# 大数据简介

简单来说，大数据要解决的问题包括两个：大量数据的存储和处理。

第一个被是实现为**分布式存储**，包括分布式文件系统（非结构化数据）以及分布式数据库（结构化数据）。

第二个则被实现为**分布式计算**，包括MapReduce，SQL查询，脚本语言，流处理等等。

目前开源的大数据生态系统，主要只有**Hadoop**，以及部分基于Hadoop的**Spark**。而专有的生态系统，著名的就是**Google**的大数据架构（也是Hadoop的启发者），此外就是**Amazon**的大数据架构。

## 分布式数据存储

分布式存储简而言之就是将数据存储在一个以上的节点上。这些节点可能位于同一地点（比如集群计算），或者分布在网络的各处。在大数据中，分布式存储通常可以分为两个部分，**分布式数据库**和**分布式文件系统**。

### CAP定理

在[理论计算机科学](https://zh.wikipedia.org/wiki/%E7%90%86%E8%AB%96%E8%A8%88%E7%AE%97%E6%A9%9F%E7%A7%91%E5%AD%B8)中，CAP定理（CAP theorem），又被称作布鲁尔定理（Brewer's theorem），它指出对于一个[分布式计算系统](https://zh.wikipedia.org/wiki/%E5%88%86%E5%B8%83%E5%BC%8F%E8%AE%A1%E7%AE%97" \o "分布式计算)来说，不可能同时满足以下三点：

* **一致性（Consistency）** （等同于所有节点访问同一份最新的数据副本）
* [**可用性**](https://zh.wikipedia.org/wiki/%E5%8F%AF%E7%94%A8%E6%80%A7)**（Availability）**（每次请求都能获取到非错的响应——但是不保证获取的数据为最新数据）
* [**分区容错性**](https://zh.wikipedia.org/w/index.php?title=%E7%BD%91%E7%BB%9C%E5%88%86%E5%8C%BA&action=edit&redlink=1)**（Partition tolerance）**（以实际效果而言，分区相当于对通信的时限要求。系统如果不能在时限内达成数据一致性，就意味着发生了分区的情况，必须就当前操作在C和A之间做出选择。）

根据定理，分布式系统只能满足三项中的两项而不可能满足全部三项。理解CAP理论的最简单方式是想象两个节点分处分区两侧。允许至少一个节点更新状态会导致数据不一致，即丧失了C性质。如果为了保证数据一致性，将分区一侧的节点设置为不可用，那么又丧失了A性质。除非两个节点可以互相通信，才能既保证C又保证A，这又会导致丧失P性质。

### 分布式文件系统

分布式文件系统和分布式存储之间的界限不是很明显，一个典型的特征是分布式文件系统中的文件可以像本地文件一样被访问。

分布式文件系统也有可能通过IBM的**DDM（Distributed Data Management Architecture）**实现，在这个框架/协议中，对远程文件的操作（创建/访问/管理）和本地文件是一样的，只是通过DDM定义的协议将这些信息传输到远程机器上，远程机器上的DDM服务器负责解释这些协议，并对它的本地文件（即远程文件）进行操作。

### 分布式数据库

除了分布式文件系统之外，分布式存储的另一大构成是分布式数据库。分布式数据库指的是数据存储在不同计算机上的数据库，可以是同一地点，也可以是网络上的不同位置。

数据库用于存储结构化数据（structured data）。**DBMS**（Database Management System，数据库管理系统）则向用户和应用程序提供对数据库的操作接口。

数据库分成**关系型数据库**（Relational DB，或者**SQL数据库**）和**非关系型数据库**（Non-relational DB，或者**NoSQL数据库**）。

分布式数据库通常是**NoSQL数据库**（因为访问速度快）。有些分布式数据库有丰富的查询接口，有些则只有简单的key-value语义。典型的例如**Google BigTable、Amazon Dynamo、Windows Azure Storage**。

分布式数据库有一个统一的[数据库管理系统](https://zh.wikipedia.org/wiki/%E6%95%B0%E6%8D%AE%E5%BA%93%E7%AE%A1%E7%90%86%E7%B3%BB%E7%BB%9F)来进行管理，称为**DDBMS（分布式数据库管理系统）**。

#### ACID特性

ACID，是指**[数据库管理系统](https://zh.wikipedia.org/wiki/%E6%95%B0%E6%8D%AE%E5%BA%93%E7%AE%A1%E7%90%86%E7%B3%BB%E7%BB%9F" \o "数据库管理系统)（**[**DBMS**](https://zh.wikipedia.org/wiki/DBMS)**）**在写入或更新资料的过程中，为保证[事务](https://zh.wikipedia.org/wiki/%E6%95%B0%E6%8D%AE%E5%BA%93%E4%BA%8B%E5%8A%A1" \o "数据库事务)（transaction）是正确可靠的，所必须具备的四个特性：**[原子性](https://zh.wikipedia.org/w/index.php?title=%E5%8E%9F%E5%AD%90%E6%80%A7&action=edit&redlink=1" \o "原子性（页面不存在）)**（atomicity，或称不可分割性）、**[一致性](https://zh.wikipedia.org/wiki/%E4%B8%80%E8%87%B4%E6%80%A7" \o "一致性)**（consistency）、**[隔离性](https://zh.wikipedia.org/wiki/%E9%9A%94%E9%9B%A2%E6%80%A7" \o "隔离性)**（isolation，又称独立性）、[**持久性**](https://zh.wikipedia.org/w/index.php?title=%E6%8C%81%E4%B9%85%E6%80%A7&action=edit&redlink=1)（durability）。

* **Atomicity（原子性）**：一个事务（transaction）中的所有操作，或者全部完成，或者全部不完成，不会结束在中间某个环节。事务在执行过程中发生错误，会被恢复（Rollback）到事务开始前的状态，就像这个事务从来没有执行过一样。即，事务不可分割、不可约简。
* **Consistency（一致性）**：在事务开始之前和事务结束以后，数据库的完整性没有被破坏。这表示写入的资料必须完全符合所有的预设[约束](https://zh.wikipedia.org/wiki/%E6%95%B0%E6%8D%AE%E5%AE%8C%E6%95%B4%E6%80%A7)、[触发器](https://zh.wikipedia.org/wiki/%E8%A7%A6%E5%8F%91%E5%99%A8_(%E6%95%B0%E6%8D%AE%E5%BA%93))、[级联回滚](https://zh.wikipedia.org/w/index.php?title=%E7%BA%A7%E8%81%94%E5%9B%9E%E6%BB%9A&action=edit&redlink=1)等。
* **Isolation（隔离性）**：数据库允许多个并发事务同时对其数据进行读写和修改的能力，隔离性可以防止多个事务并发执行时由于交叉执行而导致数据的不一致。事务隔离分为不同级别，包括读未提交（Read uncommitted）、读提交（read committed）、可重复读（repeatable read）和串行化（Serializable）。
* **Durability（持久性）**：事务处理结束后，对数据的修改就是永久的，即便系统故障也不会丢失。

#### SQL数据库

关系型数据库以**表-行-列（Table-row-column）**的形式组织，绝大部分（理论上所有的）都使用SQL语言进行操作，因此也称为**SQL数据库**（这个称呼是在**NoSQL数据库**这个称呼流行起来之后）。SQL数据库从1980年代开始成为主流的数据库。

关系型数据库使用**RDBMS**（Relational DataBase Management System）来操作。

SQL数据库在一些数据敏感的应用中表现了糟糕的性能，例如为巨量文档创建索引、高流量网站的网页服务、以及发送式流媒体等。因此导致了NoSQL数据库的流行

#### NoSQL数据库

**非关系型数据库**因为采用其它查询语言，因此也叫**NoSQL数据库**。非关系型数据库出现于1960年代，但是直到2000年代流行之后，才有了NoSQL数据库这个称呼。主要指代分布式的、非关系型、不提供**ACID**的数据库，通常提供弱一致性的保证。

NoSQL数据库提供了**表格式**之外的数据库存储检索机制，比如**key-value**（键-值）、**wide column**、**graph**或者**document**等，这些存储检索机制使得某些操作快于SQL数据库。不同的存储方式适用于不同的工作场景。

很多NoSQL在**一致性**（ACID中的C）上做了妥协，以获得足够的速度，以及分布式的存储能力。这些NoSQL数据库大都提供了一种叫做“**最终一致性**”的保证，即改动最终会在所有的节点上达成一致。因此带来的问题就是有时候读操作会返回并非最新的数据。

著名的NoSQL包括**MongoDB**、**Cassandra、Redis、Neo4j、BigTable、HBase**等。以下是根据存储格式分类的NoSQL数据库：

* [**Column**](https://en.wikipedia.org/wiki/Column_(data_store)): [Accumulo](https://en.wikipedia.org/wiki/Accumulo" \o "Accumulo), [**Cassandra**](https://en.wikipedia.org/wiki/Apache_Cassandra), [Scylla](https://en.wikipedia.org/wiki/Scylla_(database)), [Druid](https://en.wikipedia.org/wiki/Druid_(open-source_data_store)), [**HBase**](https://en.wikipedia.org/wiki/HBase), [Vertica](https://en.wikipedia.org/wiki/Vertica), **BigTable**.
* [**Document**](https://en.wikipedia.org/wiki/Document-oriented_database): [Apache CouchDB](https://en.wikipedia.org/wiki/Apache_CouchDB), [ArangoDB](https://en.wikipedia.org/wiki/ArangoDB" \o "ArangoDB), [BaseX](https://en.wikipedia.org/wiki/BaseX" \o "BaseX), [Clusterpoint](https://en.wikipedia.org/wiki/Clusterpoint" \o "Clusterpoint), [Couchbase](https://en.wikipedia.org/wiki/Couchbase), [Cosmos DB](https://en.wikipedia.org/wiki/Cosmos_DB),

[IBM Domino](https://en.wikipedia.org/wiki/Lotus_Notes), [MarkLogic](https://en.wikipedia.org/wiki/MarkLogic" \o "MarkLogic), [**MongoDB**](https://en.wikipedia.org/wiki/MongoDB), [OrientDB](https://en.wikipedia.org/wiki/OrientDB" \o "OrientDB), [Qizx](https://en.wikipedia.org/wiki/Qizx" \o "Qizx), [RethinkDB](https://en.wikipedia.org/wiki/RethinkDB" \o "RethinkDB)

* [**Key-value**](https://en.wikipedia.org/wiki/Key-value_store): [Aerospike](https://en.wikipedia.org/wiki/Aerospike_(database)), [Apache Ignite](https://en.wikipedia.org/wiki/Apache_Ignite), [ArangoDB](https://en.wikipedia.org/wiki/ArangoDB" \o "ArangoDB), [Berkeley DB](https://en.wikipedia.org/wiki/Berkeley_DB), [Couchbase](https://en.wikipedia.org/wiki/Couchbase), [Dynamo](https://en.wikipedia.org/wiki/Dynamo_(storage_system)),

[FoundationDB](https://en.wikipedia.org/wiki/FoundationDB), [InfinityDB](https://en.wikipedia.org/wiki/InfinityDB), [MemcacheDB](https://en.wikipedia.org/wiki/MemcacheDB), [MUMPS](https://en.wikipedia.org/wiki/MUMPS), [Oracle NoSQL Database](https://en.wikipedia.org/wiki/Oracle_NoSQL_Database), [OrientDB](https://en.wikipedia.org/wiki/OrientDB" \o "OrientDB),

[**Redis**](https://en.wikipedia.org/wiki/Redis), [Riak](https://en.wikipedia.org/wiki/Riak" \o "Riak), [SciDB](https://en.wikipedia.org/wiki/SciDB" \o "SciDB), SDBM/Flat File [dbm](https://en.wikipedia.org/wiki/Dbm" \o "Dbm), [ZooKeeper](https://en.wikipedia.org/wiki/Apache_ZooKeeper" \o "Apache ZooKeeper)

* [**Graph**](https://en.wikipedia.org/wiki/Graph_database): [AllegroGraph](https://en.wikipedia.org/wiki/AllegroGraph" \o "AllegroGraph), [ArangoDB](https://en.wikipedia.org/wiki/ArangoDB), [InfiniteGraph](https://en.wikipedia.org/wiki/InfiniteGraph), [Apache Giraph](https://en.wikipedia.org/wiki/Apache_Giraph), [MarkLogic](https://en.wikipedia.org/wiki/MarkLogic), [**Neo4J**](https://en.wikipedia.org/wiki/Neo4J),

[OrientDB](https://en.wikipedia.org/wiki/OrientDB), [Virtuoso](https://en.wikipedia.org/wiki/Virtuoso_Universal_Server)

#### NewSQL数据库

NewSQL是一类新型的RDMS，试图提供像**NoSQL数据库**一样的处理在线事务的性能，同时能保持传统**SQL数据库**的**ACID**性。

虽然各种NewSQL数据库内部架构不尽相同，但他们都有两个共同的特点：

1. 支持关系数据模型（relational data model）
2. 使用SQL作为接口

NewSQL数据库可以被分为三类：

* **新架构**：这类NewSQL数据库完全是重新构建的平台，从开始设计的时候就考虑到分布式架构、并发控制、流控制以及分布式查询等等功能。比如Amazon Aurora、Google Spanner、Apache Ignite等等。
* **SQL引擎**：这类NewSQL数据库使用原有的数据库存储结构，但提供了高度优化过的SQL引擎。比如MySQL Cluster、MariaDB Columnstore、SQL Server（有Columnstore和InMemory功能）。
* **透明分片（Transparent sharding）**：这类NewSQL数据库通过**分片中间层**（Sharding middleware）来将数据库自动分片到各个节点上。比如ScaleBase和Vitess。

## 分布式计算

### MapReduce

MapReduce是[Google](https://zh.wikipedia.org/wiki/Google" \o "Google)提出的一个[软件架构](https://zh.wikipedia.org/wiki/%E8%BD%AF%E4%BB%B6%E6%9E%B6%E6%9E%84" \o "软件架构)，用于大规模数据集（大于1[TB](https://zh.wikipedia.org/wiki/Terabyte)）的[并行运算](https://zh.wikipedia.org/wiki/%E4%B8%A6%E8%A1%8C%E9%81%8B%E7%AE%97" \o "并发运算)。概念“Map（映射）”和“Reduce（归纳）”，及他们的主要思想，都是从[函数式编程语言](https://zh.wikipedia.org/wiki/%E5%87%BD%E6%95%B0%E5%BC%8F%E7%BC%96%E7%A8%8B%E8%AF%AD%E8%A8%80" \o "函数式编程语言)借来的，还有从[矢量编程语言](https://zh.wikipedia.org/w/index.php?title=%E7%9F%A2%E9%87%8F%E7%BC%96%E7%A8%8B%E8%AF%AD%E8%A8%80&action=edit&redlink=1)借来的特性。

当前的软件实现是指定一个Map（映射）函数，用来把一组键值对映射成一组新的键值对，指定并发的Reduce（归纳）函数，用来保证所有映射的键值对中的每一个共享相同的键组。

### 分布式流处理

流处理是一种重要的[大数据](https://baike.baidu.com/item/%E5%A4%A7%E6%95%B0%E6%8D%AE/1356941)处理手段，其主要特点是其处理的数据是源源不断且实时到来的。分布式流处理是一种面向动态数据的细粒度处理模式，基于分布式内存，对不断产生的动态数据进行处理。其对数据处理的快速，高效，低延迟等特性，在大数据处理中发挥越来越重要的作用。

分布式流处理是指基于[分布式技术](https://baike.baidu.com/item/%E5%88%86%E5%B8%83%E5%BC%8F%E6%8A%80%E6%9C%AF/4110919" \t "_blank)对流数据进行动态处理，具有较好的伸缩性、实时性和开放性。分布式流处理通常部署于大规模集群中，通常将流数据处理过程抽象为一个**有向无环图（DAG）**。调度算法则负责将有向无环图中的组件合理的分配至集群中的可用服务器上。流处理系统作为流处理作业的平台，负责所有集群资源的管理和分配。对于用户提交的流处理作业，流处理系统需考虑该作业所处理的数据量及集群中不同节点的负载，并将其合理的分配到集群的不同作业节点上。因而如何最大化集群的利用率，即最大化集群可容纳的流处理作业数量。好的任务分配策略能够加快数据处理的速率，提升系统整体吞吐量，均衡集群中的负载，或者减少集群中的资源占用率。

### 分布式锁服务

**DLM（Distributed Lock Manager）**，分布式锁服务，用于对共享资源进行同步访问管理。DLM仅仅定义通用的资源概念，资源可以是一个文件，一条记录，一块共享内存，或者任何其他东西。

DLM可以定义资源的层级关系，比如：数据库-表-记录-字段。

DLM的实现包括Google的**Chubby**，Hadoop的**ZooKeeper**等等

# Google生态系统

很容易理解，Google是大数据的引领者。在很多互联网公司需要用到大数据平台之前，甚至诞生之前，Google就得面对大数据的存储和处理，它得向用户提供网络搜索的结果。目前主流的开源大数据框架Apache Hadoop的几个核心组件，就是对Google的几篇论文的开源实现。

## GFS（分布式文件系统）

Google文件系统（Google File System，**GoogleFS，GFS**），一种专有[分布式文件系统](https://zh.wikipedia.org/wiki/%E5%88%86%E5%B8%83%E5%BC%8F%E6%96%87%E4%BB%B6%E7%B3%BB%E7%BB%9F" \o "分布式文件系统)，由[Google](https://zh.wikipedia.org/wiki/Google)公司开发，运行于[Linux](https://zh.wikipedia.org/wiki/Linux" \o "Linux)平台上。

尽管Google并没有将该系统的软件部分作为开源软件发布，但Google在2003年公布了该系统的一些技术细节，这篇论文后来被实现为开源的**HDFS**。

2013年，Google公布了**Colossus**项目，作为下一代的Google文件系统。

## Google MapReduce（分布式计算）

Google专有版本的**MapReduce**

## BigTable（NoSQL）

BigTable是一种[压缩](https://zh.wikipedia.org/wiki/%E6%95%B0%E6%8D%AE%E5%8E%8B%E7%BC%A9" \o "数据压缩)的、高性能的、高可扩展性的，基于[Google文件系统](https://zh.wikipedia.org/wiki/Google%E6%AA%94%E6%A1%88%E7%B3%BB%E7%B5%B1)（Google File System，GFS）的数据存储系统，用于存储大规模结构化数据。BigTable是**NoSQL（非关系型）数据库**。

BigTable发展于2004年，现今已成为Google的应用程序。像是[MapReduce](https://zh.wikipedia.org/wiki/MapReduce" \o "MapReduce)就常透过BigTable来存储或更改数据，其他还有[Google Reader](https://zh.wikipedia.org/wiki/Google_Reader" \o "Google Reader)、[Google Maps](https://zh.wikipedia.org/wiki/Google_Maps" \o "Google Maps)、[Google Book Search](https://zh.wikipedia.org/wiki/Google_Book_Search" \o "Google Book Search)、"My Search History"、[Google Earth](https://zh.wikipedia.org/wiki/Google_Earth" \o "Google Earth)、[Blogger.com](https://zh.wikipedia.org/wiki/Blogger.com" \o "Blogger.com)、[Google Code](https://zh.wikipedia.org/wiki/Google_Code" \o "Google Code) hosting、[Orkut](https://zh.wikipedia.org/wiki/Orkut" \o "Orkut)、[YouTube](https://zh.wikipedia.org/wiki/YouTube" \o "YouTube)以及[Gmail](https://zh.wikipedia.org/wiki/Gmail" \o "Gmail)等。

2015年，Google提供了一个公共版本的BigTable作为云服务。

## BigQuery（网络服务）

BigQuery是一种**RESTful**的网络服务，提供了对Google存储上的大数据集进行交互式的分析。

BigQuery提供的功能包括：管理数据、查询、整合、访问控制。

## Chubby（分布式锁服务）

Google专有的**分布式锁服务**（DLM）

## Sawzall（脚本语言）

Sawzall是Google开发的脚本语言，一个过程式**DSL（Domain-specific language，领域专用语言）**，用于处理海量的日志记录（log record）。在Google内部，Sawzall在大部分场景已经被**Lingo（Logs in Go）**取代。

由于Google仅仅开放了Sawzall，没开放MapReduce，其用处不大。

## Spanner（NewSQL）

Spanner是Google开发的专有**NewSQL**数据库。Spanner并非是个纯粹的**关系（SQL）数据库**，因为其每张表必须有主键。

## Google F1（SQL）

Google F1是Google的专有关系SQL数据库，基于Spanner实现，用于替换原有的MySQL的Google变种。

## Dremel（查询引擎）

Dremel是Google开发的对大数据集（large dataset）进行交互式查询的引擎。**BigQuery**用Dremel作为查询引擎。

Dremel启发了**Drill**、**Impala**和Dremio。

## Pregel（图形处理）

Google开发的专有的分布式图形处理系统。

# Hadoop生态系统

Apache Hadoop是一系列开源大数据计算的框架/工具/软件的组合，使得海量数据可以在网络中的大量计算机中被存储及处理。Hadoop中所有模块设计都有一个基本假设：硬件的错误是常见的，并且可以被系统自动处理。

Hadoop框架本身由Java实现，以及少量的C以及脚本。

Hadoop底层的存储系统主要是**HDFS**，但也支持其他一些文件系统：**FTP**文件系统、**Amazon S3**、**Windows Azure Storage Blobs（WASB）**等。

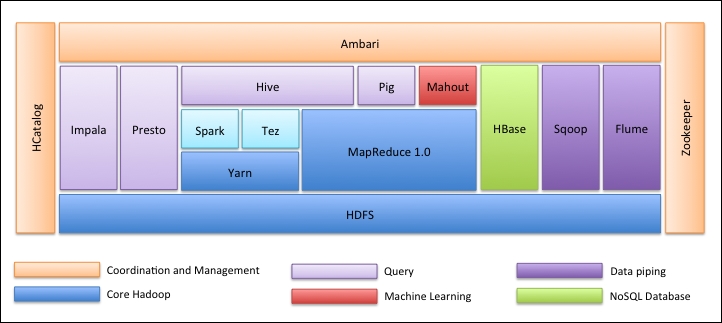
Hadoop最初的两个组件（也是两个核心组件）**Hadoop MapReduce**和**HDFS**，受启发于Google关于**Google MapReduce**（2004年）和**Google FS**（2003年）的论文，但Google的MapReduce和GFS并没有开源。

通常认为Hadoop的核心包括：

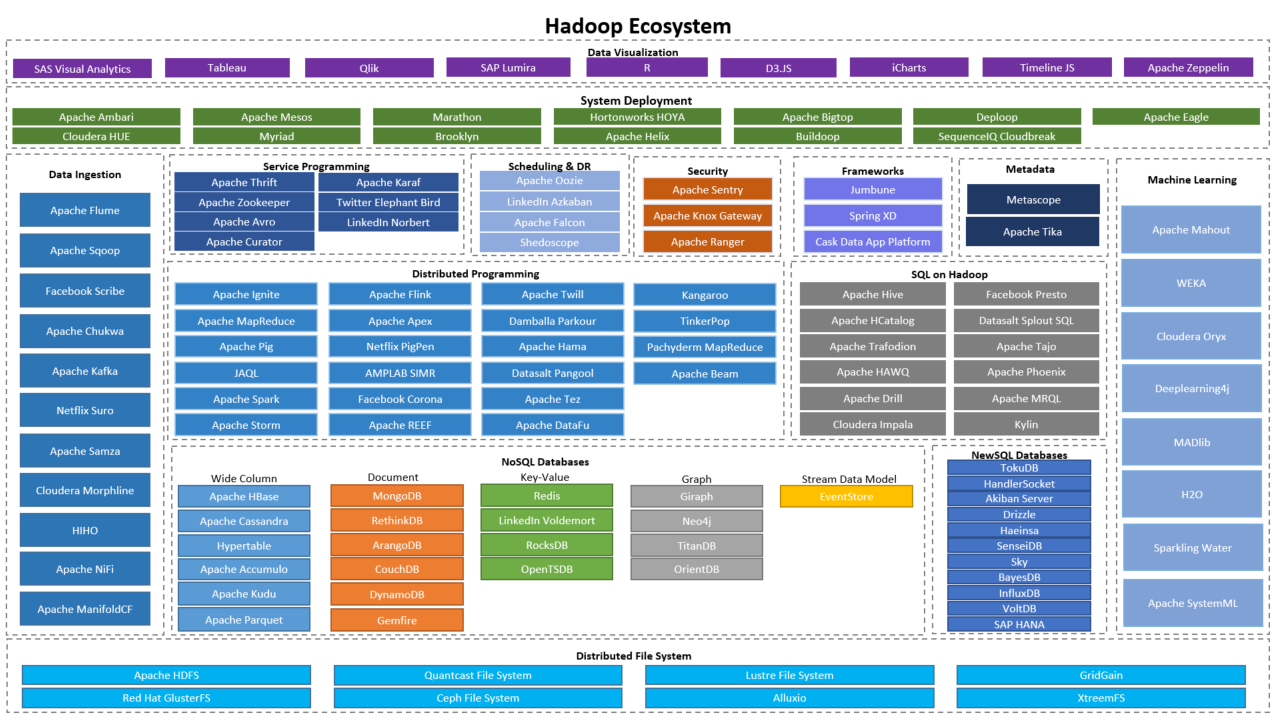
* 底层的存储系统**HDFS**
* **Hadoop MapReduce**编程模型
* 用于支持其他组件的基础库及工具**Hadoop Common**
* 资源调度器**YARN**。

除此之外，Hadoop的生态系统中还包括Ambari、Avro、Drill、Flume、HBase、HCatalog、Hive、Impala、Mahout、Oozie、Phoenix、Pig、Presto、Spark、Sqoop、Storm、ZooKeeper等等组件。

下图是Hadoop生态系统中主要的一些组件：



下图是更加完整的一个Hadoop生态系统组件构成



## Hadoop common

## HDFS（分布式文件系统）

**HDFS（Hadoop Distributed File System）**是一个分布式的文件系统，由以下几部分组成：

**Name node**：HDFS中只有一个Name node，也叫**Master node**，Name node中包含了文件系统的元数据。

**Data node**：Data node也叫**Slave node**，存放了文件数据。每个Data node每隔3秒会发送一个心跳信息给Name node。如果Name node有2分钟内没有收到某个Data node的信息，它会开始进行这个Data node里数据的备份工作。

**Secondary name node**：也叫**checkpoint node**，是Name node的助手节点，用于存放文件系统元数据的checkpoints。

**Job tracker**：接收来自于MapReduce的执行请求，从Name node获得数据的位置信息。

**Task tracker**：Job tracker的slave节点，从Job tracker接收代码，然后用这些代码处理文件，这个过程被称为Mapper

HDFS名义上有一个Name node和一簇Data node，也可以进行Name node的冗余备份，各个节点之间通过socket进行通信。HDFS通过不同节点的数据备份来提高可靠性，缺省的replication value是3，也就是说同样的数据在3个Data node有保存：2个在同一个机架（rack）中，另一个在其他机架中。Data node之间会相互通信以确保数据的平衡和备份。

## Hadoop MapReduce（分布式计算）

MapReduce这个词本身指的是一种来自于Google的编程模型，用于大规模数据集的并行计算。

MapReduce程序由映射（Map）和归纳（Reduce）两个步骤构成。

Hadoop MapReduce是根据Google论文的一个开源实现。

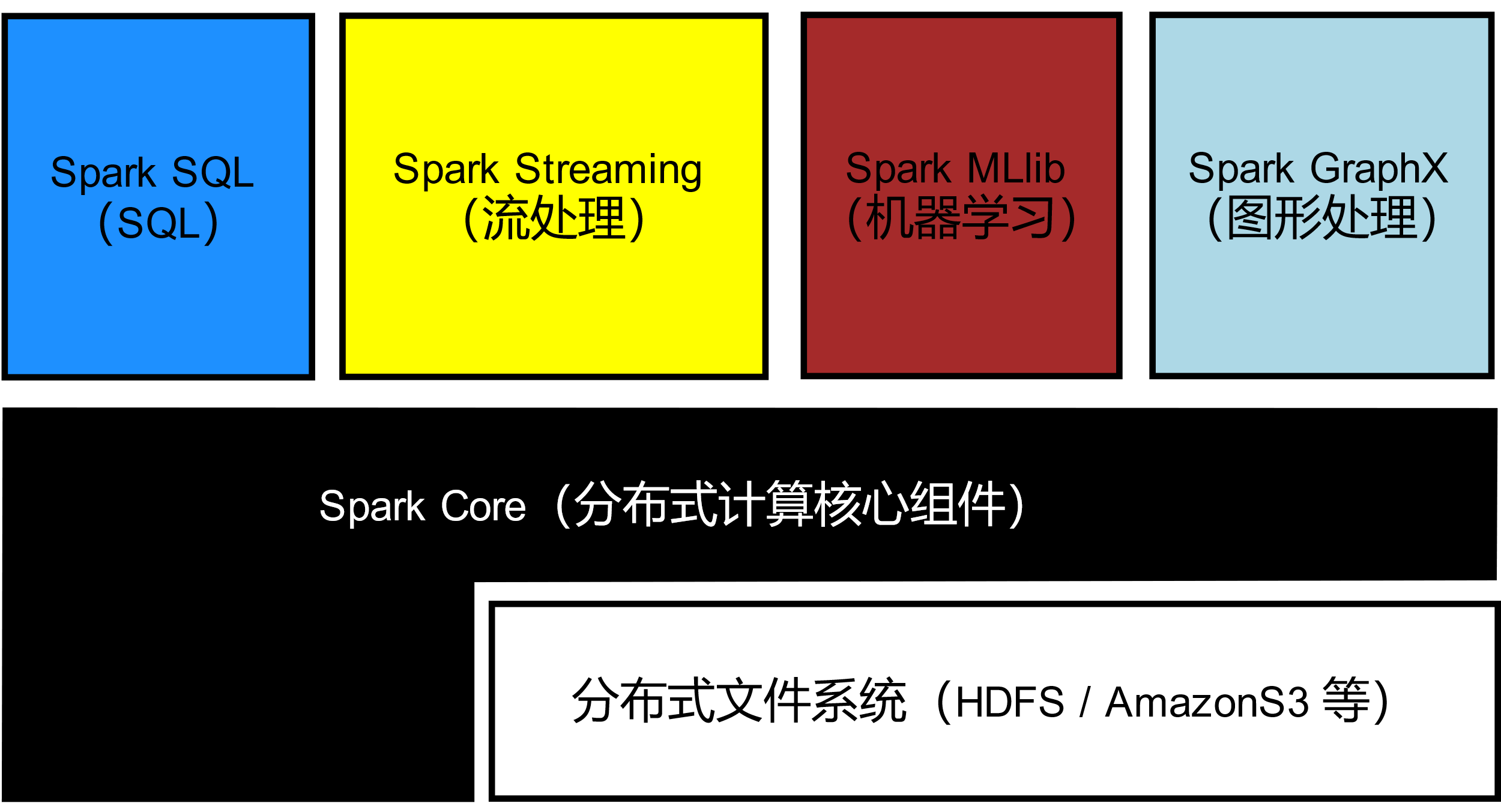


## Spark（分布式框架）

Spark是一个分布式运算框架，相对于[Hadoop](https://zh.wikipedia.org/wiki/Apache_Hadoop)的[MapReduce](https://zh.wikipedia.org/wiki/MapReduce" \o "MapReduce)会在运行完工作后将中介数据存放到磁盘中，Spark使用了存储器内运算技术，能在数据尚未写入硬盘时即在存储器内分析运算。Spark在存储器内运行程序的运算速度能做到比Hadoop MapReduce的运算速度快上100倍，即便是运行程序于硬盘时，Spark也能快上10倍速度。Spark允许用户将数据加载至集群存储器，并多次对其进行查询，非常适合用于[机器学习](https://zh.wikipedia.org/wiki/%E6%9C%BA%E5%99%A8%E5%AD%A6%E4%B9%A0)算法。

Spark既被视为Hadoop的替代，也被视为Hadoop生态的一部分。整个Spark框架相对独立，有自己的组件构成，并且可以独立运行（独立模式）。又可以作为一个Hadoop组件运行。

以下是Spark的构成：



### Spark Core（基本组件）

Spark Core是Spark的核心组件，包含Spark的基本功能：任务调度，内存管理，容错机制等，内部定义了**RDD（Resilient Distributed Dataset，弹性分布式数据集）**，提供了很多APIs来创建和操作这些RDD。为其他组件提供底层的服务。

RDDs可以透过引用外部存储系统的数据集创建（例如：共享文件系统、HDFS、HBase或其他 Hadoop 数据格式的数据源）。或者是透过在现有RDDs的转换而创建（比如：map、filter、reduce、join等等）。

### Spark SQL（SQL）

Spark SQL是基于**Spark Core**之上的一个组件，它定义了一种数据抽象化概念**DataFrame**（之前被称为**SchemaRDD**），DataFrame提供了对结构化和半结构化数据的支持。

Spark SQL提供了一个DSL（Domain-specific language，领域特定语言）来在Java、Scala、Python里操作DataFrame。它还提供了**命令行**和**ODBC／JDBC**的SQL语言支持。

### Spark Streaming（流处理）

Spark Streaming充分利用Spark核心的快速调度能力来运行流分析。它截取小批量的数据并对之运行RDD转换。这种设计使流分析可在同一个引擎内使用同一组为批量分析编写而撰写的应用程序代码。

其带来的缺点就是积攒批量数据的时间所带来的延迟，其他流处理引擎（比如**Storm**或者**Flink**）则是通过事件（event）触发。

### MLlib（机器学习）

MLlib是Spark上分布式机器学习框架。Spark分布式存储器式的架构比Hadoop磁盘式的[Apache Mahout](https://zh.wikipedia.org/w/index.php?title=Apache_Mahout&action=edit&redlink=1" \o "Apache Mahout（页面不存在）)快上10倍（在Mahout使用Spark底层之前）。MLlib可使用许多常见的机器学习和统计算法，简化大规模机器学习时间。

### GraphX（图形处理）

GraphX是Spark上的分布式图形处理框架。因为它是基于不可变的RDD的，因此不适合于处理会更新的图。GraphX提供了两组API，一组类似**Pregel**，一组是更通用的MapReduce类型的API。

## YARN（资源调度）

YARN（Yet Another Resource Negotiator）是Hadoop生态中的资源管理平台，用于管理集群中的计算资源，并将它们分配给用户的程序。

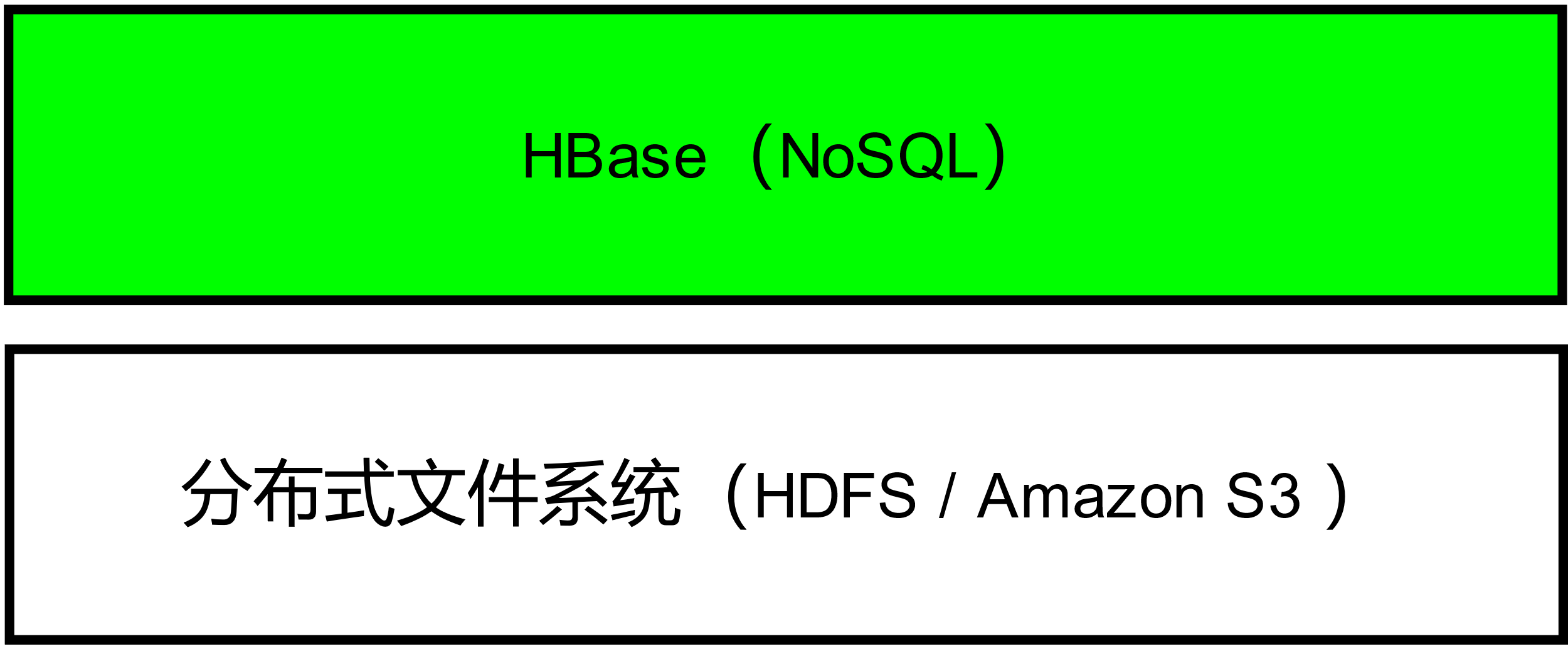
YARN有两个后台进程，分别用于跟踪任务（track job）和监控进度（monitor progress）。

## HBase（NoSQL）

HBase是一个开源的**结构化、分布式、NoSQL**数据库，源于2006年Google关于BigTable的论文，用Java实现。

HBase提供了类似于BigTable的功能：高容错性的存储大容量、稀疏数据（比如从20亿条记录中找出50个最大值）。此外Hbase还有压缩、内存内操作等等BigTable论文中提到的功能。

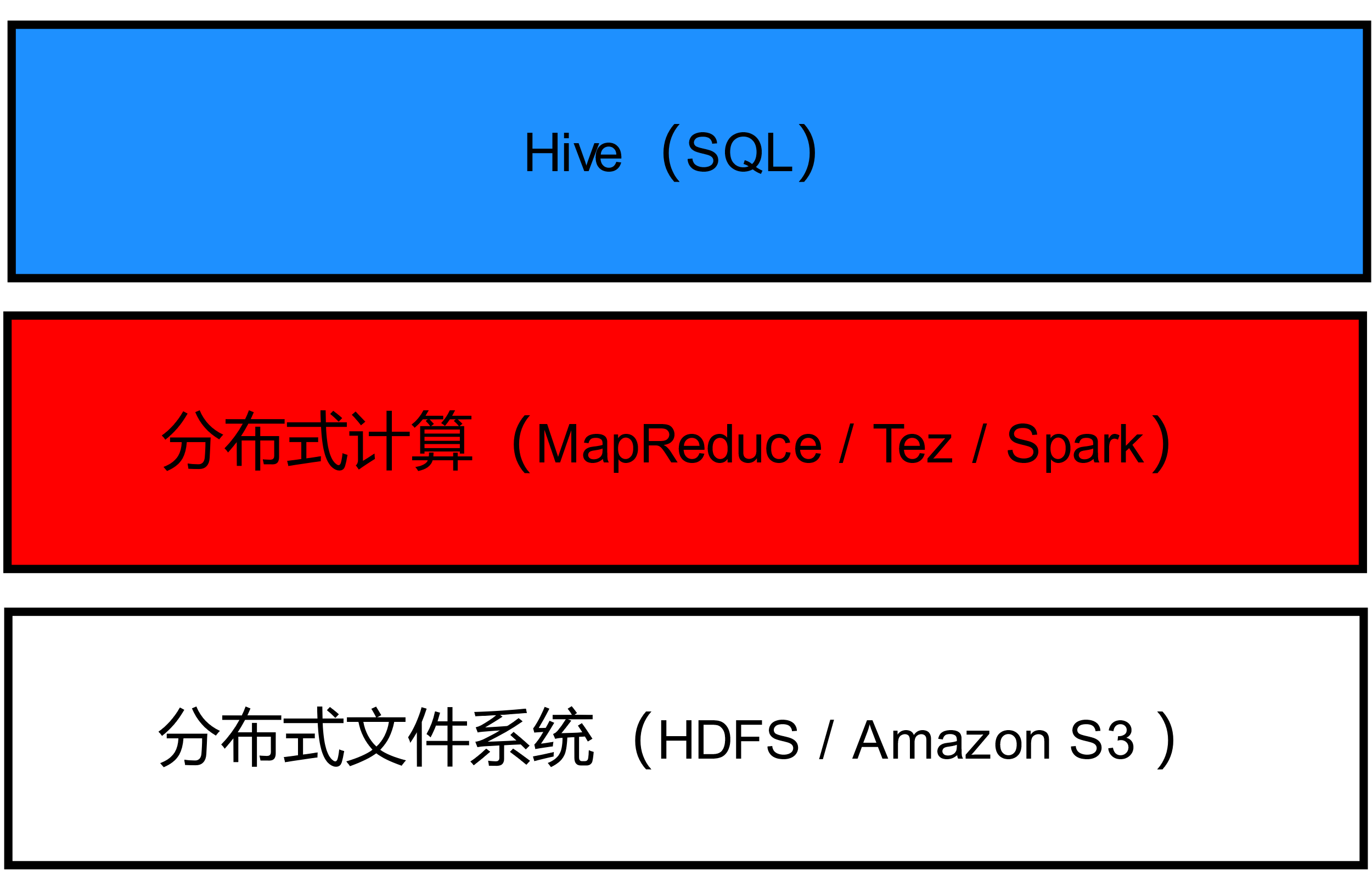
HBase作为Hadoop生态系统的一部分，运行于HDFS之上。HBase中的表可以作为Hadoop MapReduce任务的输入或输出。



## Hive（SQL）

Apache Hive是一个建立在[Hadoop](https://zh.wikipedia.org/wiki/Hadoop)架构之上的数据仓库。它能够提供数据的精炼，查询和分析。

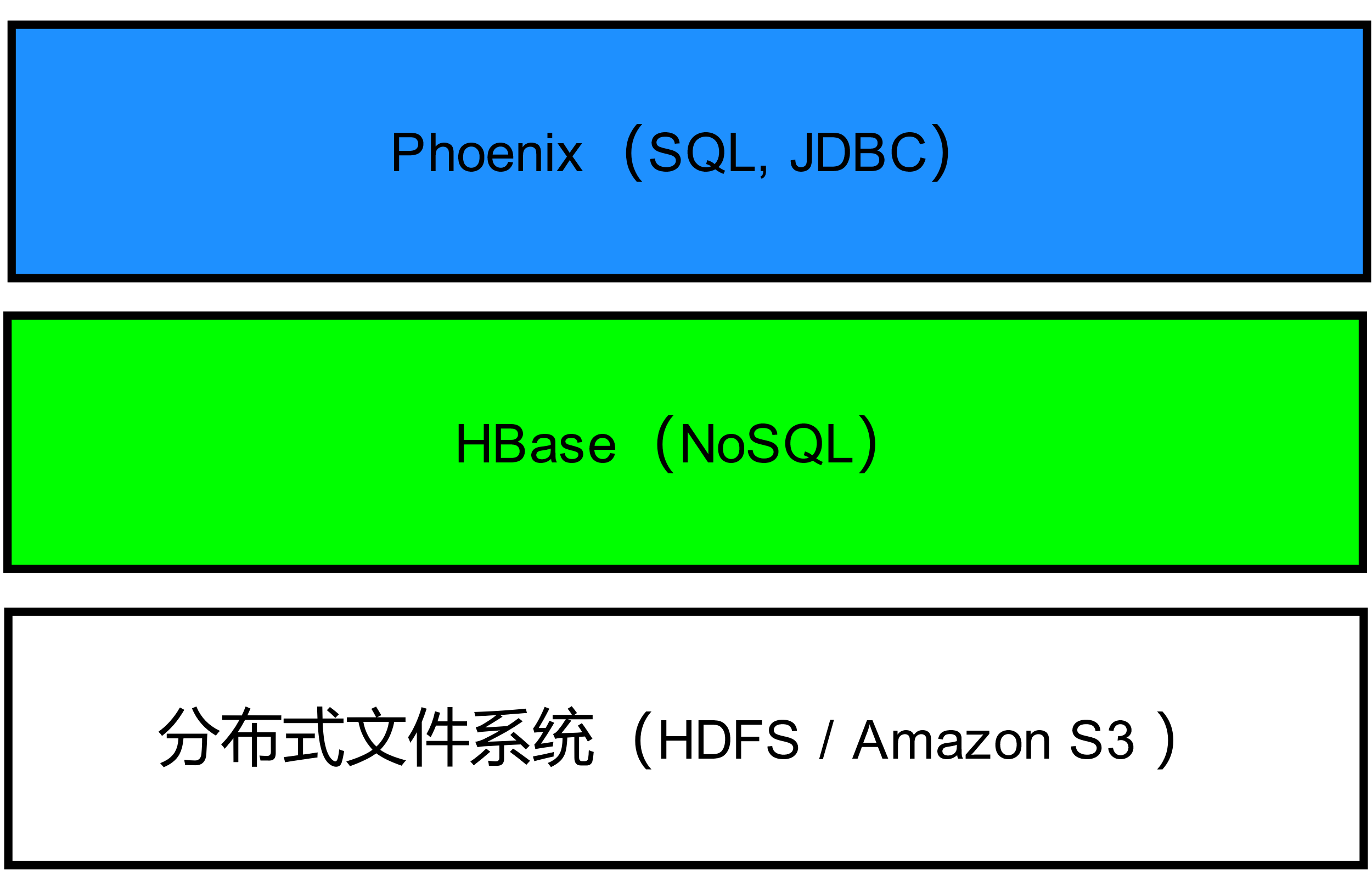
[Hive](https://hive.apache.org/)是基于Hadoop的一个数据仓库工具，可以将结构化的数据文件（在HDFS或者Amazon S3等分布式文件系统上）映射为一张数据库表，并通过一种叫做**HiveQL**（类SQL语句）的查询语言提供简单的[SQL](https://zh.wikipedia.org/wiki/SQL)查询功能，将HiveQL语句转换为**MapReduce**任务运行。其优点是学习成本低，可以快速实现简单的MapReduce统计，不必开发专门的MapReduce应用，十分适合数据仓库的统计分析。



## Phoenix（SQL）

Phoenix是一个基于**HBase**的分布式的**关系型数据库**引擎，向上提供了JDBC接口，隐藏了用户进行CREATE、DELETE、ALTER等SQL操作的时候，对下层HBase提供的NoSQL接口进行操作的复杂性。

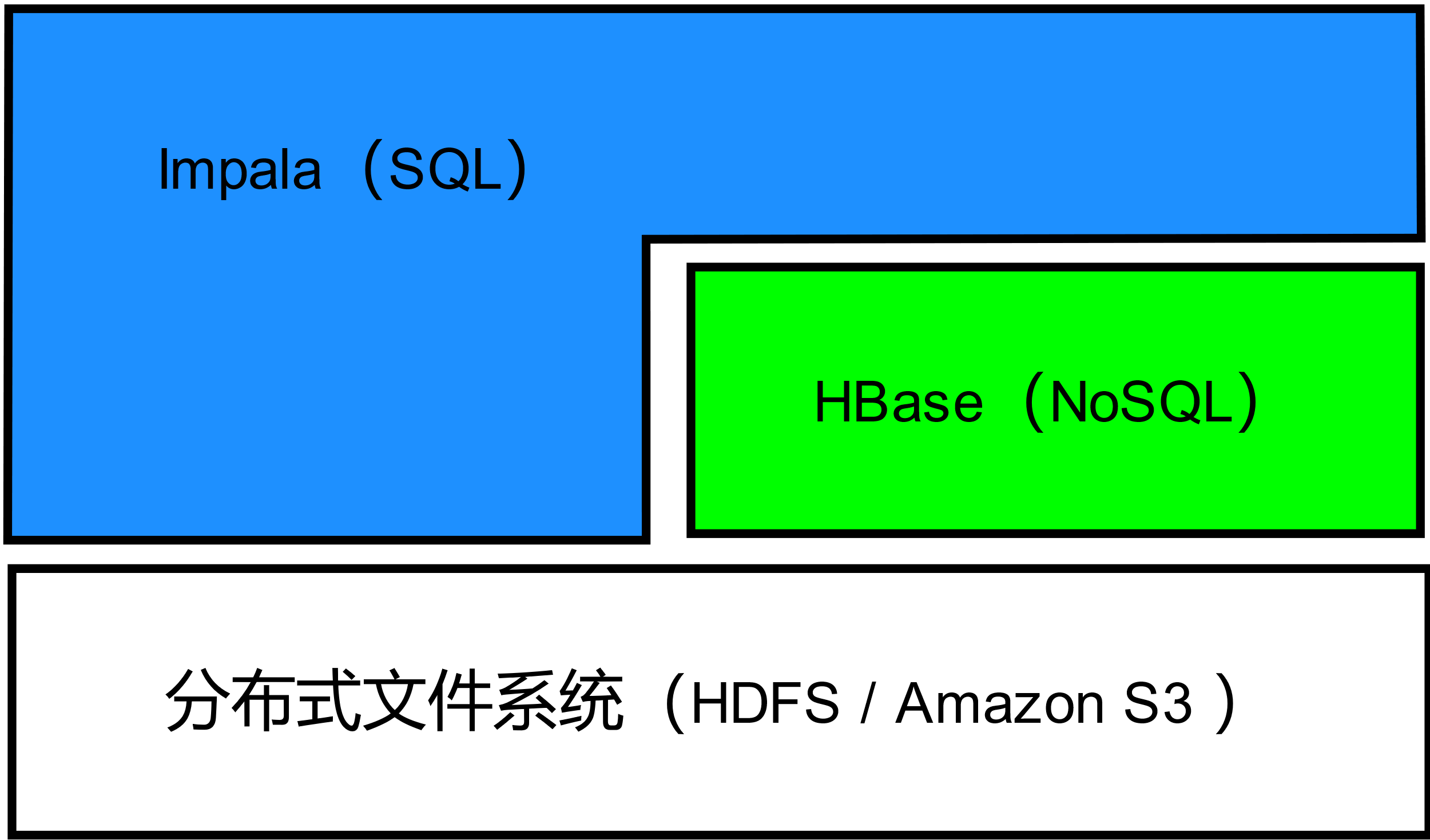
为了构建**低延迟**的程序，Phoenix将查询等操作编译为下层（HBase）的NoSQL API，而不是使用**MapReduce**直接操作底层的HDFS。



## Impala（SQL）

Impala是一个Hadoop系统中的SQL查询引擎，被认为是**Google F1**的一个开源实现。Impala提供了对**HDFS**和**HBase**的SQL查询。

Impala和**Hive**都是Hadoop系统中的SQL查询引擎，但有不同的侧重面。Impala的速度更快，更适合于进行实时查询操作，而Hive更适用于长时间的批处理查询操作



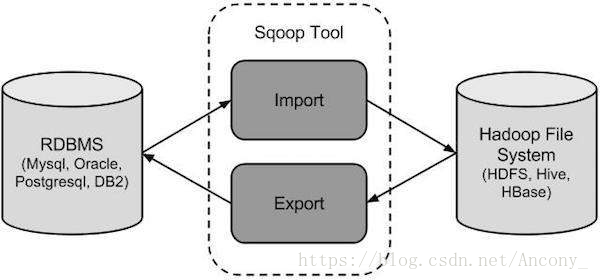
## Ambari（管理）

是一个系统管理工具，用于让系统管理员监控、管理、控制Hadoop

## Sqoop（数据迁移）

Sqoop是一个命令行工具。主要用于在Hadoop与传统的关系型数据库（MySQL ,Oracle ,Postgres等 ）间进行数据的传递，可以将一个关系型数据库中的数据导进到Hadoop的HDFS中，也可以将HDFS的数据导进到关系型数据库中。

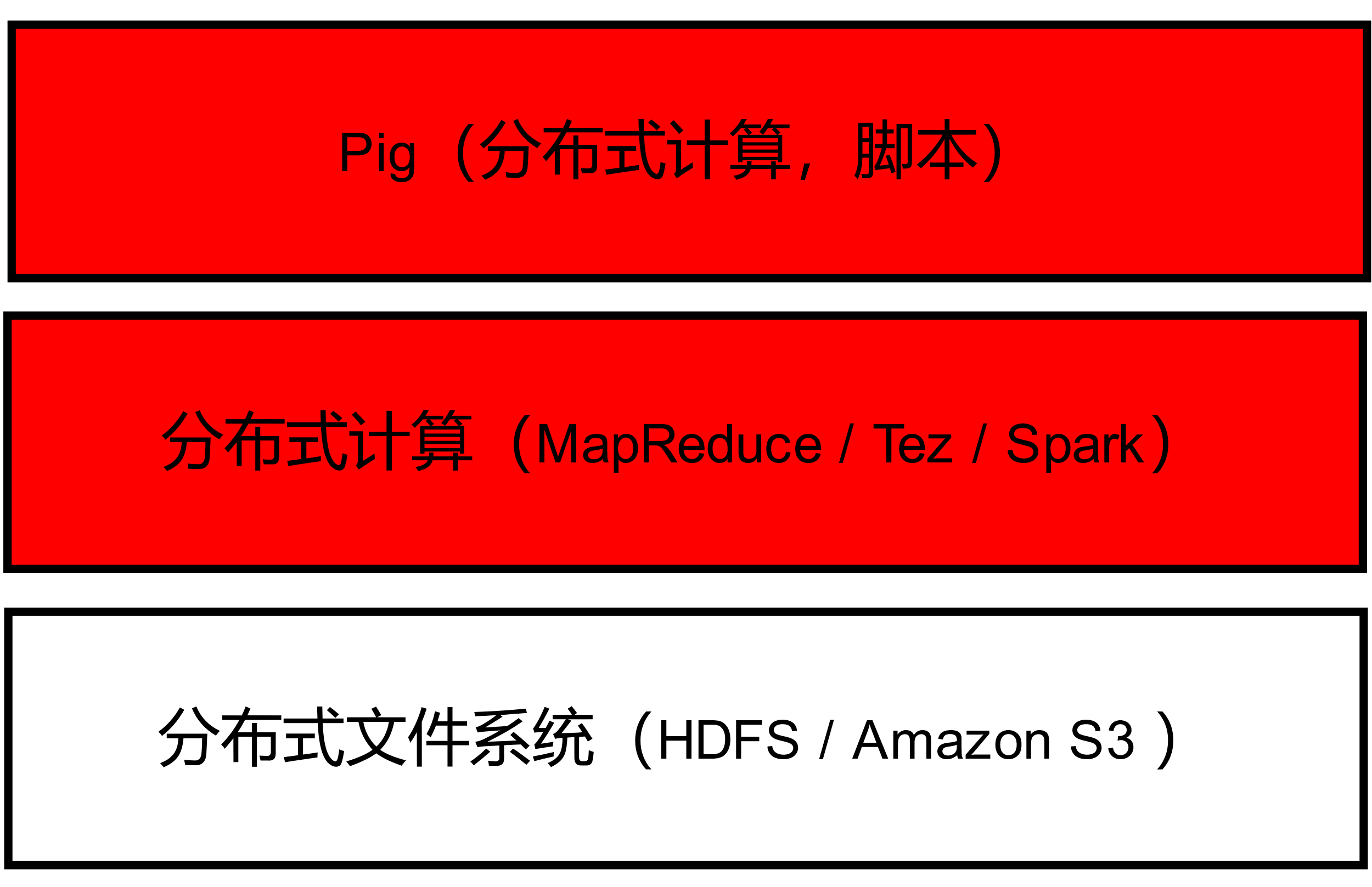
Sqoop也可以用来将数据导入**Hive**或者**HBase**。



## Pig（分布式计算脚本）

Pig是一个基于Hadoop的大规模数据分析平台，这个平台的语言叫**Pig Latin**，该语言的编译器会把Pig Latin的数据分析请求转换为一系列经过优化处理的MapReduce/Tez/Spark运算。Pig为复杂的海量数据并行计算提供了一个简单的操作和编程接口。

Pig Latin可以通过UDF（User-defined function）进行扩展，这些用户定义的函数用Java、Python、JavaScript、Ruby或者Groovy实现，然后在Pig Latin中直接调用。



## Mahout（机器学习）

Mahout是Apache开源的分布式机器学习实现。

## ZooKeeper（分布式锁服务）

ZooKeeper是一个分布式锁服务，是**Google Chubby**的开源实现，是Hadoop和Hbase的重要组件。它是一个为分布式应用提供一致性服务的软件，提供的功能包括：配置维护、域名服务、分布式同步、命名注册等。

## Drill（查询引擎）

Drill是一个开源框架，提供了大数据的交互式分析，给数据密集型的分布式应用，是Google Dremel的开源版本。

## Oozie（任务调度）

Oozie是一个基于服务器的**工作流程（workflow）**调度系统，用于管理Hadoop的任务。

Oozie中国的工作流程被定义为由**控制流程节点（control flow node）**和**行动节点（action node）**组成的**DAG（有向无环图）**。控制流程节点定义workflow的起始、结束、以及失败。行动节点则定义了workflow触发计算任务的机制。

Oozie向各种不同的任务提供了支持，包括**Hadoop MapReduce、HDFS、Pig、SSH**和email等等。也可以扩展Oozie 以支持更多的任务。

Oozie被实现为一个运行在Java servlet容器里的Java网络应用。

## Flume（日志分析）

一个分布式的软件，用于搜集、聚合、移动海量的日志数据。

## Storm（流处理）

Storm是一个分布式的流处理计算框架，由Clojure语言实现，在被Twitter收购后开源。Storm使用用户创建的“**喷口（spouts）**”和“**插销（bolts）**”来定义信息的来源和处理方式，从而分布式的处理流数据。

每个Storm应用都被设计为一个**DAG（Directed acyclic graph，有向无环图）**的拓扑结构，图中的每个定点都是一个**喷口**或者**插销**，每条边是一个流。

## Avro（RPC和数据序列化）

Avro是一个RPC（Remote Procedure Call，远程调用）和数据序列化的框架，使用JSON定义数据类型和协议，并将数据序列化为压缩的二进制格式。

## Giraph（图形处理）

Giraph是一个大数据图形处理系统，通过**MapReduce**执行分布式图形处理。

Giraph计算的输入是由点和两点之间直连的边所组成的图，例如，点可以表示人，边可以表示朋友请求。每个顶点保存一个值，每个边也保存一个值。输入不仅取决于图的[拓扑](https://zh.wikipedia.org/wiki/%E6%8B%93%E6%89%91" \o "拓扑)逻辑，也包括定点和边的初始值。

Giraph基于Google发布的关于**Pregel**的论文实现，Pregel是Google自己的专有分布式图形处理组件。

# Amazon生态系统

## EMR

Elastic MapReduce

## S3

## Redshift

## Athena

# 各生态对应关系

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **Google** | **Hadoop** | **Spark** | **Amazon** |
| **分布式文件系统** | GFS | HDFS |  | S3 |
| **并行计算** | Google MapReduce | Hadoop MapReduce | Spark Core？ | EMR |
| **NoSQL数据库** | BigTable | HBase |  |  |
| **分布式锁服务** | Chubby | ZooKeeper |  |  |
| **脚本语言** | Sawzall | Pig |  |  |
| **SQL接口** |  | Hive | Spark SQL |  |
| **数据迁移** |  | Sqoop |  |  |
| **机器学习** |  | Mahout | Spark MLlib |  |
| **资源调度** |  | Yarn |  |  |
|  | Google F1 | Impala |  |  |
| **DB查询引擎** | Dremel | Drill |  |  |
| **RPC和序列化** | ProtoBuf | Avro |  |  |
| **流处理** |  | Storm | Spark Streaming |  |
| **图形处理** | Pregel | Giraph | Spark GraphX |  |
|  | BigQuery |  |  |  |
|  | Spanner |  |  |  |
|  |  |  |  |  |

# Hadoop生态的演进

来源（<https://zhuanlan.zhihu.com/p/25972902>）

大数据本身是个很宽泛的概念，Hadoop生态圈（或者泛生态圈）基本上都是为了处理超过单机尺度的数据处理而诞生的。你可以把它比作一个厨房所以需要的各种工具。锅碗瓢盆，各有各的用处，互相之间又有重合。你可以用汤锅直接当碗吃饭喝汤，你可以用小刀或者刨子去皮。但是每个工具有自己的特性，虽然奇怪的组合也能工作，但是未必是最佳选择。

**大数据，首先你要能存的下大数据。**传统的文件系统是单机的，不能横跨不同的机器。HDFS（Hadoop Distributed FileSystem）的设计本质上是为了大量的数据能横跨成百上千台机器，但是你看到的是一个文件系统而不是很多文件系统。比如你说我要获取/hdfs/tmp/file1的数据，你引用的是一个文件路径，但是实际的数据存放在很多不同的机器上。你作为用户，不需要知道这些，就好比在单机上你不关心文件分散在什么磁道什么扇区一样。HDFS为你管理这些数据。

**存的下数据之后，你就开始考虑怎么处理数据。**虽然HDFS可以为你整体管理不同机器上的数据，但是这些数据太大了。一台机器读取成T上P的数据（很大的数据哦，比如整个东京热有史以来所有高清电影的大小甚至更大），一台机器慢慢跑也许需要好几天甚至好几周。对于很多公司来说，单机处理是不可忍受的，比如微博要更新24小时热博，它必须在24小时之内跑完这些处理。那么我如果要用很多台机器处理，我就面临了如何分配工作，如果一台机器挂了如何重新启动相应的任务，机器之间如何互相通信交换数据以完成复杂的计算等等。这就是MapReduce / Tez / Spark的功能。MapReduce是第一代计算引擎，Tez和Spark是第二代。MapReduce的设计，采用了很简化的计算模型，只有Map和Reduce两个计算过程（中间用Shuffle串联），用这个模型，已经可以处理大数据领域很大一部分问题了。

那什么是Map什么是Reduce？考虑如果你要统计一个巨大的文本文件存储在类似HDFS上，你想要知道这个文本里各个词的出现频率。你启动了一个MapReduce程序。Map阶段，几百台机器同时读取这个文件的各个部分，分别把各自读到的部分分别统计出词频，产生类似（hello, 12100次），（world，15214次）等等这样的Pair（我这里把Map和Combine放在一起说以便简化）；这几百台机器各自都产生了如上的集合，然后又有几百台机器启动Reduce处理。Reducer机器A将从Mapper机器收到所有以A开头的统计结果，机器B将收到B开头的词汇统计结果（当然实际上不会真的以字母开头做依据，而是用函数产生Hash值以避免数据串化。因为类似X开头的词肯定比其他要少得多，而你不希望数据处理各个机器的工作量相差悬殊）。然后这些Reducer将再次汇总，（hello，12100）＋（hello，12311）＋（hello，345881）= （hello，370292）。每个Reducer都如上处理，你就得到了整个文件的词频结果。这看似是个很简单的模型，但很多算法都可以用这个模型描述了。Map＋Reduce的简单模型很黄很暴力，虽然好用，但是很笨重。第二代的Tez和Spark除了内存Cache之类的新feature，本质上来说，是让Map/Reduce模型更通用，让Map和Reduce之间的界限更模糊，数据交换更灵活，更少的磁盘读写，以便更方便地描述复杂算法，取得更高的吞吐量。

有了MapReduce，Tez和Spark之后，程序员发现，MapReduce的程序写起来真麻烦。他们希望简化这个过程。这就好比你有了汇编语言，虽然你几乎什么都能干了，但是你还是觉得繁琐。你希望有个**更高层更抽象的语言层来描述算法和数据处理流程**。于是就有了Pig和Hive。Pig是接近**脚本方式**去描述MapReduce，Hive则用的是**SQL**。它们把脚本和SQL语言翻译成MapReduce程序，丢给计算引擎去计算，而你就从繁琐的MapReduce程序中解脱出来，用更简单更直观的语言去写程序了。

**有了Hive之后，人们发现SQL对比Java有巨大的优势。**一个是它太容易写了。刚才词频的东西，用SQL描述就只有一两行，MapReduce写起来大约要几十上百行。而更重要的是，非计算机背景的用户终于感受到了爱：我也会写SQL！于是数据分析人员终于从乞求工程师帮忙的窘境解脱出来，工程师也从写奇怪的一次性的处理程序中解脱出来。大家都开心了。Hive逐渐成长成了大数据仓库的核心组件。甚至很多公司的流水线作业集完全是用SQL描述，因为易写易改，一看就懂，容易维护。

自从数据分析人员开始用Hive分析数据之后，它们发现，Hive在MapReduce上跑，真几把慢！流水线作业集也许没啥关系，比如24小时更新的推荐，反正24小时内跑完就算了。但是数据分析，人们总是希望能跑更快一些。比如我希望看过去一个小时内多少人在充气娃娃页面驻足，分别停留了多久，对于一个巨型网站海量数据下，这个处理过程也许要花几十分钟甚至很多小时。而这个分析也许只是你万里长征的第一步，你还要看多少人浏览了跳蛋多少人看了拉赫曼尼诺夫的CD，以便跟老板汇报，我们的用户是猥琐男闷骚女更多还是文艺青年／少女更多。你无法忍受等待的折磨，只能跟帅帅的工程师蝈蝈说，快，快，再快一点！

于是Impala，Presto，Drill(交互式查询引擎)诞生了（当然还有无数非著名的交互SQL引擎，就不一一列举了）。三个系统的核心理念是，MapReduce引擎太慢，因为它太通用，太强壮，太保守，我们SQL需要更轻量，更激进地获取资源，更专门地对SQL做优化，而且不需要那么多容错性保证（因为系统出错了大不了重新启动任务，如果整个处理时间更短的话，比如几分钟之内）。这些系统让用户更快速地处理SQL任务，牺牲了通用性稳定性等特性。如果说MapReduce是大砍刀，砍啥都不怕，那上面三个就是剔骨刀，灵巧锋利，但是不能搞太大太硬的东西。

这些系统，说实话，一直没有达到人们期望的流行度。因为这时候又两个异类被造出来了。他们是Hive on Tez / Spark和SparkSQL。它们的设计理念是，MapReduce慢，但是如果我用新一代通用计算引擎Tez或者Spark来跑SQL，那我就能跑的更快。而且用户不需要维护两套系统。这就好比如果你厨房小，人又懒，对吃的精细程度要求有限，那你可以买个电饭煲，能蒸能煲能烧，省了好多厨具。

上面的介绍，基本就是一个数据仓库的构架了。底层HDFS，上面跑MapReduce／Tez／Spark，在上面跑Hive，Pig。或者HDFS上直接跑Impala，Drill，Presto。这解决了中低速数据处理的要求。那如果我要更高速的处理呢？如果我是一个类似微博的公司，我希望显示不是24小时热博，我想看一个不断变化的热播榜，更新延迟在一分钟之内，上面的手段都将无法胜任。于是又一种计算模型被开发出来，这就是Streaming（流）计算。Storm是最流行的流计算平台。流计算的思路是，如果要达到更实时的更新，我何不在数据流进来的时候就处理了？比如还是词频统计的例子，我的数据流是一个一个的词，我就让他们一边流过我就一边开始统计了。流计算很牛逼，基本无延迟，但是它的短处是，不灵活，你想要统计的东西必须预先知道，毕竟数据流过就没了，你没算的东西就无法补算了。因此它是个很好的东西，但是无法替代上面数据仓库和批处理系统。

还有一个有些独立的模块是KV Store，比如Cassandra，HBase，MongoDB以及很多很多很多很多其他的（多到无法想象）。所以KV Store就是说，我有一堆键值，我能很快速滴获取与这个Key绑定的数据。比如我用身份证号，能取到你的身份数据。这个动作用MapReduce也能完成，但是很可能要扫描整个数据集。而KV Store专用来处理这个操作，所有存和取都专门为此优化了。从几个P的数据中查找一个身份证号，也许只要零点几秒。这让大数据公司的一些专门操作被大大优化了。比如我网页上有个根据订单号查找订单内容的页面，而整个网站的订单数量无法单机数据库存储，我就会考虑用KV Store来存。KV Store的理念是，基本无法处理复杂的计算，大多没法JOIN，也许没法聚合，没有强一致性保证（不同数据分布在不同机器上，你每次读取也许会读到不同的结果，也无法处理类似银行转账那样的强一致性要求的操作）。但是丫就是快。极快。

每个不同的KV Store设计都有不同取舍，有些更快，有些容量更高，有些可以支持更复杂的操作。必有一款适合你。

除此之外，还有一些更特制的系统／组件，比如Mahout是分布式机器学习库，Protobuf是数据交换的编码和库，ZooKeeper是高一致性的分布存取协同系统，等等。

有了这么多乱七八糟的工具，都在同一个集群上运转，大家需要互相尊重有序工作。所以另外一个重要组件是，调度系统。现在最流行的是Yarn。你可以把他看作中央管理，好比你妈在厨房监工，哎，你妹妹切菜切完了，你可以把刀拿去杀鸡了。只要大家都服从你妈分配，那大家都能愉快滴烧菜。

你可以认为，大数据生态圈就是一个厨房工具生态圈。为了做不同的菜，中国菜，日本菜，法国菜，你需要各种不同的工具。而且客人的需求正在复杂化，你的厨具不断被发明，也没有一个万用的厨具可以处理所有情况，因此它会变的越来越复杂。

# Spark Streaming和Storm 区别

* 从计算模型上：Hadoop是批处理型，Spark Streaming和Storm是流处理型
* 从处理时延上：Hadoop数分钟-数小时，Spark Streaming秒级，Storm毫秒级
* 从吞吐量上：Hadoop>Spark Streaming >Storm
* 从容错性上：Hadoop>Storm>Spark Streaming

对于Hadoop来说，Spark Streaming和Storm能做的它都能做，但就是处理时延大，如果处理较大文件且不需要实时性必然选择Hadoop。

如果处理的都是一些小文件且需要需要一定实时性选择Spark Streaming（准实时，对一个时间段内的数据收集起来，作为一个RDD，再处理）。

如果需要非常强的实时性选择Storm（纯实时，来一条数据，处理一条数据，常见消息来源MQ）。

## 对于Storm来说

1、建议在那种需要纯实时，不能忍受1秒以上延迟的场景下使用，比如实时金融系统，要求纯实时进行金融交易和分析

2、此外，如果对于实时计算的功能中，要求可靠的事务机制和可靠性机制，即数据的处理完全精准，一条也不能多，一条也不能少，也可以考虑使用Storm

3、如果还需要针对高峰低峰时间段，动态调整实时计算程序的并行度，以最大限度利用集群资源（通常是在小型公司，集群资源紧张的情况），也可以考虑用Storm

4、如果一个大数据应用系统，它就是纯粹的实时计算，不需要在中间执行SQL交互式查询、复杂的transformation算子等，那么用Storm是比较好的选择

## 对于Spark Streaming来说

1、如果对上述适用于Storm的三点，一条都不满足的实时场景，即，不要求纯实时，不要求强大可靠的事务机制，不要求动态调整并行度，那么可以考虑使用Spark Streaming

2、考虑使用Spark Streaming最主要的一个因素，应该是针对整个项目进行宏观的考虑，即，如果一个项目除了实时计算之外，还包括了离线批处理、交互式查询等业务功能，而且实时计算中，可能还会牵扯到高延迟批处理、交互式查询等功能，那么就应该首选Spark生态，用Spark Core开发离线批处理，用Spark SQL开发交互式查询，用Spark Streaming开发实时计算，三者可以无缝整合，给系统提供非常高的可扩展性

## Spark Streaming与Storm的优劣分析

事实上，Spark Streaming绝对谈不上比Storm优秀。这两个框架在实时计算领域中，都很优秀，只是擅长的细分场景并不相同。

Spark Streaming仅仅在吞吐量上比Storm要优秀。但是问题是，是不是在所有的实时计算场景下，都那么注重吞吐量？不尽然。因此，通过吞吐量说Spark Streaming强于Storm，不靠谱。

事实上，Storm在实时延迟度上，比Spark Streaming就好多了，前者是纯实时，后者是准实时。而且，Storm的事务机制、健壮性 / 容错性、动态调整并行度等特性，都要比Spark Streaming更加优秀。

Spark Streaming还有一点是Storm绝对比不上的，就是：它位于Spark生态技术栈中，因此Spark Streaming可以和Spark Core、Spark SQL无缝整合，也就意味着，我们可以对实时处理出来的中间数据，立即在程序中无缝进行延迟批处理、交互式查询等操作。这个特点大大增强了Spark Streaming的优势和功能。

# 问题