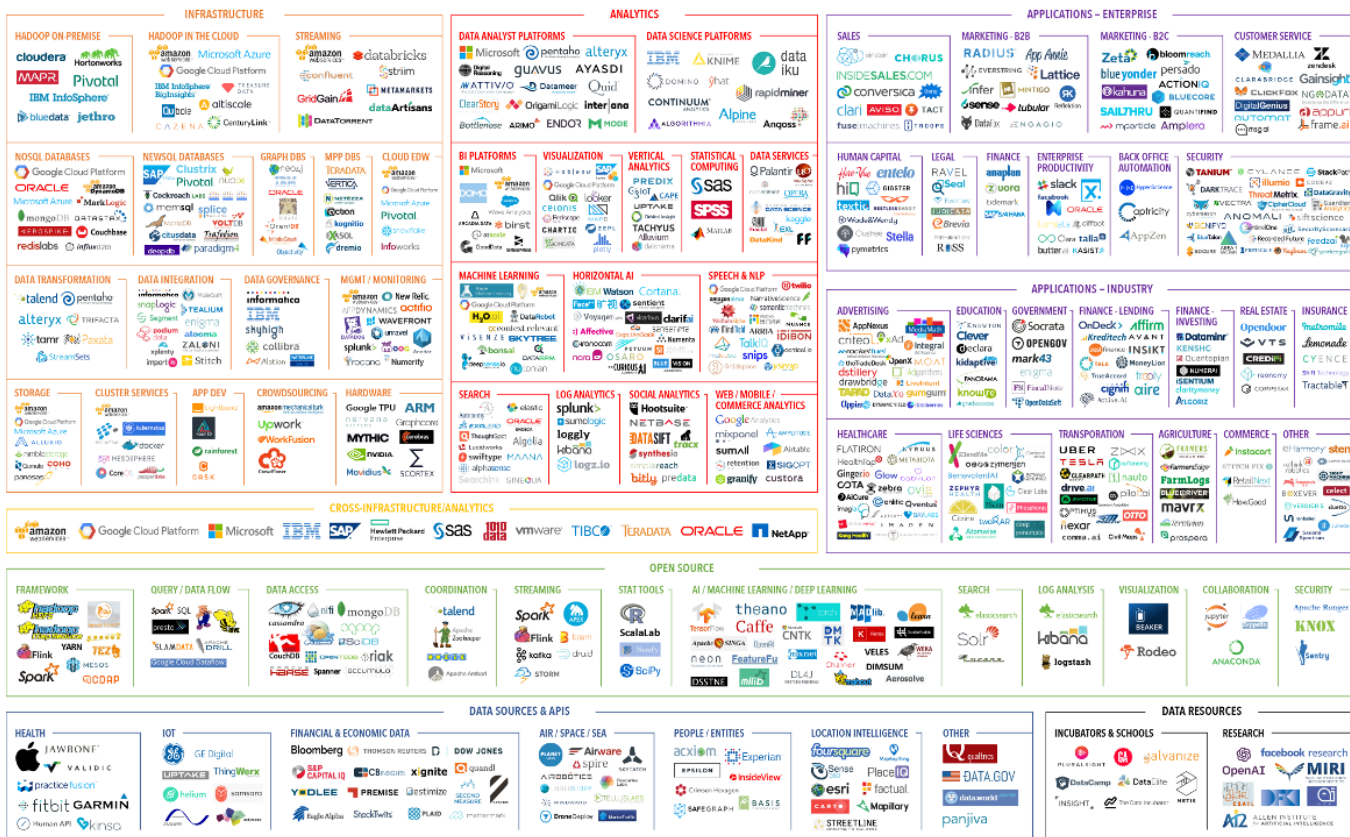


## BIG DATA LANDSCAPE 2017



Last updated 4/5/2017

© Matt Turck (@mattturck), Jim Hao (@jimrhao), & FirstMark (@firstmarkcap) [mattturck.com/bigdata2017](http://mattturck.com/bigdata2017)

**FIRSTMARK**   
EARLY-STAGE VENTURE CAPITAL

#### EARLY STAGE VENTURE CAPITAL

# 目录

前言.....	5
1. 大数据生态圈.....	6
1.1. 为什么诞生了大数据技术.....	6
1.1.1. Hadoop 的诞生.....	6
1.2. 我们该如何处理大的数据量.....	7
1.2.1. 分布式和集群的区别.....	7
1.2.2. 消息的处理机制.....	8
1.3. 大数据处理的方式.....	8
1.4. Hadoop.....	8
1.5. Flume.....	9
1.6. Hive/pig.....	10
1.7. Hbase.....	11
1.8. Sqoop.....	11
1.9. Kafka.....	11
1.10. Zookeeper.....	12
1.11. 实时流处理.....	12
1.11.1. Storm.....	12
1.11.2. Spark.....	13
1.11.3. Flink.....	13
1.11.4. 比较和使用场景.....	13
1.12. Oozie/Azkaban.....	14
2. 大数据基础.....	14
2.1. 计算机科学与技术.....	14
2.2. 大数据（分布式系统）.....	16

2.2.1.	什么是大数据 .....	16
2.2.2.	大数据的特征 4V .....	16
2.2.3.	从技术上讲 .....	17
2.2.4.	大数据在生活中的应用 .....	18
2.2.5.	数据 .....	18
2.2.6.	技术架构 .....	18
2.3.	开源&Apache 软件基金会 .....	22
2.3.1.	开源 .....	22
2.3.2.	Apache 基金会 .....	22
2.3.3.	怎样学习一个 Apache 开源项目？ .....	23
2.4.	分布式系统原理 .....	23
2.4.1.	核心思想：分而治之 .....	23
2.4.2.	数据分区 .....	24
2.4.3.	分区算法 .....	24
2.4.4.	容错 .....	25
2.4.5.	缩容扩容 .....	25
3.	探索 .....	26

## 前言

本资料共分三部分，大体上介绍了大数据的一些概念和理论以及大数据生态圈的一些框架和平台，试图在有限的时间内让大家对大数据有一个整体的认识，并能为大家继续探索部分或根据自己感兴趣的点查资料拓展。此份资料也算是抛砖引玉（并且希望如此），希望能对大家深入了解大数据有所帮助！

由于时间有限，从昨天开始到今天只有一天的准备，资料中难免会有遗漏和错误，也希望使用者能 Tolerant，也可以给我提出您的宝贵意见，根据后期情况如果大家确实有用，后期有时间我会进行修复和完善。同时也非常感谢大家的支持。

yore Yuan 于北京.东大桥  
2018-06

# 1. 大数据生态圈

## 1.1. 为什么诞生了大数据技术

全球互联网上的量每两年会翻一番。截止到 全球已步入大数据时代，互联网上的量每两年会翻一番。截止到

2013 年，全球数据量为 年，全球数据量为 4.3 泽字节，2020 年有望达到 40 泽。注：摘自一大数据标准化白皮书

(2018)。这份文档有点枯燥，不过可以看下 p47--大数据参考架构图；p79--附件：成功案例

数据量增大 (4V)，但是对处理速度上的要求并没有降低，这些导致单台计算机无法再业务需求的时间内给出处理结果，所以需要针对大批量的数据处理给出一套解决方案，比较早的是在数据检索方面。

### 1.1.1. Hadoop 的诞生

说起 Hadoop 的诞生，我一般会想到几个关键词：

#### Google 大数据三篇著名论文

The Google File System(2003) -> **HDFS**

MapReduce: Simplified Data Processing on Large Clusters(2004)-> **MR**

Bigtable: A Distributed Storage System for Structured Data(2006)-> **HBase**

#### Doug Cutting(道格.卡丁 Hadoop 之父)

1985 年,Cutting 毕业于美国斯坦福大学。他并不是一开始就决心投身 IT 行业的，在大学时代的头两年，Cutting 学习了诸如物理、地理等常规课程。因为学费的压力，Cutting 开始意识到，自己必须学习一些更加实用、有趣的技能。这样，一方面可以帮助自己还清贷款，另一方面，也是为自己未来的生活做打算。因为斯坦福大学座落在 IT 行业的“圣地”硅谷，所以学习软件对年轻人来说是再自然不过的事情了

Cutting 的第一份工作是在 Xerox 做实习生，Xerox 当时的激光扫描仪上开发屏幕保护程序，由于这套程序是基于系统底层开发的，所以 其他同事可以给这个程序添加不同的主题。这份工作给了 Cutting 一定的满足感，也是他最早的“平台”级的作品。Xerox 对 Cutting 后来研究搜索技术起到了决定性的影响，除了短暂的在苏格兰工作的经历外，Cutting 事业的起步阶段大部分都是在 Xerox 度过的，这段 时间让他在搜索技术的知识上有了很大提高。他花了四年的时间搞研发，这四年中，他阅读了大量的论文，同时，自己也发表了很多论文，用 Cutting 自己的 话说——“我的研究生是在 Xerox 读的。”

尽管 Xerox 让 Cutting 积累了不少技术知识，但他却认为，自己当时搞的这些研究只是纸上谈兵，没有人试验过这些理论的可实践性。于是，他决定勇敢地迈出这一步，让



Hadoop之父Doug Cutting

搜索技术可以为更多人所用。1997 年底，Cutting 开始以每周两天的时间投入，在家里试着用 Java 把这个想法变成现实，不久之后，**Lucene** 诞生了。作为第一个提供全文文本搜索的开源函数库，Lucene 的伟大自不必多言。

之后，Cutting 再接再厉，在 Lucene 的基础上将开源的思想继续深化。2004 年，Cutting 和同为程序员出身的 Mike Cafarella 决定开发一款可以代替当时的主流搜索产品的开源搜索引擎，这个项目被命名为 **Nutch**。在此之前，Cutting 所在的公司 Architext（其主要产品为 Excite 搜索引擎）因没有顶住互联网经济泡沫的冲击而破产，那时的 Cutting 正处在 **Freelancer** 的生涯中，所以他希望自己的项目能通过一种低开销的方式来构建网页中的大量算法。幸运的是，Google 这时正好发布了一项研究报告，就是上面的前两篇论文，根据这两篇论文，他们的 Nutch 项目很快的得到了进展，并最终以 **Hadoop** 的名字发布（这个名字是我孩子给一个棕黄色的大象玩具命名的。我的命名标准就是简短，容易发音和拼写，没有太多的意义，并且不会被用于别处。小孩子恰恰是这方面的高手）。

Google 与 Cutting 的小趣闻...

也可参阅《Hadoop 权威指南 4》1.6 Apache Hadoop 发展简史

## 1.2. 我们该如何处理大的数据量

就比如现实中有有一个场景，某个任务，起初可能一个人处理完全没压力，并且还能按时完成任务，但是随着这个量和难度的不断增大，再让一个人处理开始出现了问题，比如完成的质量会下降、完成的期限会延长甚至至是拖垮。

那现实中一个人处理不了这个任务了，这时肯定会找多个人一起做这个任务，这样每个人就可以分开完成各自的任务，最终把整个任务按时完成。

解决大数据的问题也是同样，一台机器无法处理的时候，就找一群机器来处理，甚至是大规模的集群，但是多台机器去处理同一个任务就会出现很多问题，有哪些呢？

- ①多节点执行 Task 和单节点执行 Task 有什么不同？
- ②任务如何分配(资源调度)
- ③怎么知道任务执行完毕（消息处理机制）
- ④如果某个节点挂了，任务如何处理(Failover and fencing)
- ⑤处理的结果保存到何处(分布式存储)

### 1.2.1. 分布式和集群的区别

主要的区别就是，分布式是通过缩短单个任务的执行时间来提升效率的；集群是通过提高单位时间内执行的任务数来提升效率的。



## 1.2.2. 消息的处理机制

(1) at most once:最多一次

(2) at least once:至少一次

(3) exactly once:恰好一次

## 1.3. 大数据处理的方式

离线处理（批处理）

实时处理（流处理）

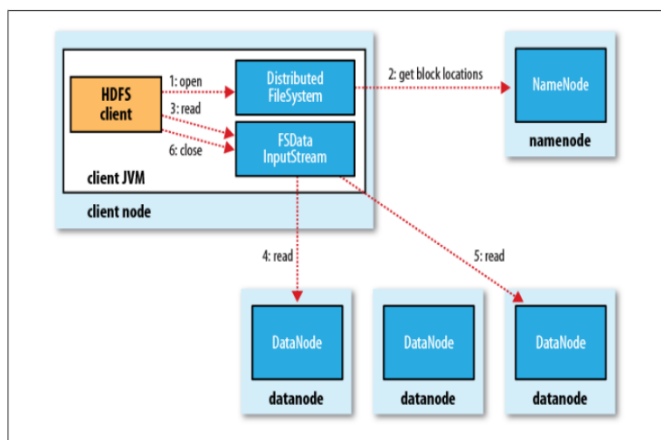
## 1.4. Hadoop

Java 92.7%   C++ 3.1%   C 1.8%   JavaScript 1.2%   Shell 0.5%   HTML 0.4%   Other 0.3%

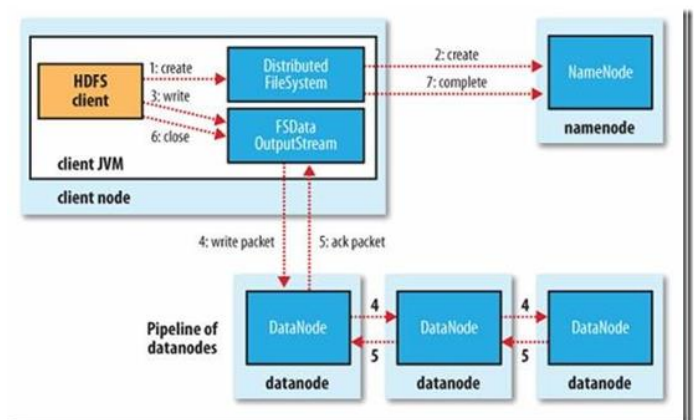
HDFS

Hadoop 的分布式文件系统

《HDFS权威指南》图解HDFS写过程

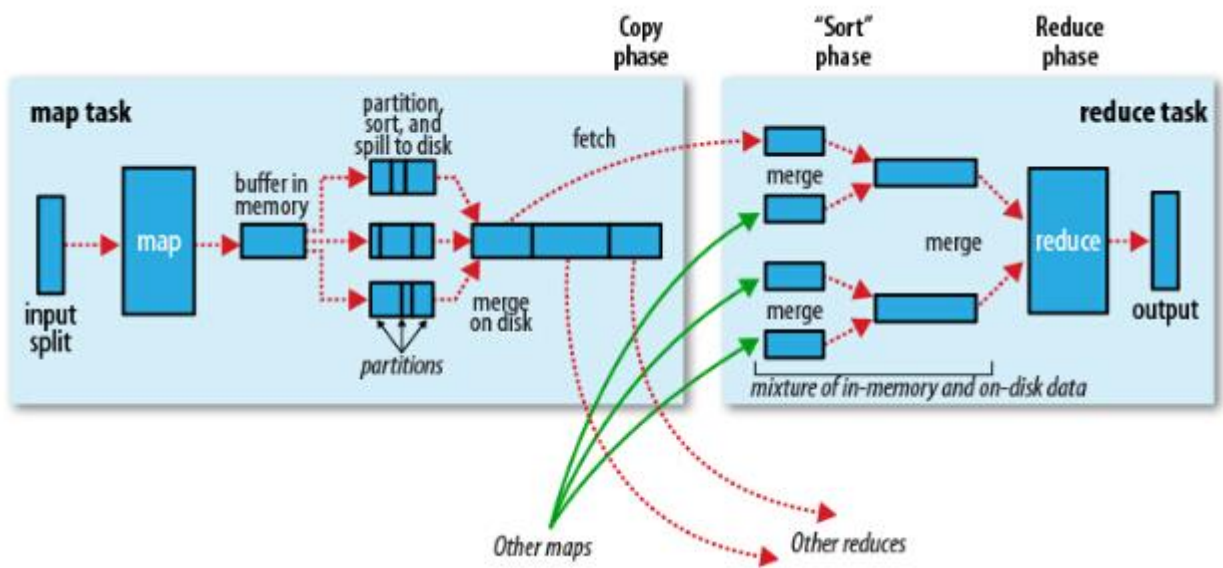


《HDFS权威指南》图解HDFS读过程



MapReduce



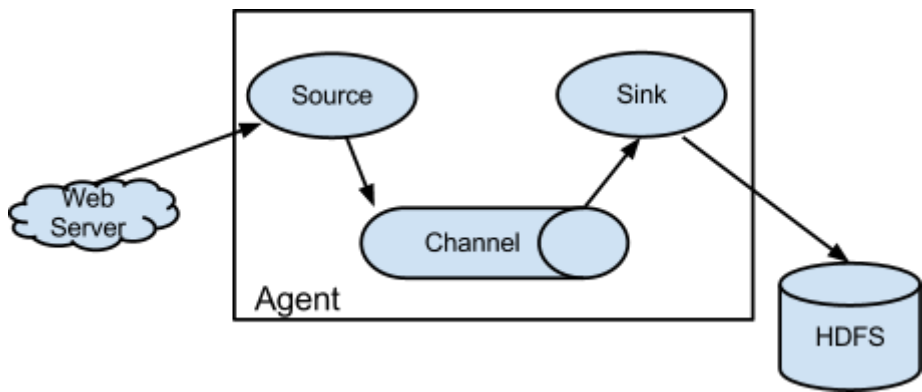


YARN  
资源调度  
图见 2.4.1 核心思想：分而治之小节图

1.5. Flume



flume 是一个分布式、可靠、和高可用的海量日志采集、聚合和传输的系统。支持在日志系统中定制各类数据发送方，用于收集数据;同时，Flume 提供对数据进行简单处理，并写到各种数据接受方(比如文本、HDFS、



Hbase 等)的能力。

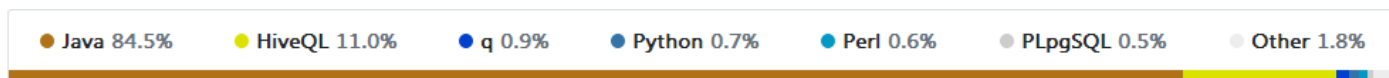
Agent	一个 Agent 包含 source ， channel, sink 和其他组件。
Source	Source 负责接收 event 或通过特殊机制产生 event，并将 events 批量的放到一个或多个 Channel
Channel	Channel 位于 Source 和 Sink 之间，用于缓存进来的 event

Sink	Sink 负责将 event 传输到下一个 source 或最终目的地，成功后将 event 从 channel 移除
Client	Client 是一个将原始 log 包装成 events 并且发送他们到一个或多个 agent 的实体，目的是从数据源系统中解耦 Flume，在 flume 的拓扑结构中不是必须的。
Events	Event 是 Flume 数据传输的基本单元。可以是日志记录、 avro 对象等。

Flume 主要由 3 个重要的组件购成：

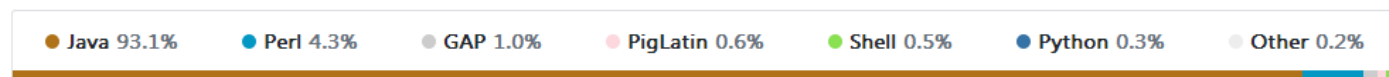
- Source:完成对日志数据的收集，分成 transtion 和 event 打入到 channel 之中。
- Channel:主要提供一个队列的功能，对 source 提供中的数据进行简单的缓存。
- Sink:取出 Channel 中的数据，进行相应的存储文件系统，数据库，或者提交到远程服务器。

## 1.6. Hive/pig



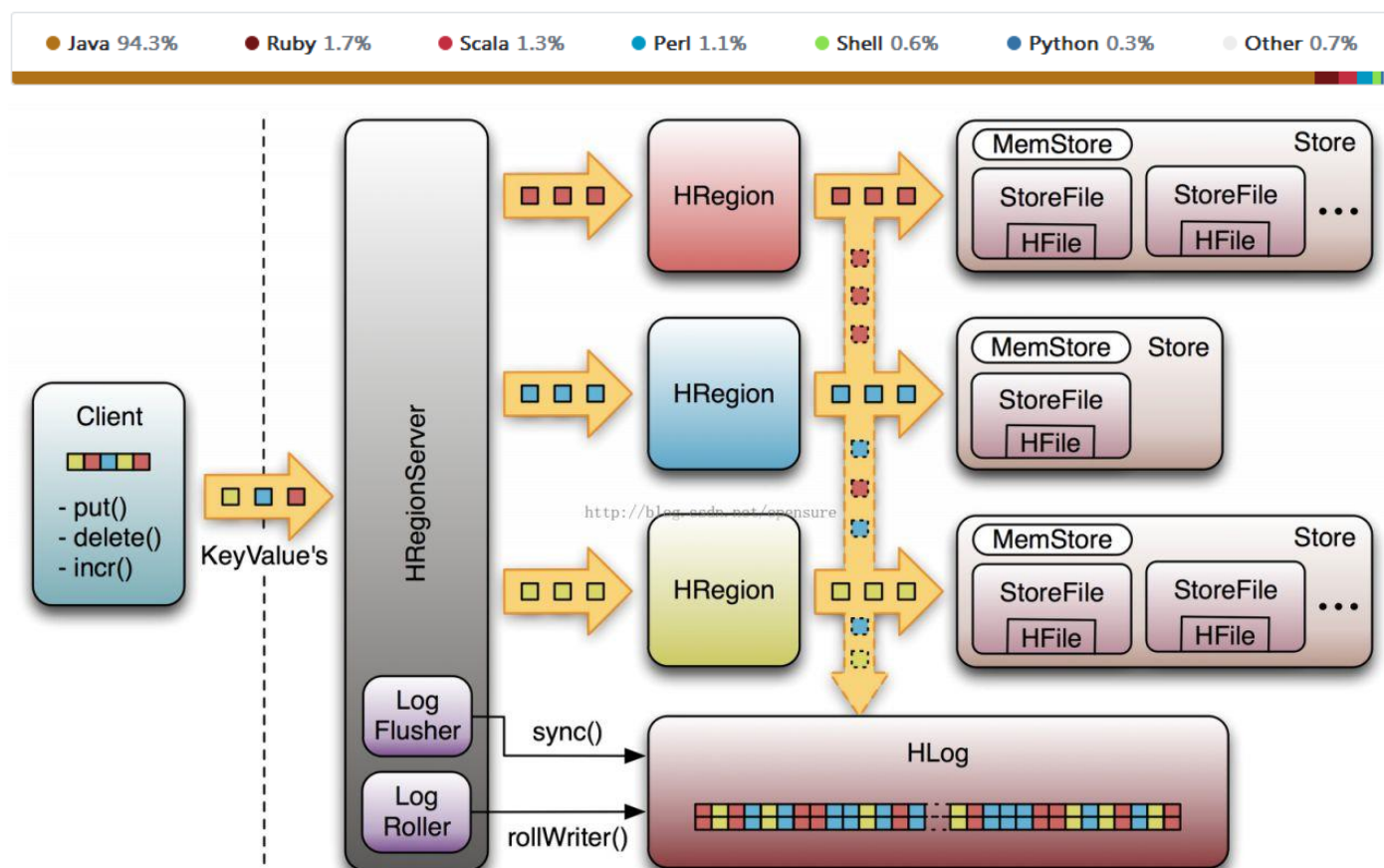
Hive 最早是由 Fackbook 开发 ,Hive 是 Hadoop 生态系统中必不可少的一个工具 ,他提供了一种 SQL 方言，可以查询存储在 Hadoop 分布式文件系统的数据或其他和 Hadoop 集成的文件系统和像 HBase 和 Cassandra 这样的数据库中的数据。Hive 降低了将 SQL 应用程序转移到 Hadoop 系统上的难度。

Hive 不支持 OLTP（联机事务处理）所需的关键功能，更接近成为一个 OLAP（联机事务分析）工具。



Pig 最早是和 Hive 同期由 Yahoo 开发完成。Pig 被描述成一种数据流语言，而不是一种查询语言。在 Pig 中，用户需要写一系列的声明语句来定义某些关系和其他一些关系之间的联系，这里每一个新的关系都会执行新的数据转换过程。Pig 常用语 ETL（数据抽取，转换和装载）过程的一部分。

## 1.7. Hbase



## 1.8. Sqoop



Sqoop 是一款开源的工具,主要用于在 Hadoop(Hive)与传统的数据库(mysql、postgresql...)间进行数据的传递,可以将一个关系型数据库(例如:MySQL,Oracle,Postgres等)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。

## 1.9. Kafka



Kafka 是一个分布式的流式数据平台

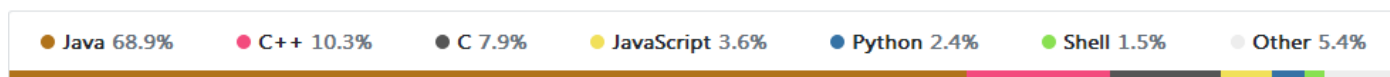
发布和订阅:像消息传递系统一样读写数据流;



处理：编写实时响应时间的可伸缩流处理应用程序；

存储：将数据安全的存储在分布式多副本容错行的集群中

## 1. 10. Zookeeper



ZooKeeper 是一种为分布式应用所设计的高可用、高性能且一致的开源协调服务，它提供了一项基本服务：分布式锁服务。由于 ZooKeeper 的开源特性，后来我们的开发者在分布式锁的基础上，摸索出了其他的使用方法：配置维护、组服务、分布式消息队列、分布式通知/协调等。

如果有一个分布式系统，其中一台机器挂载了一个资源，其他三个物理分布的进程都竞争这个资源，但我们又不希望他们同时对这个资源进行访问，这时我们就需要一个协调器。这个协调器就是一个锁。在分布式环境下的这个锁叫做分布式锁。

在一台机器上是比较简单的，如果服务调度成功就是成功，调度室发生异常就是调度失败。但是在分布式环境中，由于网络的不可靠，你对一个服务的调用失败了并不表示一定是失败的，可能是执行成功了，但是响应返回的时候失败了。

目前，在分布式协调技术方面做得比较好的就是 Google 的 Chubby 还有 Apache 的 ZooKeeper 他们都是分布式锁的实现者。

## 1. 11. 实时流处理

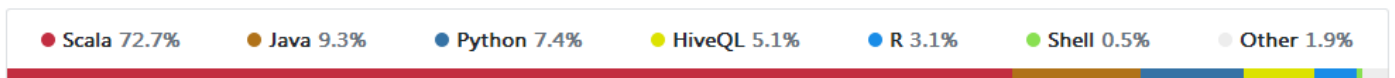
### 1. 11. 1. Storm



Apache Storm is a free and open source distributed realtime computation system. Storm makes it easy to reliably process unbounded streams of data, doing for realtime processing what Hadoop did for batch processing. Storm is simple, can be used with any programming language, and is a lot of fun to use!

Storm has many use cases: realtime analytics, online machine learning, continuous computation, distributed RPC, ETL, and more. Storm is fast: a benchmark clocked it at over a million tuples processed per second per node. It is scalable, fault-tolerant, guarantees your data will be processed, and is easy to set up and operate.

## 1. 11. 2. Spark



Apache Spark™是用于大规模数据处理的统一分析引擎。

## 1. 11. 3. Flink



Apache Flink® is an open-source stream processing framework for distributed, high-performing, always-available, and accurate data streaming applications.

Apache Flink 是一个开源的分布式、高性能、始终可用的真实时流处理框架。Flink 的核心是在数据流上提供数据分发、通信、具备容错的分布式计算。同时，Flink 在流处理引擎上构建了批处理引擎，原生支持了迭代计算、内存管理和程序优化。

## 1. 11. 4. 比较和使用场景

storm

纯实时，来一条数据，处理一条数据  
毫秒级  
吞吐量，低  
事务机制，支持且完善  
健壮性/容错性 ZK, Acker,非常强  
动态调整并行度 支持

SparkStreaming

准实时，对一个时段内的数据处理  
秒级  
吞吐量，高  
事务机制，支持但不完善  
健壮性/容错性 Checkpoint, WAL,一般

动态调整并行度 不支持

### Storm 适用场景

- ①在做实时计算时如果我们不能忍受 1 秒以上的延迟比如金融系统，则选择使用 storm
- ②对事务机制和可靠性要求较高的，比如数据处理要完全精确，一条也不能多，一条也不能少，选择 storm
- ③在高峰和低峰时，需要动态调整计算程序的并行度的，可以考虑使用 storm
- ④如果我们的项目是一个纯实时计算框架，中间不需要执行 SQL、复杂的 transformation 算子，选择 Storm

### Spark 应用场景

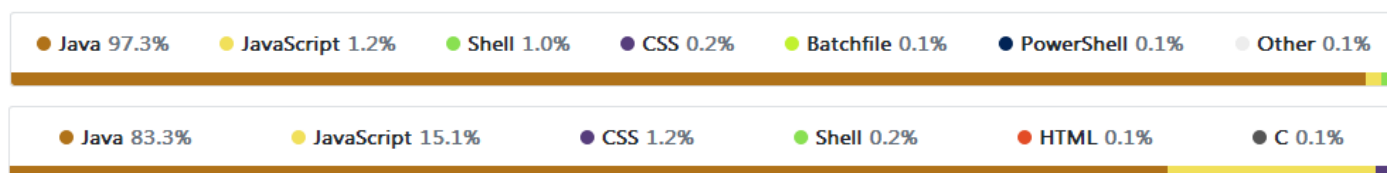
①上面的几点都不适用的，实时性要求不是很高，不要求强大的事务机制，不要求动态调整并行度，可以考虑使用

②整个项目整体考虑上有实时也有离线批处理、交互式查询等业务的，可以使用 spark(spark Corek 开发离线批处理、

Spark SQL 开发交互式查询、Spark Streaming 开发实时计算)

## 1. 12. Oozie/Azkaban

GitHub 上 Oozie 和 Azkaban 的开发语言占比分别如下两图



较流行的两种 Hadoop 工作流引擎调度器 Azkaban 与 Oozie。Apache Oozie 配置工作流的过程是编写大量的 XML 配置，而且代码复杂度比较高，不易于二次开发。Oozie 相比 Azkaban 是一个重量级的任务调度系统，功能全面，但配置使用也更复杂。如果我们不在意某些功能的缺失，轻量级调度器 Azkaban 是很不错的候选对象。

## 2. 大数据基础

### 2. 1. 计算机科学与技术

掌握本质。以不变应万变。

- 计算机组成原理
- 计算机操作系统
- 编译原理
- 程序设计语言 ( C、C++、Java、C#、PHP、JavaScript、Python、Scala )

集合，NIO，多线程，类加载，GC，内存管理

- 计算机网络

MQTT，HTTP，FTP

- 数据结构与算法
- 数据库原理
- 软件工程

需求，设计，开发，测试，上线，维护

- More：数学（线性代数、概率与统计、离散数学）&英语

计算机存储单位换算表

英文单位	英文全拼	中文单位	中文解释	换算
1B	Byte	字节		8bit
1KB	Kilobyte	千字节		1024B
1MB	Mega byte	兆字节	简称“兆”	1024KB
1GB	Giga byte	吉字节	又称“千兆”	1024MB
1TB	Tera byte	太字节	万亿字节	1024GB
1PB	Peta byte	拍字节	千万亿字节	1024TB
1EB	Exa byte	艾字节	百亿亿字节	1024PB
1ZB	Zetta byte	泽字节	十万亿亿字节	1024 EB
1YB	Yotta byte	尧字节	一亿亿亿字节	1024 ZB
1BB	Bronto byte		一千亿亿亿字节	1024 YB
1NB	Nona byte			1024BB
1DB	Dogga byte			1024NB



## 2.2. 大数据（分布式系统）

### 2.2.1. 什么是大数据

美国著名咨询公司麦肯锡在《大数据的下一个前沿：创新、竞争和生产力》中给出的大数据的定义是：大数据是指大小超出了典型数据库软件工具收集、存储、管理和分析能力的数据集。

美国国籍标准技术研究所大数据局工作组在《大数据：定义和分类》中认为：大数据是指那些传统数据架构无法有效地处理的新数据集。因此，采用新的架构来高效率完成数据处理，这些数据集特征包括：容量（Volume）、数据类型的多样性（Variety）、多个领域数据的差异性、数据的动态特征（数据或流动率，可变性）。

维基百科给出的定义是：大数据，或称巨量数据、海量数据、大数据，指的是所涉及的数据量规模巨大到无法通过人工在河里的时间内达到截取、管理、处理、并整理成为人类所能解读的信息。

国内普遍的理解是：**具有数据量巨大、来源多样、生成极快、且多变特征并且难以用传统数据体系结构有效处理的包含大量数据集的数据。**

### 2.2.2. 大数据的特征 4V

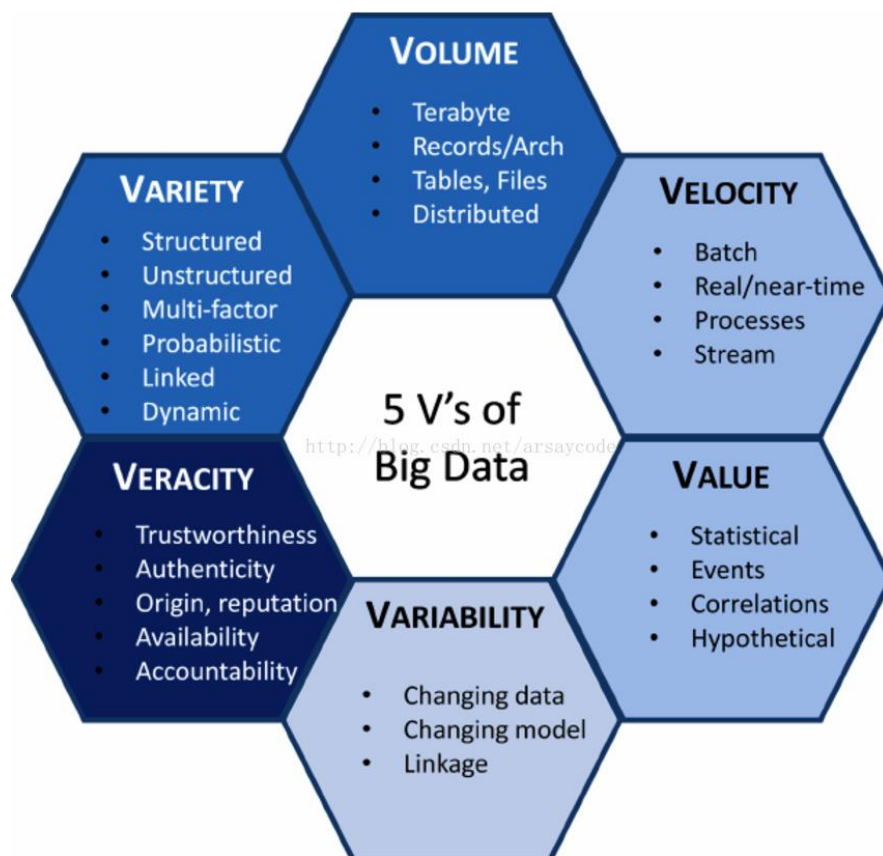
Value, Volume, Variety, Velocity 价值、容量、多样、速度

①数据价值（Value）：数据价值密度相对较低，或者说是浪里淘沙却又弥足珍贵。

②数据体量巨大（Volume）：数据量大，包括采集、存储和计算的量都非常大。

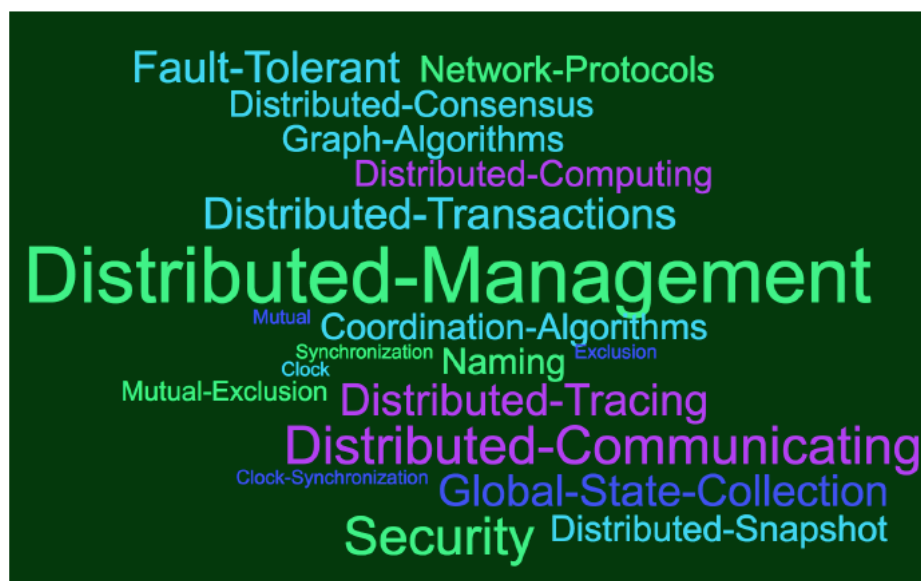
③数据类型繁多（Variety）：种类和来源多样化。除了结构化数据(数据库)外，大数据还包含各种非结构化数据，例如文本、音频、视频、点击流量、文本记录等，以及半结构化数据，例如电子邮件，办公处理文档等。

④处理速度快 (Velocity)：数据增长速度快，处理速度也快，时效性要求高。比如搜索引擎要求几分钟前的新闻能够被用户查询到，个性化推荐算法尽可能要求实时完成推荐。



### 2.2.3. 从技术上讲

从技术的角度讲，大数据就是指分布式相关的技术。



Distributed Systems  
Domain

© 2017 Alibaba Middleware Group

## 2.2.4. 大数据在生活中的应用

生活中大数据的应用：电商/资讯推荐、反电信欺诈、个人信用画像、共享单车跟踪

## 2.2.5. 数据

数据源：

按行业，互联网、金融、电信等；

按系统，上网日志、业务应用日志、交易记录等。

数据格式：

文本，二进制。文本有 JSON、XML 系、CSV 等。

数据编码：

Unicode、GBK、UTF-8 等。

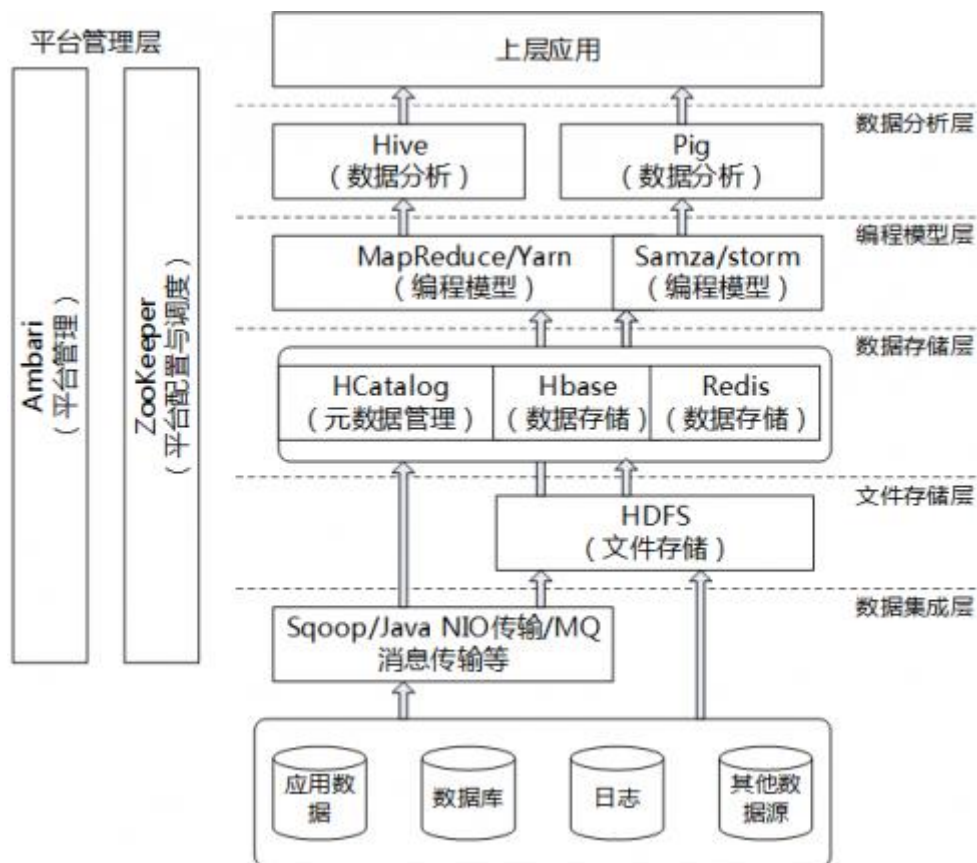
数据压缩：

Zip、Snappy 等。

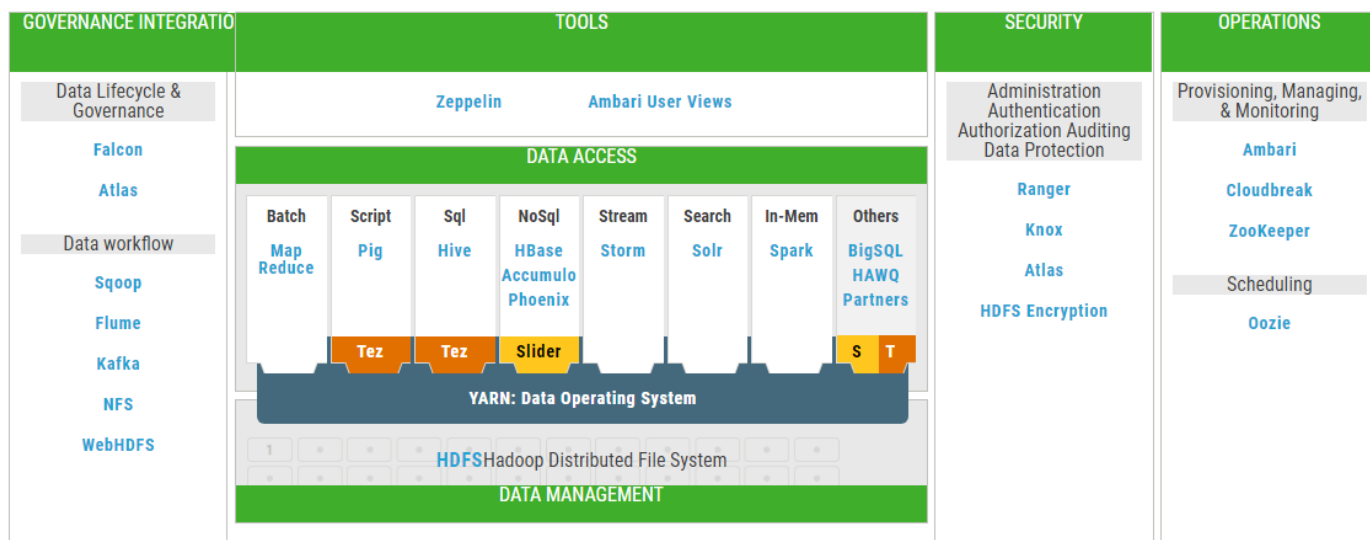
## 2.2.6. 技术架构

### 2.2.6.1. 大数据平台

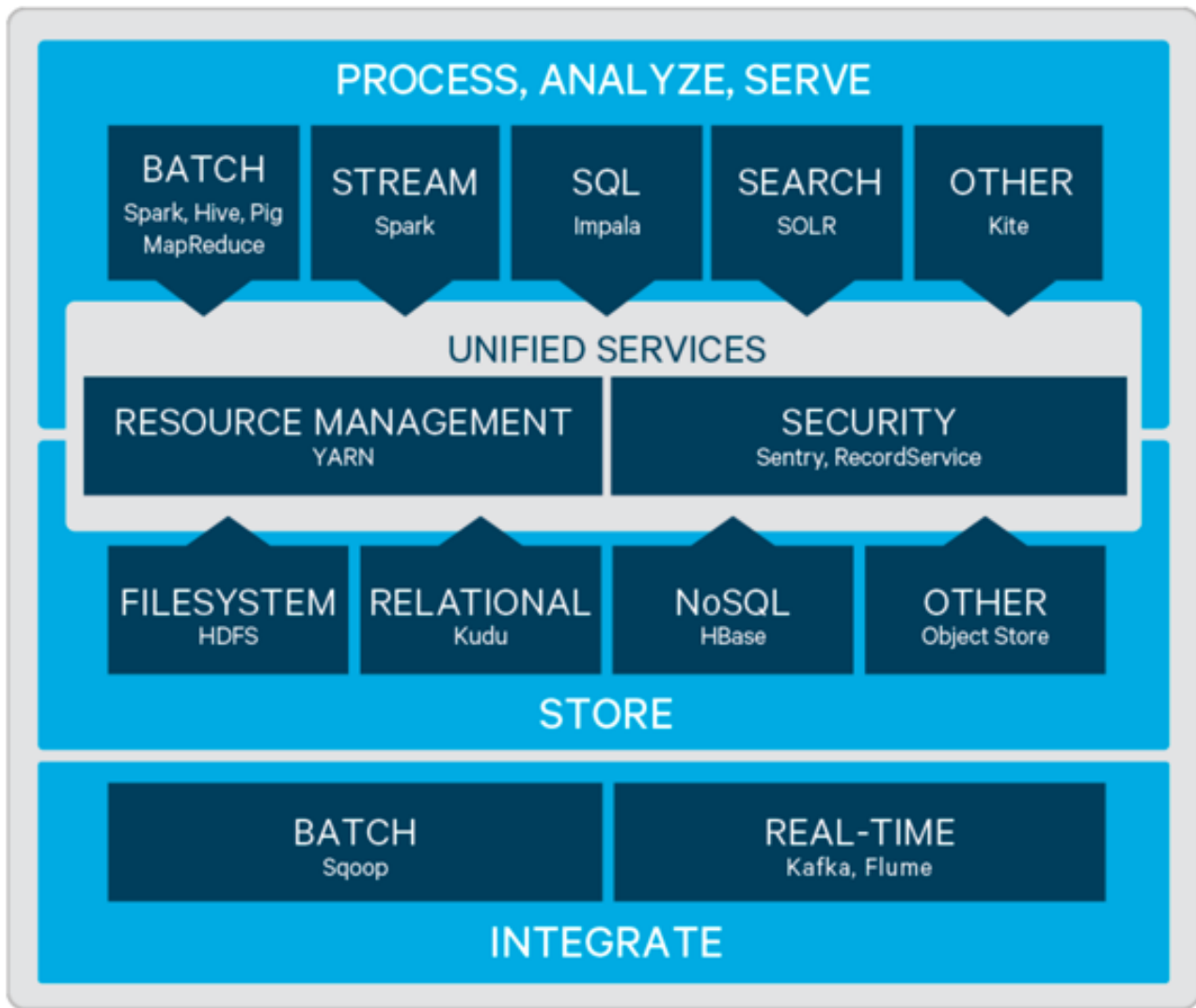
数据采集、数据处理/标准化、数据存储、数据分析/挖掘、数据应用/服务



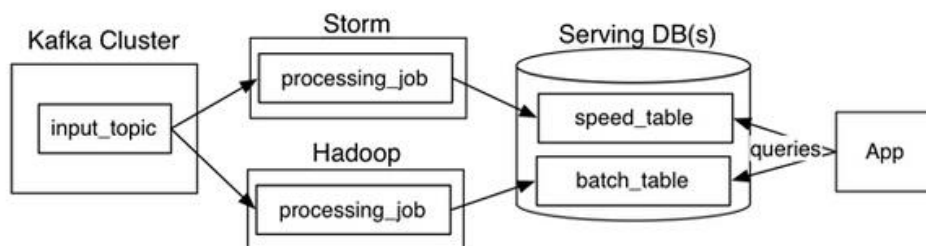
Hortonworks:



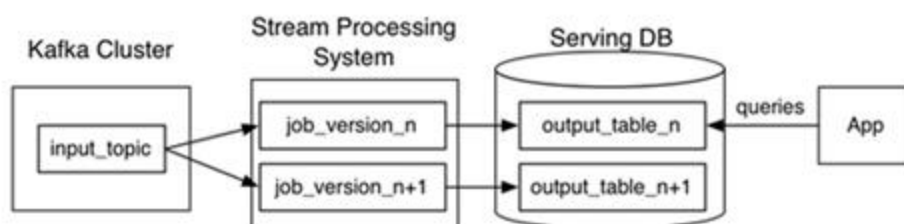
Cloudera:



#### 2. 2. 6. 2. Lambda 架构

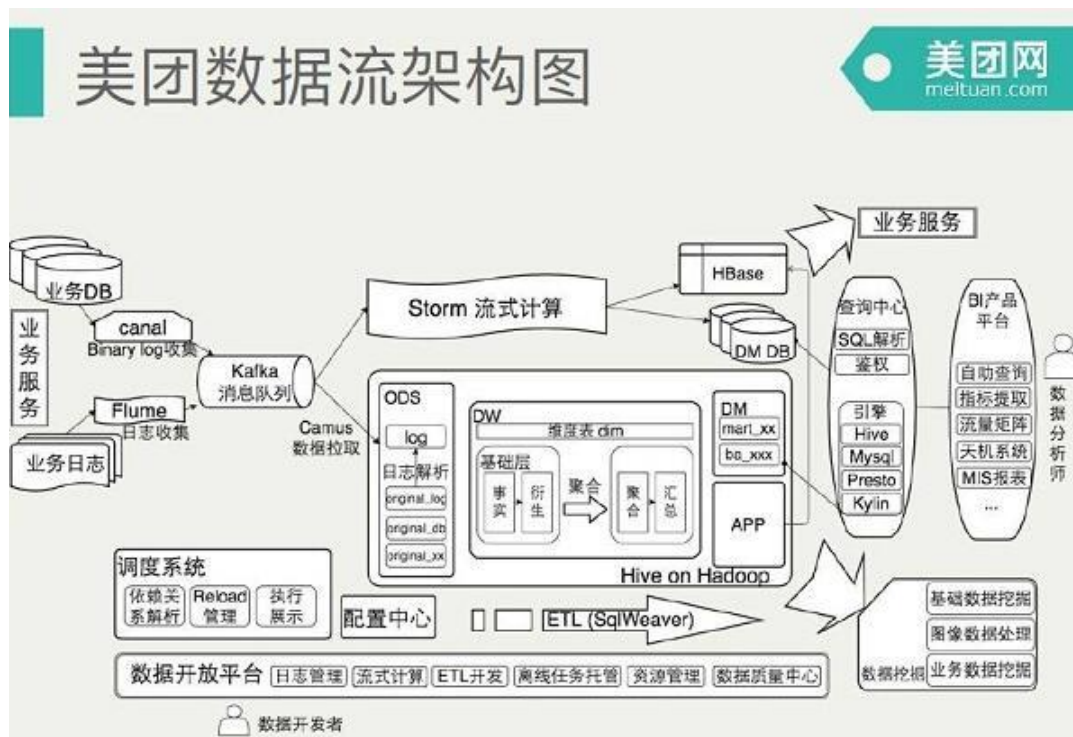


#### 2. 2. 6. 3. Kappa 架构

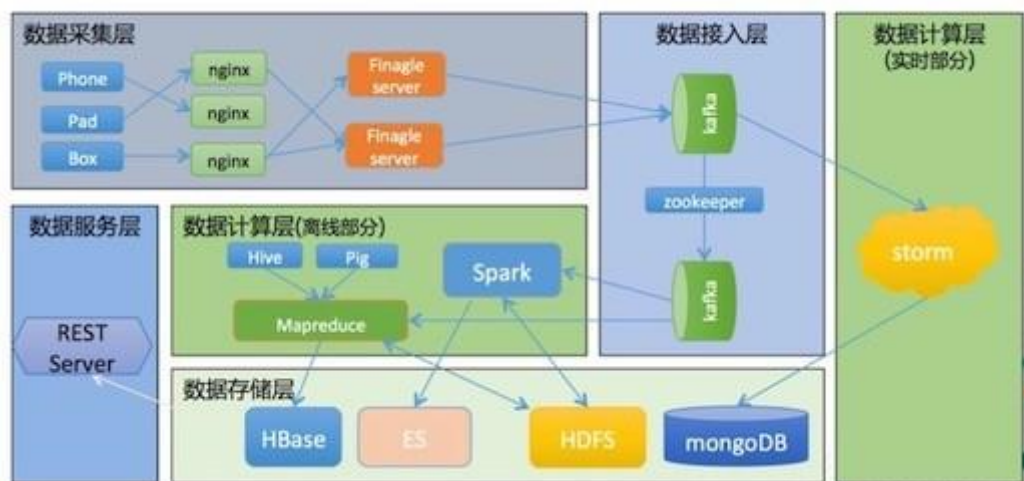


## 2.2.6.4. 部分互联网公司大数据架构

美团：



友盟：



## 2.3. 开源&Apache 软件基金会

### 2.3.1. 开源





Apache 基金会 ( <http://www.apache.org/> )

Linux 基金会 ( <https://www.linuxfoundation.org/> )

GitHub ( <https://github.com/> )

### 2.3.2. Apache 基金会

[Home](#) [About](#) [Projects](#) [People](#) [Get Involved](#) [Download](#) [Support Apache](#)




[The Apache Way](#)  
[Contribute](#)  
[ASF Sponsors](#)


**OPEN.**  
THE APACHE SOFTWARE FOUNDATION provides support for the Apache Community of open-source software projects, which provide software products for the public good.

**INNOVATION.**  
THE APACHE PROJECTS ARE DEFINED by collaborative consensus based processes, an open, pragmatic software license and a desire to create high quality software that leads the way in its field.

**COMMUNITY.**  
WE CONSIDER OURSELVES not simply a group of projects sharing a server, but rather a community of developers and users.



**APACHECON: TOMORROW'S TECHNOLOGY TODAY.**  
Innovation insight from the people and communities behind dozens of Apache projects, from ActiveMQ to Zeppelin: 200+ sessions, 30+ topics, 4 sub-conferences, Innovations from the Apache Incubator, community best-practices, The Apache Way, and more. [Video presentations now available!](#)



#### APACHE PROJECTS

<http://www.apache.org/index.html#projects-list>

User -> Contributor -> Committer -> PMC member -> ASF member

<http://www.apache.org/foundation/how-it-works.html#roles>

<http://www.apache.org/licenses/#2.0>



## 2.3.3. 怎样学习一个 Apache 开源项目？

### 2.3.3.1. What ? How ? Why ?

知其然，知其所以然。

官网：<http://storm.apache.org/>

文档：<http://storm.apache.org/releases/current/index.html>

Quikstart/Example：<https://github.com/apache/storm/tree/master/examples/storm-starter>

代码：<https://github.com/apache/storm>

Issue：<https://issues.apache.org/jira/browse/STORM/>

Wiki：<https://cwiki.apache.org/confluence/display/STORM/Storm+Home>

Mailing list：[user@storm.apache.org](mailto:user@storm.apache.org)      [dev@storm.apache.org](mailto:dev@storm.apache.org)

### 2.3.3.2. 出现问题？

自己解决（官网 FAQ、邮件、Issue、Google） --（超过 1 天）--> 寻求帮助（邮件）

## 2.4. 分布式系统原理

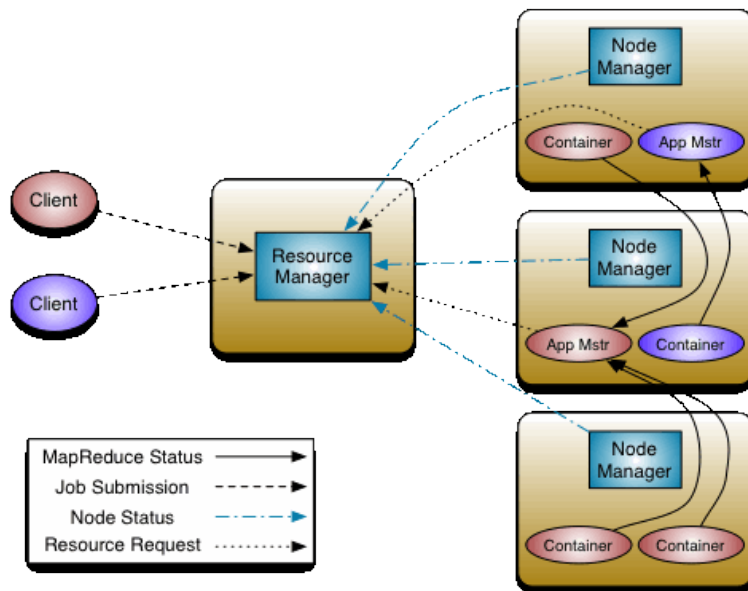
### 2.4.1. 核心思想：分而治之

分什么？资源、任务/数据

一台计算机，从单线程到多线程；一个集群，从单节点到多节点。

通过 Master-Slave 模式对集群进行管理。Master 负责集群的资源管理与任务管理。

Hadoop YARN 架构：



## 2.4.2. 数据分区

常用分区算法：均匀（随机、轮询），hash，一致性 Hash

Hash 分区： $\text{hash}(\text{key}) \% n$

## 2.4.3. 分区算法

任务分配给其他机器的方法，主要由负载均衡的分配算法来解决，常用的负载主要有：

（1）轮询；

将所有请求，依次分发到每台服务器上，适合服务器硬件同相同的场景。

优点：服务器请求数目相同；

缺点：服务器压力不一样，不适合服务器配置不同的情况；

（2）随机；

请求随机分配到各个服务器。

优点：使用简单；

缺点：不适合机器配置不同的场景；

（3）Hash（源地址散列）；

根据 IP 地址进行 Hash 计算，得到 IP 地址。

优点：将来自同一 IP 地址的请求，同一会话期内，转发到相同的服务器；实现会话粘滞。

缺点：目标服务器宕机后，会话会丢失；

(4) 最少链接；

将请求分配到连接数最少的服务器（目前处理请求最少的服务器）。

优点：根据服务器当前的请求处理情况，动态分配；

缺点：算法实现相对复杂，需要监控服务器请求连接数；

(5) 加权等。

在轮询，随机，最少链接，Hash' 等算法的基础上，通过加权的方式，进行负载服务器分配。

优点：根据权重，调节转发服务器的请求数目；

缺点：使用相对复杂；

## 2.4.4. 容错

分布式计算：重新调度+消息重发

分布式存储：多副本

容错：检查点，心跳，租约 Lease

## 2.4.5. 缩容扩容

分布式计算：缩容重新调度，扩容对原有任务无影响

分布式存储：缩容无影响，扩容对原有数据无影响

### 3.探索

( 1 ) 数据中间件插件的了解，在不同的业务中的选取，如;Kafka、RabbitMQ、RocketMQ 等等



<http://zeromq.org/>

<http://www.rabbitmq.com/>

<http://kafka.apache.org/>

<http://bookkeeper.apache.org/distributedlog/>

<http://rocketmq.apache.org/>

<http://activemq.apache.org/>

<http://dev.iron.io/>

( 2 ) 大数据生态圈中各个框架的最新版的新特性，比如 HBase2.0.0、Spark2.3.0、Hadoop3.0 等

( 3 ) 大数据存储类在不同的应用场景下的选取，比如如何构建一个海量数据下可扩展的查询效率比较快的数

据库系统，比如 HBase、Hive、Kylín、Druid、Solr、Elasticsearch、Impala