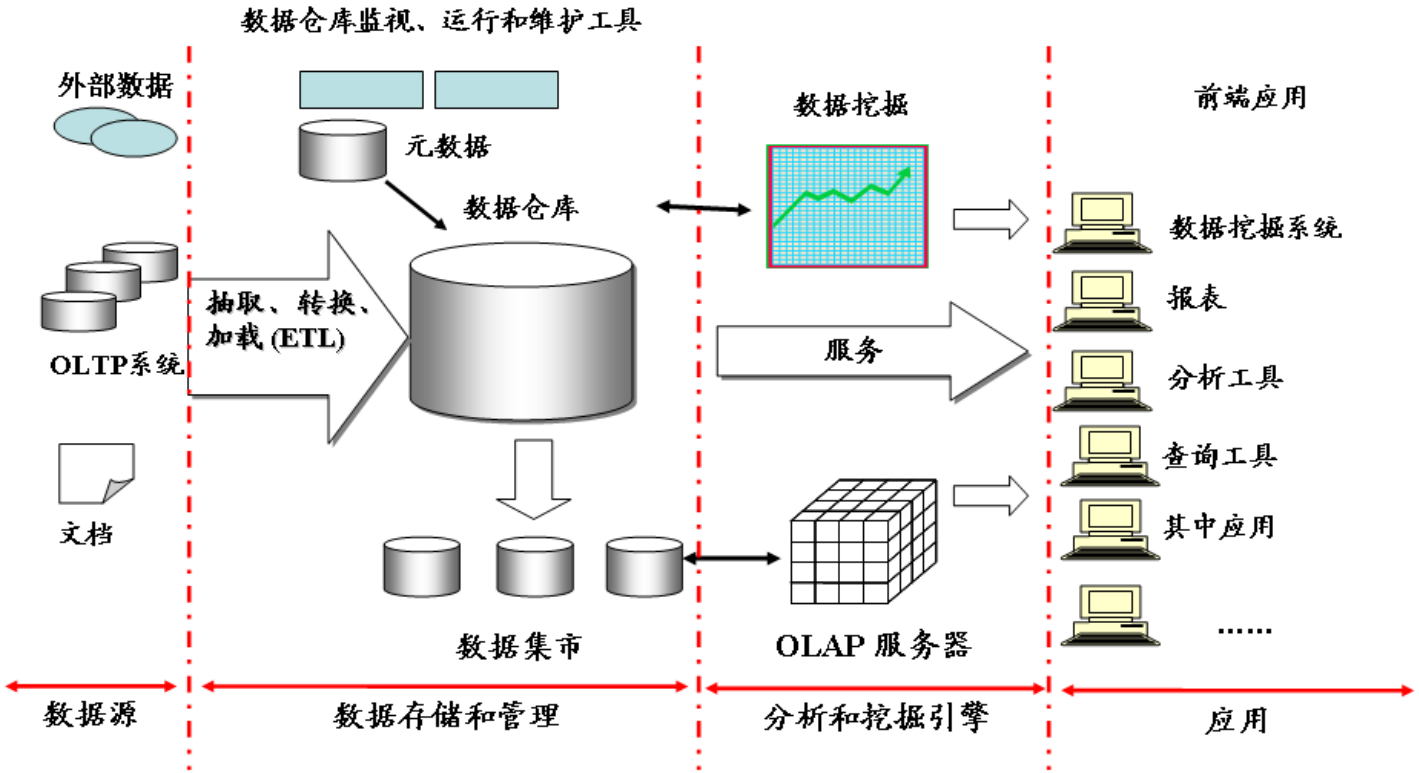
10

Spark Streaming

- 静态数据与流数据
- 流计算
- Spark Streaming

静态数据和流数据

很多企业为了支持决策分析而构建的数据仓库系统，其中存放的大量历史数据就是静态数据。技术人员可以利用数据挖掘和OLAP（On-Line Analytical Processing）分析工具从静态数据中找到对企业有价值的信息



静态数据和流数据

近年来，在Web应用、网络监控、传感监测等领域，兴起了一种新的数据密集型应用——流数据，即数据以大量、快速、时变的流形式持续到达

实例：PM2.5检测、电子商务网站用户点击流

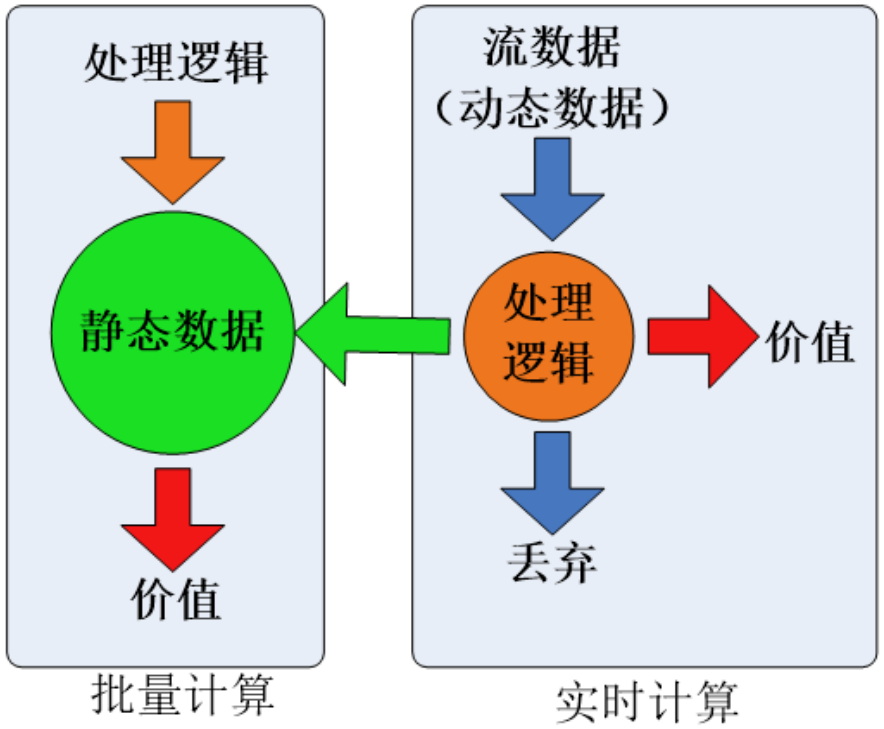
流数据具有如下特征：

- 数据快速持续到达，潜在大小也许是无穷无尽的
- 数据来源众多，格式复杂
- 数据量大，但是不十分关注存储，一旦经过处理，要么被丢弃，要么被归档存储
- 注重数据的整体价值，不过分关注个别数据
- 数据顺序颠倒，或者不完整，系统无法控制将要处理的新到达的数据元素的顺序

批量计算和实时计算

对静态数据和流数据的处理，对应着两种截然不同的计算模式：批量计算和实时计算

- 批量计算：充裕时间处理静态数据，如Hadoop
- 流数据不适合采用批量计算，因为流数据不适合用传统的关系模型建模
- 流数据必须采用实时计算，响应时间为秒级
- 数据量少时，不是问题，但是，在大数据时代，数据格式复杂、来源众多、数据量巨大，对实时计算提出了很大的挑战。因此，针对流数据的实时计算——流计算，应运而生



流计算概念

流计算：实时获取来自不同数据源的海量数据，经过实时分析处理，获得有价值的信息

流计算概念

流计算秉承一个基本理念，即**数据的价值随着时间的流逝而降低，**如用户点击流。因此，当事件出现时就应该立即进行处理，而不是缓存起来进行批量处理。为了及时处理流数据，就需要一个低延迟、可扩展、高可靠的处理引擎

对于一个流计算系统来说，它应达到如下需求：

- 高性能：处理大数据的基本要求，如每秒处理几十万条数据
- 海量式：支持TB级甚至是PB级的数据规模
- 实时性：保证较低的延迟时间，达到秒级别，甚至是毫秒级别
- 分布式：支持大数据的基本架构，必须能够平滑扩展
- 易用性：能够快速进行开发和部署
- 可靠性：能可靠地处理流数据

流计算框架

- 当前业界诞生了许多专门的流数据实时计算系统来满足各自需求
- 目前有三类常见的流计算框架和平台：商业级的流计算平台、开源流计算框架、公司为支持自身业务开发的流计算框架
- 商业级：IBM InfoSphere Streams和IBM StreamBase
- 较为常见的是开源流计算框架，代表如下：
  - Twitter Storm：免费、开源的分布式实时计算系统，可简单、高效、可靠地处理大量的流数据
  - Yahoo! S4（Simple Scalable Streaming System）：开源流计算平台，是通用的、分布式的、可扩展的、分区容错的、可插拔的流式系统
- 公司为支持自身业务开发的流计算框架：
  - Facebook Puma
  - Dstream（百度）
  - 银河流数据处理平台（淘宝）

流计算处理流程

传统的数据处理流程，需要先采集数据并存储在关系数据库等数据管理系统中，之后由用户通过查询操作和数据管理系统进行交互

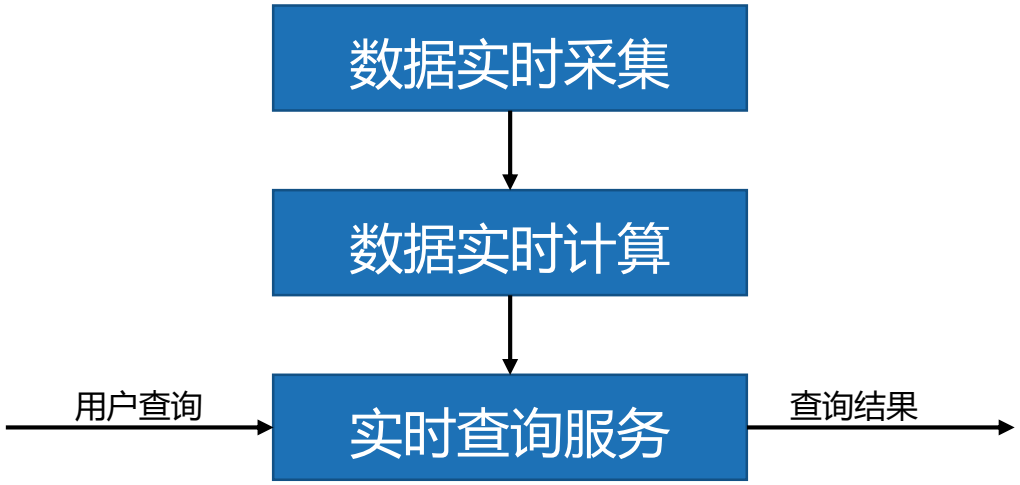
传统的数据处理流程示意图

传统的数据处理流程隐含了两个前提：

- **存储的数据是旧的。**存储的静态数据是过去某一时刻的快照，这些数据在查询时可能已不具备时效性了
- **需要用户主动发出查询来获取结果**

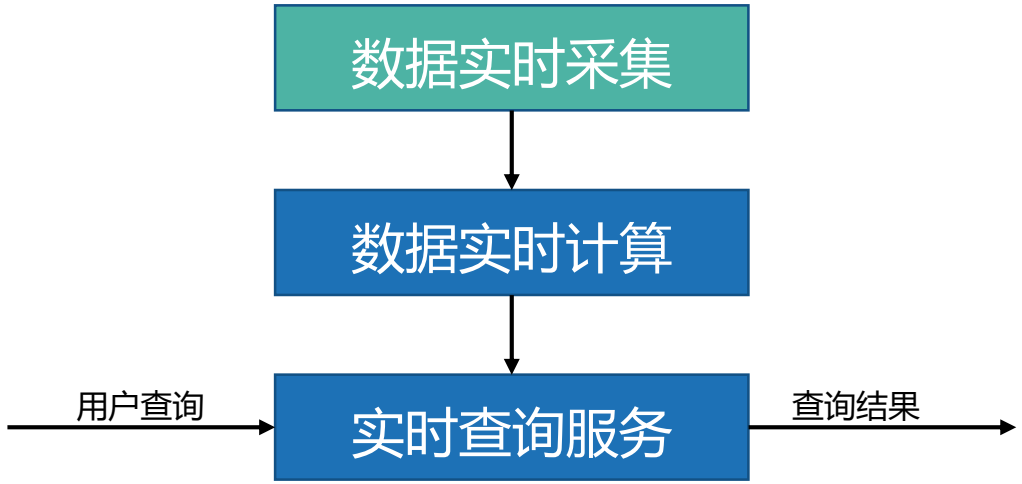
流计算处理流程

流计算的处理流程一般包含三个阶段：数据实时采集、数据实时计算、实时查询服务



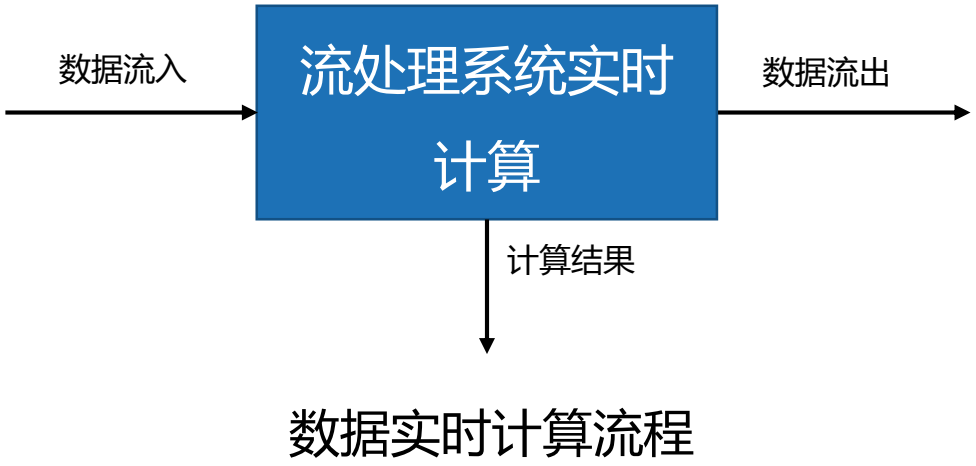
数据实时采集

- 数据实时采集阶段通常采集多个数据源的海量数据，需要保证实时性、低延迟与稳定可靠
- 以日志数据为例，由于分布式集群的广泛应用，数据分散存储在不同的机器上，因此需要实时汇总来自不同机器上的日志数据
- 目前有许多互联网公司发布的开源分布式日志采集系统均可满足每秒数百MB的数据采集和传输需求，如：
  - Facebook的Scribe
  - LinkedIn的Kafka
  - 淘宝的Time Tunnel
  - 基于Hadoop的Chukwa和Flume

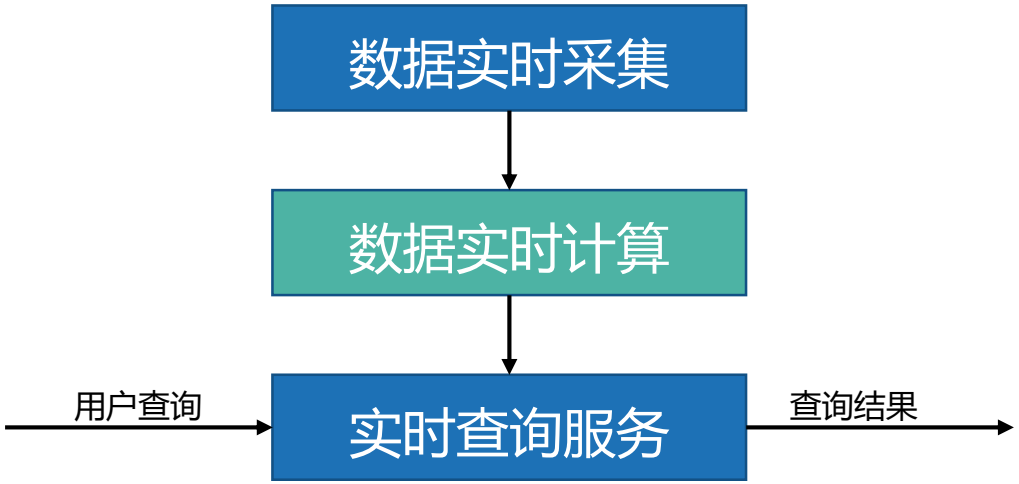


数据实时计算

- 数据实时计算阶段对采集的数据进行实时的分析和计算，并反馈实时结果
- 经流处理系统处理后的数据，可视情况进行存储，以便之后再进行分析计算。在时效性要求较高的场景中，处理之后的数据也可以直接丢弃

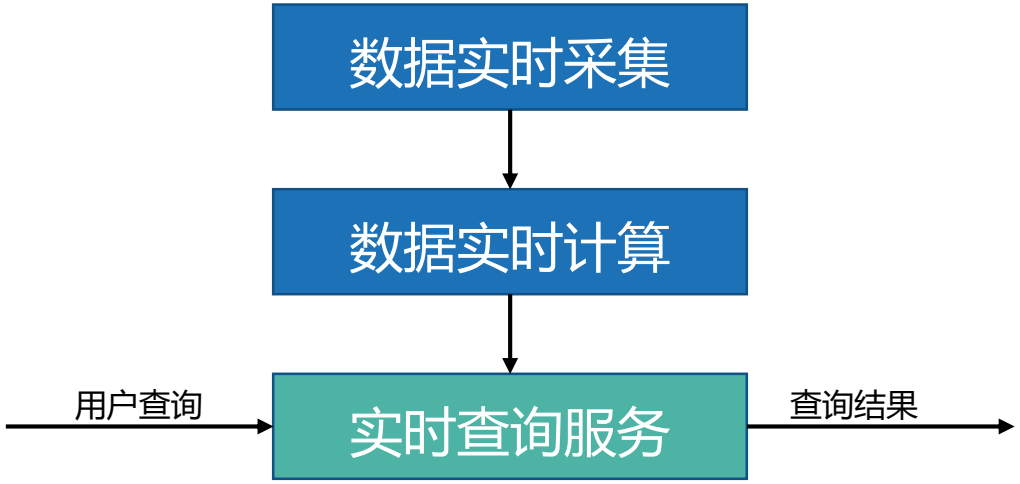


数据实时计算流程



实时查询服务

- 实时查询服务：经由流计算框架得出的结果可供用户进行实时查询、展示或储存
- 传统的数据处理流程，用户需要主动发出查询才能获得想要的结果。而在流处理流程中，实时查询服务可以不断更新结果，并将用户所需的结果实时推送给用户
- 虽然通过对传统的数据处理系统进行**定时**查询，也可以实现不断地更新结果和结果推送，但通过这样的方式获取的结果，仍然是根据过去某一时刻的数据得到的结果，与实时结果有着本质的区别



流计算处理流程

流处理系统与传统的数据处理系统有如下不同：

- 流处理系统处理的是实时的数据，而传统的数据处理系统处理的是预先存储好的静态数据
- 用户通过流处理系统获取的是实时结果，而通过传统的数据处理系统，获取的是过去某一时刻的结果
- 流处理系统无需用户主动发出查询，实时查询服务可以主动将实时结果推送给用户

Spark Streaming设计

Spark Streaming可整合多种输入数据源，如Kafka、Flume、HDFS，甚至是普通的TCP套接字。经处理后的数据可存储至文件系统、数据库，或显示在仪表盘里

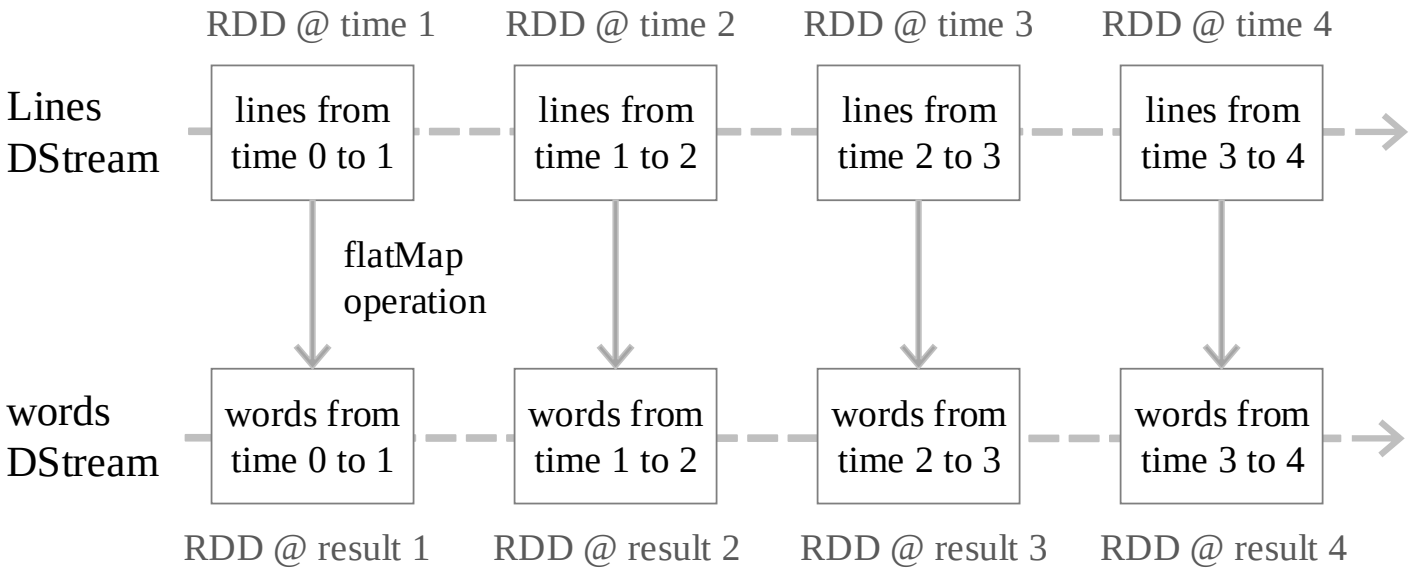
Spark Streaming支持的输入、输出数据源

Spark Streaming设计

Spark Streaming的基本原理是将实时输入数据流以时间片（秒级）为单位进行拆分，然后经Spark引擎以类似批处理的方式处理每个时间片数据

Spark Streaming设计

Spark Streaming最主要的抽象是DStream（Discretized Stream，离散化数据流），表示连续不断的数据流。在内部实现上，Spark Streaming的输入数据按照时间片（如1秒）分成一段一段，每一段数据转换为Spark中的RDD，这些分段就是Dstream，并且对DStream的操作都最终转变为对相应的RDD的操作

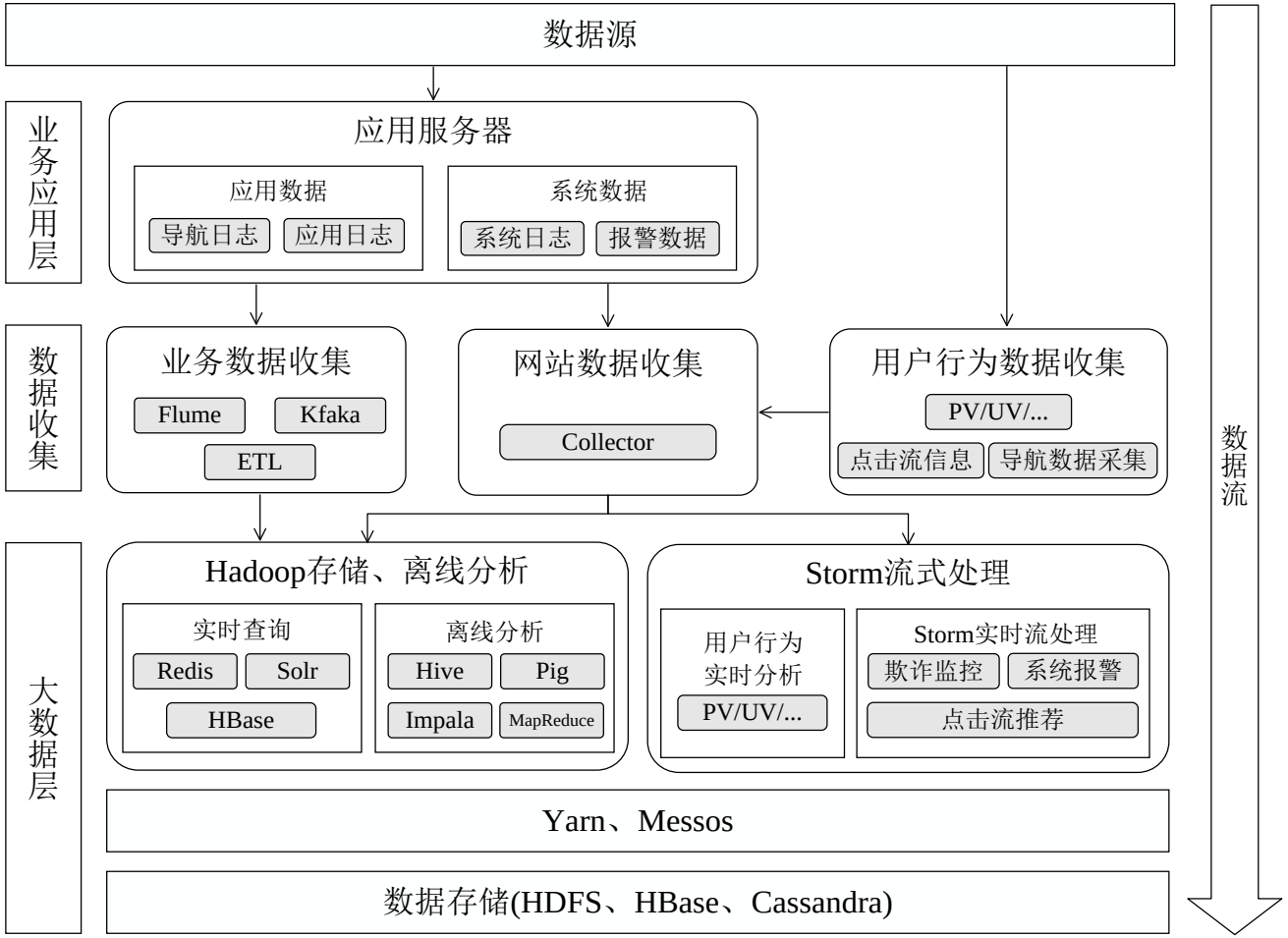


DStream操作示意图

Spark Streaming与Storm的对比

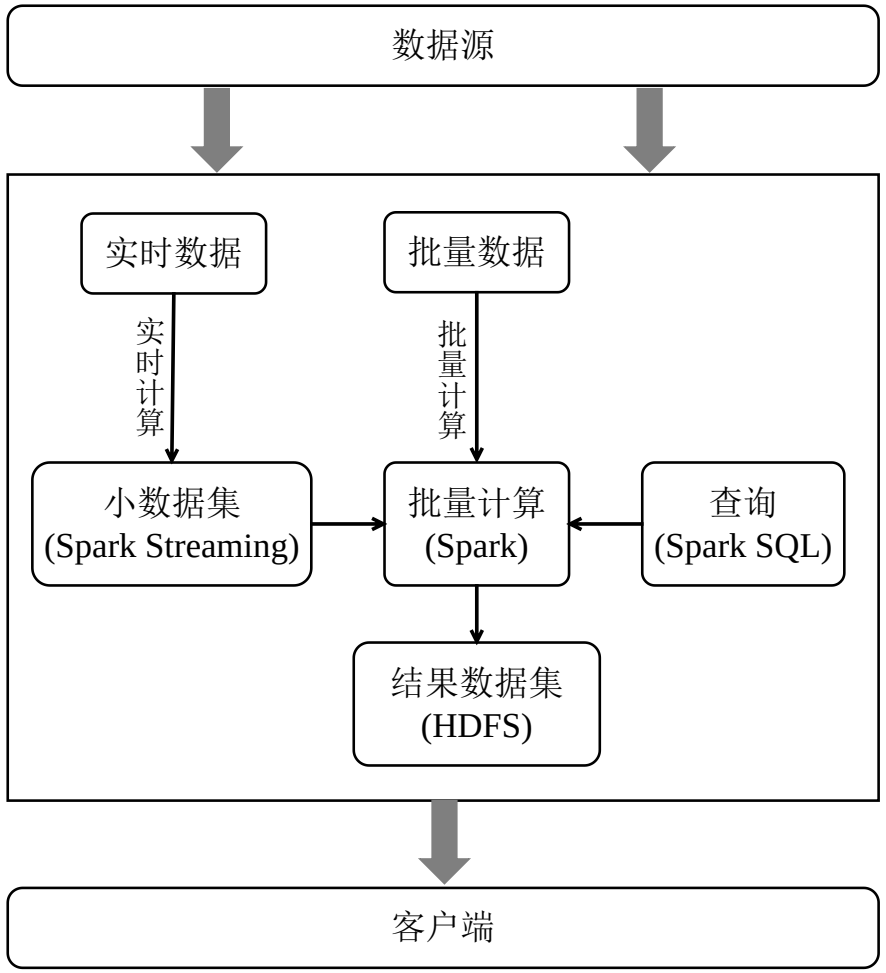
- Spark Streaming和Storm最大的区别在于，Spark Streaming无法实现毫秒级的流计算，而Storm可以实现毫秒级响应
- Spark Streaming构建在Spark上，一方面是因为Spark的低延迟执行引擎（100ms+）可以用于实时计算，另一方面，相比于Storm，RDD数据集更容易做高效的容错处理
- Spark Streaming采用的小批量处理的方式使得它可以同时兼容批量和实时数据处理的逻辑和算法，因此，方便了一些需要历史数据和实时数据联合分析的特定应用场合

从“Hadoop+Storm”架构转向Spark架构



采用Hadoop+Storm部署方式的一个案例

从“Hadoop+Storm”架构转向Spark架构

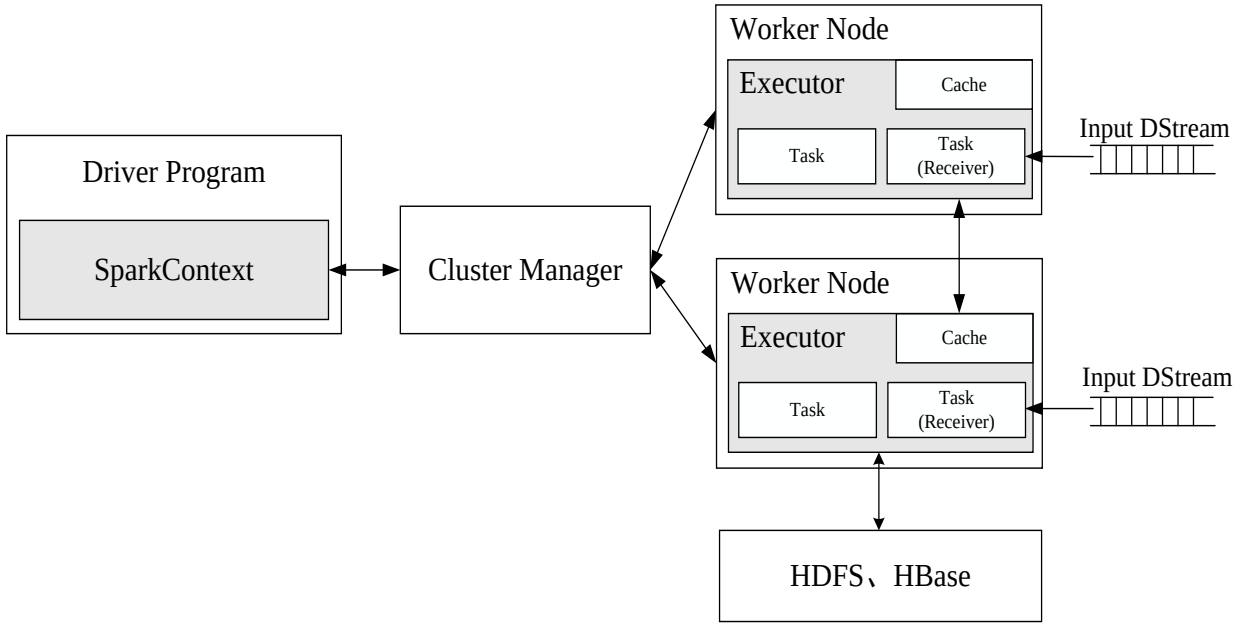


用Spark架构满足批处理和流处理需求

采用Spark架构具有如下优点：

- 实现一键式安装和配置、线程级别的任务监控和告警；
- 降低硬件集群构建、软件维护、任务监控和应用开发的难度；
- 便于做成统一的硬件、计算平台资源池。

Spark Streaming工作机制



- 在Spark Streaming中，有一个组件Receiver，作为一个长期运行的task跑在一个
- Executor上
- 每个Receiver都会负责一个input DStream（比如从文件中读取数据的文件流，比如套接字流，或者从Kafka中读取的一个输入流等等）
- Spark Streaming通过input DStream与外部数据源进行连接，读取相关数据

Spark Streaming程序的基本步骤

编写Spark Streaming程序的基本步骤是：

1. 通过创建输入DStream来定义输入源
2. 通过对DStream应用转换操作和输出操作来定义流计算
3. 用streamingContext.start()来开始接收数据和处理流程
4. 通过streamingContext.awaitTermination()方法来等待处理结束（手动结束或因为错误而结束）
5. 可以通过streamingContext.stop()来手动结束流计算进程

创建StreamingContext对象

- 如果要运行一个Spark Streaming程序，就需要首先生成一个StreamingContext对象，它是Spark Streaming程序的主入口
- 可以从一个SparkContext对象创建一个StreamingContext对象。
- 在pyspark中的创建方法：进入pyspark以后，就已经获得了一个默认的SparkConext对象，也就是sc。因此，可以采用如下方式来创建StreamingContext对象：

```
from pyspark.streaming import StreamingContext
ssc = StreamingContext(sc, 1)
```

创建StreamingContext对象

如果是编写一个独立的Spark Streaming程序，而不是在pyspark中运行，则需要通过如下方式创建StreamingContext对象：

```
from pyspark import SparkContext, SparkConf
from pyspark.streaming import StreamingContext
conf = SparkConf()
conf.setAppName('TestDStream')
conf.setMaster('local[2]')
sc = SparkContext(conf = conf)
ssc = StreamingContext(sc, 1)
```