入行以来，我从事大数据运维也有6、7年了，之前我做过系统运维、DBA，也做过大数据分析师，最后选择了大数据运维方向.在这期间， 我见证并目睹了国内大数据行业发展的历程。掌握大数据就是掌握未来

先看看，这个行业的前景

ppt

今天我来介绍一下大数据hadoop生态圈。

所谓大数据是相对于小数据、传统数据来说的，大数据要解决的就是大规模数据存储、大规模数据计算、大规模数据处理，而 Hadoop 生态系统就是用来实现这些功能的。

要讲清大数据的原理，我们还要从一个故事讲起。

从故事开始：一个电商平台的用户行为分析需求
最近，就职于一家电商公司的小李遇到了一些麻烦事，因为领导突然给他布置了一个任务，要把他们电商平台里所有的用户在 PC 端和 App 上的浏览、点击、购买等行为日志都存放起来集中分析，并形成报表，以供老板每天查看。

最初，小李觉得这个任务比较简单，他的基本思路是将日志数据全部存入 MySQL 库中，然后通过不同条件进行查询、分析，得到老板想要的结果即可，但在具体实施过程中，小李遇到了前所未有的麻烦。

首先，这些数据量太大了，每天网站产生近 500G 的数据，这么大量的日志存储到一个单机的 MySQL 库中，已经难度很大了，磁盘空间经常告警；
其次，老板要的报表展示维度有 20 个之多，过多的维度会导致数据库查询 SQL 非常庞大，SQL 查询效率极度低下，一个 SQL 查询要跑十几个小时，这导致前一天的分析报表，老板第二天无法准时看到
最后，老板要求这些电商数据至少保留一年时间，以供跨年分析、对比，而目前的架构，数据都存储在 MySQL 库中，还是单机，这显然无法满足老板的需求。
小李已经从事技术工作多年了，还是有一定知识储备，他决定改造目前的技术架构。

首先，他将数据库服务器增加到 5 台，并在每台服务器上配置了大容量 SSD 磁盘组，以解决存储能力不足的问题；其次，对 MySQL 进行分库分表处理，以解决单表过大，数据集中

存储查询效率低下的问题。技术架构调整以后，小李觉得分库分表坑太多，一个几百行的大SQL，加上跨库 Join，以及各种复杂的计算，实现快速查询，根本不现实。因此，仍然无法满足老板的需求，更可怕的是，老板提出以后会增加实时查询、分析的功能，这种需求，通过 MySQL 来实现，完全是不可能的。

难道就没有能实现老板需求的技术方案吗？ 不， 其实我们可以通过Hadoop完成老板提出的需求。

Hadoop 与大数据之间到底是什么关系？
Hadoop 是 Apache 下的一个开源项目，说起 Hadoop，通常都会跟“大数据”这几个字联系在一起，但大数据并不等于 Hadoop，大数据本身是个很宽泛的概念，你可以把大数据理解为 Hadoop 的生态圈（或者泛生态圈）。

Hadoop 生态圈好比家里的厨房，厨房里有锅、碗、瓢、盆、勺等各种做饭用具，这些用具类似 Hadoop 生态圈里的各种软件，比如 HDFS、Hive、Pig、Spark、flink等，这些软件各有各的用途，相互配合而又具有自己的独立特性。

接着，我们来分析一下，Hadoop 生态圈架构是否能解决小李当前的困难：海量数据的存储问题和数据查询效率问题。

1、数据存储：HDFS，一个分布式文件系统
HDFS，它是 Hadoop 技术体系中的核心基石，负责分布式存储数据，你可以把它理解为一个分布式的文件系统。此文件系统的主要特征是数据分散存储，一个文件存储在 HDFS 上时会被分成若干个数据块，每个数据块分别存储在不同的服务器上。

假如你有 100 台服务器，那么所有数据会平均分担在这 100 台机器上。而且，为了保证数据安全，每个存储在 HDFS 上的文件，可以设置不同的备份数。假如你设置了 3 个文件备份，只要你的服务器不是同时坏 3 个，那 HDFS 上面的数据都是安全的。

这个 HDFS 就解决了小李的两个问题：存储容量和数据安全。

2. 数据分析：MapReduce 计算引擎
数据存储问题解决了，接下来数据该如何分析处理呢？

单机处理的话，已经证明过是不可能的，必须用多台服务器并行处理，那么就要考虑如何分配计算任务到多台机器。如果一台机器挂了，该如何重新启动相应的分析任务，以及机器之间如何互相通信、交换数据以完成复杂的计算等。这就是马上要讲的 Hadoop 中的计算引擎，其有多种计算引擎，MapReduce 是第一代计算引擎，Tez 和 Spark 是第二代。

MapReduce 的强大在于分布式计算，也就是将计算任务分布在多个服务器上，因此服务器数量越多，计算速度就越快。

MapReduce 主要分为两阶段：Map 阶段和 Reducer 阶段。比如，你要读取 HDFS 上一个大文件中某个 IP 出现的频次，那么 Map 阶段就是多台机器同时读取这个文件内容的一个部分，然后分别统计出各自读到的内容中此 IP 出现的频次，这相当于是分散读取；Reducer 阶段是将 Map 阶段的输出结果作为输入，然后进行整合、汇总，最终得到一个此 IP 出现次数的结果。

由此可以看出，MapReduce 的过程就是一个分分合合的过程，而这个分布式计算功能完美解决了小李在 MySQL 中查询效率低下的问题。

那老板提出的实时查询分析功能，Hadoop 这个生态圈能实现吗？当然可以。

Hadoop 生态圈
我们先来了解下 Hadoop 生态圈的常用组件

下图展示了 Hadoop 生态圈常见的软件和应用场景：

可以看出，Hadoop 的基础是 HDFS 和 Yarn，在此基础上有各种计算模型，如 MapReduce、Spark、HBase 等；而在计算模型上层，对应的是各种分布式计算辅助工具，如 Hive、Pig、Sqoop 等。此外，还有分布式协作工作 ZooKeeper 以及日志收集工具 Flume，这么多工具如何协作使用呢？这就是任务调度层 Oozie 的存在价值，它负责协调任务的有序执行。最顶层是 Hadoop 整个生态圈的统一管理工具，Ambari 可以为 Hadoop 以及相关大数据软件使用提供更多便利。

下面我来依次介绍图中的技术点。

1. HDFS（Hadoop 分布式文件系统）
HDFS 是 Hadoop 生态圈中提供分布式存储支持的系统，上层的很多计算框架（Hbase、Spark 等）都依赖于 HDFS 存储。

若要构建 HDFS 文件系统，不需要特有的服务器，普通 PC 即可实现，它对硬件和磁盘没有任何特殊要求，也就是说，HDFS 可在低成本的通用硬件上运行。前面的介绍中，我们也看到了，它不但解决了海量数据存储问题，还解决了数据安全问题。

为了更好的理解它的作用，我们来看一个 HDFS 分布式文件系统的实现原理图：

可以看出，HDFS 主要由 NameNode 和 DataNode 两部分组成。

NameNode 是 HDFS 的管理节点，它存储了元数据（文件对应的数据块位置、文件大小、文件权限等）信息，同时负责读写调度和存储分配；
DataNode 节点是真正的数据存储节点，用来存储数据。另外，在 DataNode 上的每个数据块会根据设置的副本数，进行分级复制，保证同一个文件的每个数据块副本，都不在同一个机器上。
2. MapReduce（分布式计算模型）离线计算
何为离线计算，其实就是非实时计算。

比如，老板让小李今天出昨天电商网站的报表数据，这其实是对数据做离线计算；老板要马上看到来自北京 App 端用户的实时访问数据，这就是实时计算。当然实时计算也不是完全实时，它一定有一个延时，只不过这个延时很短而已。

MapReduce 到现在已经 15 年了，这种 Map 加 Reduce 的简单计算模型，解决了当时单机计算的缺陷，时至今日还有很多场景仍在使用这种计算模型，但已经慢慢不能满足我们的使用需求了。大数据时代的今天，数据量都在 PB 级甚至 EB 级别，对数据的分析效率有了更高的要求。

于是，第二代计算模型产生了，如 Tez 和 Spark，它们通过大量使用内存、灵活的数据交换，更少的磁盘读写来提高分析效率。

3. Yarn（分布式资源管理器）

计算模型层出不穷，这么多计算模型如何协同工作、如何做好资源管理，就显得至关重要了。于是，在 MapReduce 基础上演变出了 Yarn 这个资源管理器，它的出现主要就是为了解决原始 Hadoop 扩展性较差、不支持多种计算模型的问题。

在YARN中，支持CPU和内存两种资源管理，资源管理由ResourceManager（RM）、ApplicationMaster（AM）和NodeManager（NM）共同完成。其中，RM负责对各个NM上的资源进行统一管理和调度。而NodeManager则负责资源的供给和隔离。当用户提交一个应用程序时，会创建一个用以跟踪和管理这个程序的AM，它负责向RM申请资源，并要求NM启动指定资源的任务。这就是YARN的基本运行机制。

最后，Yarn 作为一个通用的分布式资源管理器，它可以管理多种计算模型，如 Spark、Storm、MapReduce 、Flink 等都可以放到 Yarn 下进行统一管理。

4. Spark（内存计算）
Spark 提供了内存中的分布式计算能力，相比传统的 MapReduce 大数据分析效率更高、运行速度更快。总结一句话：以内存换效率。

说到 Spark，不得不提 MapReduce。传统的 MapReduce 计算过程的每一个操作步骤发生在内存中，但产生的中间结果会储存在磁盘里，下一步操作时又会将这个中间结果调用到内存中，如此循环，直到分析任务最终完成。这就会产生读取成本，造成效率低下。

而 Spark 在执行分析任务中，每个步骤也是发生在内存之中，但中间结果会直接进入下一个步骤，直到所有步骤完成之后才会将最终结果写入磁盘。也就是说 Spark 任务在执行过程中，中间结果不会“落地”，这就节省了大量的时间。

在执行一个分析任务中，如果执行步骤不多，可能看不出 MapReduce 和 Spark 执行效率的区别，但是当一个任务有很多执行步骤时，Spark 的执行效率就体现出来了。

5. HBase（分布式列存储数据库）
在介绍 HBase 之前，我们首先了解两个概念：面向行存储和面向列存储。

面向行存储，这个应该接触比较多，比如我们熟悉的 MySQL、Oracle 等就是此种类型的。面向行存储的数据库主要适合于事务性要求严格的场合，这种传统关系型数据库为了实现强一致性，通过严格的事务来进行同步，这就让系统在可用性和伸缩性方面大大折扣。

面向列存储的数据库也叫非关系型数据库（NoSQL），比如Cassandra、HBase等。这种数据库通常将不同数据的同一个属性值存在一起，在查询时只遍历需要的数据，实现了数据即是索引。因此，它的最大优点是查询速度快，这对数据完整性要求不高的大数据处理领域，比如互联网，犹为重要。

Hbase继承了列存储的特性，它非常适合需对数据进行随机读、写操作、比如每秒对PB级数据进行几千次读、写访问是非常简单的操作。 其次，Hbase构建在HDFS之上，其内部管理的文件全部存储在HDFS中。这使它具有高度容错性和可扩展性，并支持Hadoop mapreduce程序设计模型。

如果你的应用是交易历史查询系统、查询场景简单，检索条件较少、每天有千万行数据更新、那么Hbase将是一个很好的选择。其实，行存储和列存储只是不同的维度而已，没有天生的优劣，而大数据时代大部分的查询模式决定了列式存储优于行式存储。

讲到这里，突然发现，小李遇到的技术难题，其实用 HBase 也能实现。

6. Hive（数据仓库）

小李打算在 Hadoop 生态平台上完成公司电商数据的存储和分析了，但又遇到了难题，MapReduce 的程序写起来很麻烦，如果通过写 MapReduce 程序来实现老板的需求，不但要重新学习，而且功能实现也繁琐。该怎么办呢？

经过调研与查阅资料，一款 Hive 工具出现在他面前。Hive 定义了一种类似 SQL 的查询语言（HQL），它可以将 SQL 转化为 MapReduce 任务在 Hadoop 上执行。这样，小李就可以用更简单、更直观的语言去写程序了。

因此，哪怕你不熟悉 MapReduce 程序，只要会写标准的 SQL 语句，也能对 HDFS 上的海量数据进行分析和计算。

7. Oozie（工作流调度器）

小李现在已经能够熟练使用 Hive 对数据进行各种维度的分析了，由于老板要求定时给出报表数据，所以小李就将数据分析任务写成脚本，然后放到操作系统的定时任务（Crontab）中定期执行。刚开始这种方式完全满足了老板的要求，但随着报表任务的增多，一个脚本已经无法满足。

于是，小李根据不同的任务需求，写了多个脚本程序，然后放到操作系统定时任务中去执行。这种方法大多时候都能正常完成分析任务，但也遇到了任务分析错误或失败的情况，小李最终发现这是定时任务出现了问题。

原来在小李写的多个脚本中，个别脚本有相互依赖性，也就是说，假定有脚本 A 和脚本 B，脚本 B 要执行的话，必须等待脚本 A 完成，否则脚本 B 启动就没有意义了。因此，他在操作系统定时任务中，通过设置脚本开始执行时间的差别来避免这种依赖性。

比如，脚本 A 凌晨 6 点执行，小李预估此脚本最多执行到 8 点就完成了，所以设置脚本 B 在 8:30 时启动执行。可是，仔细一想，就觉得这种设置肯定存在问题，比如脚本 A 执行失败，或者在 8:30 没有完成怎么办？

小李发现，某次任务执行失败是因为他认为脚本 C 2 个小时肯定执行完，但事实上却执行了 4 个多小时，由于当天的日志量非常大，分析时间也相应延长了，脚本 C 在预估的时间内没有完成，而下个脚本 D 如约启动，脚本 D 的执行要依赖于脚本 C 的输出结果，因此脚本 D 肯定执行失败。

如何解决这个问题呢？Oozie 出场了。Oozie 是一个基于工作流引擎的调度器，它其实就是一个运行在 Java Servlet 容器（如 Tomcat）中的 Javas Web 应用，你可以在它上面运行 Hadoop 的 Map Reduce 和 Pig 等任务，。

对于 Oozie 来说，工作流就是一系列的操作（如 Hadoop 的 MR，Pig 的任务、Shell 任务等），通过 Oozie 可以实现多个任务的依赖性。也就是说，一个操作的输入依赖于前一个任务的输出，只有前一个操作完全完成后，才能开始第二个。

Oozie 工作流通过 hPDL 定义（hPDL 是一种 XML 的流程定义语言），工作流操作通过远程系统启动任务。当任务完成后，远程系统会进行回调来通知任务已经结束，然后再开始下一个操作。

8. Sqoop 与 Pig

小李还有一个苦恼，他要把原来存储在 MySQL 中的数据导入 Hadoop 的 HDFS 上，是否能实现呢？这当然可以，通过 Sqoop（SQL-to-Hadoop）就能实现，它主要用于传统数据库和 Hadoop 之间传输数据。数据的导入和导出本质上是 MapreDuce 程序，充分利用了 MR 的并行化和容错性。

通过 Hive 可以把脚本和 SQL 语言翻译成 MapReduce 程序，扔给计算引擎去计算。Pig 与 Hive 类似，它定义了一种数据流语言，即 Pig Latin，它是 MapReduce 编程的复杂性的抽象，Pig Latin 可以完成排序、过滤、求和、关联等操作，支持自定义函数。Pig 自动把 Pig Latin 映射为 MapReduce 作业，上传到集群运行，减少用户编写 Java 程序的苦恼。

9. Flume（日志收集工具）

现在小李已经基本解决了老板提出的各种数据分析需求，数据分析任务在 Hadoop 上有条不紊的进行。现在电商平台的数据是通过 rsync 方式定时从电商服务器上同步到 Hadoop 平台的某台机器，然后通过这台机器 put 到 HDFS 上，每天定时同步一次，由于数据量很大，同步一次数据在一个小时左右，并且同步数据的过程会消耗大量网络带宽。小李想，有没有更合适的数据传输机制，一方面可以保证数据传输的实时性、完整性，另一方面也能节省网络带宽。

通过 Flume 可以圆满完成小李现在的困惑，那么什么是 Flume 呢？来个官方的概念，Flume 是将数据从产生、传输、处理并最终写入目标路径的过程抽象为数据流，在具体的数据流中，数据源支持在 Flume 中定制数据发送方，从而支持收集各种不同协议数据。

同时，Flume 数据流提供对日志数据进行简单处理的能力，如过滤、格式转换等。此外，Flume 还具有能够将日志写往各种数据目标（文件、HDFS、网络）的能力。在 Hadoop 平台，我们主要使用的是通过 Flume 将数据从源服务器写入 Hadoop 的 HDFS 上。

10. Kafka（分布式消息队列）

相信我们都乘坐过地铁，正常情况下先安检后刷卡，最后进站上车，如果遇到上下班高峰期，地铁的人流会很多，坐地铁的顺序就变成了先进入引流系统排队，然后进行安检，最后进站上车，从这里可以看出，在地铁人流量大的时候会多一个“引流系统排队”，通过这个引流系统，可以保证在人多的时候乘坐地铁也能有条不紊的进行。

这个引流系统就跟我们要介绍的 Kafka 的作用非常类似，它在人和地铁中间作为一个缓存，实现解耦合的作用。

专业术语来描述一下，现在是个大数据时代，各种商业、社交、搜索、浏览都会产生大量的数据。那么如何快速收集这些数据，如何实时的分析这些数据，是一个必须要解决的问题，同时，这也形成了一个业务需求模型，即生产者生产（Produce）各种数据、消费者（Consume）消费（分析、处理）这些数据。那么面对这些需求，如何高效、稳定的完成数

据的生产和消费呢？这就需要在生产者与消费者之间，建立一个通信的桥梁，这个桥梁就是消息系统。从微观层面来说，这种业务需求也可理解为不同的系统之间如何传递消息。

Kafka 是 Apache 组织下的一个开源系统，它的最大特性就是可以实时的处理大量数据以满足各种需求场景：比如基于 Hadoop 平台的数据分析、低时延的实时系统、Storm/Spark 流式处理引擎等。Kafka 现在它已被多家大型公司作为多种类型的数据管道和消息系统使用。

11. ZooKeeper（分布式协作服务）

对集群技术应该并不陌生，就拿最简单的双机热备架构来说，双机热备主要用来解决单点故障问题，传统的方式是采用一个备用节点，这个备用节点定期向主节点发送 ping 包，主节点收到 ping 包以后向备用节点发送回复信息，当备用节点收到回复的时候就会认为当前主节点运行正常，让它继续提供服务。而当主节点故障时，备用节点就无法收到回复信息了，此时，备用节点就认为主节点宕机，然后接替它成为新的主节点继续提供服务。

这种传统解决单点故障的方法，虽然在一定程度上解决了问题，但是有一个隐患，就是网络问题，可能会存在这样一种情况：主节点并没有出现故障，只是在回复响应的时候网络发生了故障，这样备用节点就无法收到回复，那么它就会认为主节点出现了故障；接着，备用节点将接管主节点的服务，并成为新的主节点，此时，集群系统中就出现了两个主节点（双 Master 节点）的情况，双 Master 节点的出现，会导致集群系统的服务发生混乱。这样的话，整个集群系统将变得不可用，为了防止出现这种情况，就需要引入 ZooKeeper 来解决这种问题。

ZooKeeper 是如何来解决这个问题的呢，这里以配置两个节点为例，假定它们是“节点 A”和“节点 B”，当两个节点都启动后，它们都会向 ZooKeeper 中注册节点信息。我们假设“节点A”锁注册的节点信息是“master00001”，“节点B”注册的节点信息是“master00002”，注册完以后会进行选举，选举有多种算法，这里以编号最小作为选举算法，那么编号最小的节点将在选举中获胜并获得锁成为主节点，也就是“节点A”将会获得锁成为主节点，然后“节点B”将被阻塞成为一个备用节点。这样，通过这种方式 ZooKeeper 就完成了对两个 Master 进程的调度。完成了主、备节点的分配和协作。

如果“节点A”发生了故障，这时候它在 ZooKeeper 所注册的节点信息会被自动删除，而 ZooKeeper 会自动感知节点的变化，发现“节点 A”故障后，会再次发出选举，这时候“节点 B”将在选举中获胜，替代“节点 A”成为新的主节点，这样就完成了主、被节点的重新选举。

如果“节点A”恢复了，它会再次向 ZooKeeper 注册自身的节点信息，只不过这时候它注册的节点信息将会变成“master00003”，而不是原来的信息。ZooKeeper 会感知节点的变化再次发动选举，这时候“节点 B”在选举中会再次获胜继续担任“主节点”，“节点 A”会担任备用节点。

通俗的讲，ZooKeeper 相当于一个和事佬的角色，如果两人之间发生了一些矛盾或者冲突，无法自行解决的话，这个时候就需要 ZooKeeper 这个和事佬从中进行调解，而和事佬调解的方式是站在第三方客观的角度，根据一些规则（如道德规则、法律规则），客观的对冲突双方做出合理、合规的判决。

12. Ambari（大数据运维工具）

Ambari 是一个大数据基础运维平台，它实现了 Hadoop 生态圈各种组件的自动化部署、服务管理和监控告警，Ambari 通过 puppet 实现自动化安装和配置，通过 Ganglia 收集监控度量指标，用 Nagios 实现故障报警。目前 Ambari 已支持大多数 Hadoop 组件，包括 HDFS、MapReduce、Oozie、Hive、Pig、 Hbase、ZooKeeper、Sqoop、Kafka、Spark、Druid、Storm 等几十个常用的 Hadoop 组件。

作为大数据运维人员，通过 Ambari 可以实现统一部署、统一管理、统一监控，可极大提高运维工作效率。

总结

到这里，已经介绍完了 Hadoop 生态圈常用的组件，相信你们对它们的用途也有了大致了解，

今天的内容比较多，这源于大数据生态圈组件繁多以及应用的复杂性，

分享就到这里　谢谢大家