行数据积累（包括数据的介入和存储）、数据处理（离线计算和实时计算）以及数据应用（查询分析、数据挖掘等）。腾讯大数据平台的整体规划主要由下图6.2所示：



图6.2 腾讯大数据平台整体规划

腾讯大数据平台最重要的是Gaia。Gaia以后可以承载各种编程框架、各种应用，是个统一的资源管理调度系统，各种业务都植根于“大地”之上。Gaia是基于Yarn的通用资源调度平台，提供高并发任务调度和资源管理，实现集群资源共享，可伸缩性和可靠性，不仅可以为MR等离线业务提供服务，还可以支持实时计算，甚至在线service业务。

Gaia的技术特点：强扩展性、高调度吞吐、弹性内存管理、多维度资源管理、丰富的用户api 、建立“on Gaia”生态圈等。而腾讯的TDW支持百PB级的数据存储和计算，提供海量、高效稳定的大数据平台支撑和决策支持；具体的应用场景主要由四个方面：社交，根据人们发的动态，可以实时了解到大家的动态和喜好，推荐相关的文章或者商品；电商，实时统计商品热度，了解人们喜好哪种商品，实施精准营销；游戏，实时预测用户感受，可以对游戏进行优化；营销，实时感知用户兴趣变化、环境变化等，实现精准营销。

## 6.2 美团点评大数据平台

美团点评大数据平台如下图6.3所示：



图6.3 美团点评大数据平台

美团点评大数据平台的架构，最底层是美团的基础设施，美团有自己的云平台，所以所有的服务器都在云平台上。云平台之上是基础的 HDFS 这样的服务，像 Hbase 这些相关的基础设施。上面一层是机器学习平台，非线性数目性的工具都有相应的部署。最上面是美团点评几个大的业务群，包括酒店、旅游的业务，包括娱乐这样综合的业务，包括整个公司的广告平台。在主架构之上有一个 Log 平台，因为业务很多，所以业务产生的数据，不管是 B 端还是 C 端，很多数据都会统一落到 Log 平台上，由 Log 平台对接到顶层业务平台上。另外一个就是监控平台，服务监控、数据监控以及业务调度。

美团外卖提供的服务一个是个性化程度很高，另外一块精细化的需求很大。在时间场景上、地域场景上，甚至更细的业务场景上，要把精细化程度做深；另外一块是多元化的特点，形态多元、内容多元包括供给单位的多元;还有实时化程度很重要，配送这一块的距离在实时变化，需要做实时监控。美团的大数据平台保障这些的实现。下面介绍美团点评深度学习的两个关键应用场景。

### 6.2.1 图像技术

美团平台上的技术内容多样化，包括图像和文字，尤其用户在手机上面，会做很多决策的时候，图像对他的决策影响至关重要。美团做了两个关键事情，一个是图像质量的技术，另一个是 OCR 技术。



图6.4 示例

图像质量技术有什么应用场景？如上图6.4所示，左侧这个图明显看着比较舒服，右侧图就看着乱七八糟，这个很难主观描述，就是人的直观感觉。具体来说，左侧两个图片的清晰度可能高一点，或者完整度高一些。在用户做决策的时候，如果在平台上面分的是质量比较高的图片，用户整个点击会比较快。对商家来讲，用户很容易发生需求匹配的转化。但是右侧这个图来讲，整个体验转化的能力上会相对比较弱。基于这个背景，对图像质量进行判断，发现存在一些关键的难点和挑战，主要有三个关键点：标准评判很难，包括定义它的清晰度、完整度还有图片里面的构成，整个行业也没有对图像质量的标准定义。用户的关注点不一样，有些用户可能关注图片是否清晰，有些用户关注图片是不是完整，有没有关键的信息给切走。而人工标注成本高。

为了解决这个问题，美团引入了建模的方式，一个像美学的课程，另一个是图像基础的特征，并在样本标注上采用自动化标注的方式。

我们去看图片的来源，直观的理解是用户自动上传的图片，也就是 UGC 的图片，质量相对差一些。我们专业的人员开设的图片质量会好一些，可以做一些基本的标注。另外一块是基于用户行为来看，我们会展示给用户点击比较多的图片，可能是用户感兴趣或者质量比较好的图片；点击比较少的图片，我们认为它是质量比较差的图片，这样可以通过自动化的方式把样本标注出来。在图片输入上，没有直接给每个图片识别它是高质量、中质量还是低质量的图片，基本上采用的是一个学习的方式，比如图片的美感，通过 DNN 方式来判断图片，包括它的品类，属于快餐、西餐还是简餐之类。另外一块是基础特征，图像的色彩，包括图像的对比度。这些基础特征也可以用来做整个建模，还有一块是整个在深度学习上面，通过 KOS 的方式判断哪一个图像质量比较好，哪一个图像质量比较差，最主要是从一堆图片里面选择比较好的图片展示给用户。涉及到的数据比较商业化，整体来讲，建模做完了之后，用户体验式得到了提升。

### 6.2.2 OCR技术

OCR 是做数据的检测，OCR 在美团外卖有哪些比较关键的应用呢? 比较大的方面是自动审核，为了保证给用户提供可靠的服务或者优质的服务，我们需要审核商家的牌照，商家的经营许可证，商家的产品质量，这一块如果用人工审核的话，人力成本会很高。另外一块是自动录入，商家入驻到外卖平台之后，要把菜录入到上面去，尤其商家要不断的改菜单，不断的录入菜单，这一块是自动录入的过程，这是我们基本应用上的两个需求点。图像有很多成像的方式，另外一块文字很复杂，很多图片上的文字不是标准的楷体，各种各样乱七八糟的字体会有，甚至还有变形的字体，图像的背景很复杂，或者说人看到都很难区分这种场景，这对整个 OCR 提出了技术难点的挑战。

文字的检测首先从图片开始，比如我们检测哪一些是图片里面的文字，哪一些是图片本身。另外一块是文字的识别，检测到图片之后，怎么把里面的文字识别出来，这一块整体采用的是 CNN 方式，加入 BLSTM 的模式，考虑了整个序列建模的能力，最后一块是 CTC 的序列识别模型。美团整个过程由原来人工审核变成了自动化的过程，整个运营效率得到了很大的提升。

# 7．结束与总结

大数据平台架构领域作为目前最热门的研究方向之一，吸引了越来越多学术界和工业界人士对其进行不断地研究与发展。本文详述了大数据平台架构当前的研究现状和发展趋势，介绍了大数据架构中hadoop、spark、flink以及具体的场景应用等几大模块。首先本文介绍了这些架构方法可以成功解决众多具有挑战性的问题，比如海量数据信息存储，海量数据计与分析等。然后介绍了几个主要的大数据平台架构，分别是：（1）HADOOP的核心组件相关技术与应用， HDFS（分布式文件系统）用于存储海量数据文件、YARN（运算资源调度系统）负责为运算程序提供服务器运算资源、MAPREDUCE（分布式运算编程框架）将一个存储在分布式文件系统中的大规模数据集进行分片计算归总；（2）Spark作为一种快速、通用、可扩展的大数据分析引擎，提供了一个全面、统一的框架用于管理各种有着不同性质（文本数据、图表数据等）的数据集和数据源（批量数据或实时的流数据）的大数据处理的需求；（3）Flink是一个针对流数据和批数据的分布式处理引擎，其所要处理的主要场景就是流数据，同时可以支持本地的快速迭代，以及一些环形的迭代任务。并且Flink可以定制化内存管理；最后，本文介绍了大数据平台架构技术几个重要的实际应用。

综上所述，大数据平台架构有着很重要的实际应用场景，这种技术的创新将会对互联网上产生的所有数据进行有效的利用，从而可以在海量的、无序的信息中快速提炼出有针对性的有效信息，使数据利益最大化，有很好的应用场景。

**参考文献**

1. 王永坤, 罗萱, 金耀辉. 基于私有云和物理机的混合型大数据平台设计及实现[J]. 计算机工程与科学, 2018.
2. Taylor R C. An overview of the Hadoop/MapReduce/HBase framework and its current applications in bioinformatics[J]. Bmc Bioinformatics, 2010, 11 Suppl 12(S12):S1.
3. Verma J P , Patel B , Patel A . Big data analysis: Recommendation system with hadoop framework[C]// IEEE International Conference on Computational Intelligence & Communication Technology. IEEE, 2015.
4. Verbitskiy I, Thamsen L, Kao O. When to Use a Distributed Dataflow Engine: Evaluating the Performance of Apache Flink[C]// Ubiquitous Intelligence & Computing, Advanced & Trusted Computing, Scalable Computing & Communications, Cloud & Big Data Computing, Internet of People, & Smart World Congress. 2017.
5. 韩健. 基于Hadoop技术的银行大数据平台架构设计[J]. 电子世界, 2017(22):164-165.
6. Li Z, Shen H, Ligon W, et al. An Exploration of Designing a Hybrid Scale-Up/Out Hadoop Architecture Based on Performance Measurements[J]. IEEE Transactions on Parallel & Distributed Systems, 2017, 28(2):386-400.
7. 冯兴杰, 王文超. Hadoop与Spark应用场景研究[J]. 计算机应用研究, 2018(9).
8. Khan M, Jin Y, Li M, et al. Hadoop Performance Modeling for Job Estimation and Resource Provisioning[J]. IEEE Transactions on Parallel & Distributed Systems, 2016, 27(2):441-454.
9. 刘备, 谭新明, 曹文彬. Spark Streaming动态资源分配策略[J]. 计算机应用, 2017(6).
10. Samadi Y, Zbakh M, Tadonki C. Performance comparison between Hadoop and Spark frameworks using HiBench benchmarks[J]. Concurrency & Computation Practice & Experience, 2017(2).
11. Edwards M, Rambani A, Zhu Y, et al. Design of Hadoop-based Framework for Analytics of Large Synchrophasor Datasets[J]. Procedia Computer Science, 2012, 12(4):254-258.
12. 姜文. 基于Hadoop平台的数据分析和应用[D]. 北京邮电大学, 2011.
13. Grace R K, Manimegalai R, Kumar S S. Medical Image Retrieval System in Grid Using Hadoop Framework[C]// International Conference on Computational Science & Computational Intelligence. 2014.
14. Verma A, Mansuri A H, Jain N. Big data management processing with Hadoop MapReduce and spark technology: A comparison[C]// Colossal Data Analysis & Networking. 2016.
15. Mavridis I, Karatza E. Performance evaluation of cloud-based log file analysis with Apache Hadoop and Apache Spark[J]. Journal of Systems & Software, 2016, 125:S0164121216302370.
16. 赵乐乐, 黄刚, 马越. 基于Docker的Hadoop平台架构研究[J]. 计算机技术与发展, 2016, 26(9):99-103.
17. Sun Z, Hong Z, Liu Z, et al. Migrating GIS Big Data Computing from Hadoop to Spark: An Exemplary Study Using Twitter[C]// IEEE International Conference on Cloud Computing. 2017.
18. Marcu O C, Costan A, Antoniu G, et al. Spark Versus Flink: Understanding Performance in Big Data Analytics Frameworks[C]// IEEE International Conference on Cluster Computing. 2016.
19. Falkenthal M, Leymann F, Kempa-Liehr A W, et al. OpenTOSCA for the 4th Industrial Revolution: Automating the Provisioning of Analytics Tools based on Apache Flink[C]// International Conference on the Internet of Things. 2016.
20. Garcíagil D, Ramírezgallego S, García S, et al. A comparison on scalability for batch big data processing on Apache Spark and Apache Flink[J]. Big Data Analytics, 2017, 2(1):1.