高校大数据管理技术及数据挖掘分析技术技术热点及应用模式调研

目录

[1 图形数据库 2](#_Toc436727676)

[1.1 简介 2](#_Toc436727677)

[1.2 数据存储模式 3](#_Toc436727678)

[1.3 应用场景 4](#_Toc436727679)

[1.4 Neo4j集群 5](#_Toc436727680)

[2 分布式存储及分布式计算（Hadoop） 5](#_Toc436727681)

[2.1 分布式文件系统 5](#_Toc436727682)

[2.2 分布式计算Map/Reduce 19](#_Toc436727683)

[2.3 分布式数据库Hbase 27](#_Toc436727684)

[3 数据仓库与数据联机分析 29](#_Toc436727685)

[3.1 数据挖掘系统关键技术 29](#_Toc436727686)

[4 数据集成交换 48](#_Toc436727687)

[4.1 集成中心库模式 48](#_Toc436727688)

[4.2 参照模型标准及数据 49](#_Toc436727689)

[4.3 执行标准模型及数据 49](#_Toc436727690)

[4.4 集成接口支持 49](#_Toc436727691)

[4.5 数据集成KM模块 50](#_Toc436727692)

[4.6 集成设计工具 50](#_Toc436727693)

[4.7 集成查看工具 50](#_Toc436727694)

[4.8 集成调度工具 50](#_Toc436727695)

[4.9 应用场景 51](#_Toc436727696)

[5 数据可视化技术 ECHARTS 51](#_Toc436727697)

[5.1 混搭 51](#_Toc436727698)

[5.2 拖拽重计算 51](#_Toc436727699)

[5.3 数据视图 51](#_Toc436727700)

[5.4 动态类型切换 52](#_Toc436727701)

[5.5 图例开关 52](#_Toc436727702)

[5.6 数据区域选择 52](#_Toc436727703)

[5.7 多图联动 52](#_Toc436727704)

[5.8 值域漫游 52](#_Toc436727705)

[5.9 大规模数据模式 53](#_Toc436727706)

[5.10 动态数据添加 53](#_Toc436727707)

[5.11 标线辅助 53](#_Toc436727708)

[5.12 多维度堆积 53](#_Toc436727709)

[5.13 标注 & 标线 53](#_Toc436727710)

[5.14 个性化定制 54](#_Toc436727711)

[5.15 事件交互 54](#_Toc436727712)

[5.16 百搭时间轴 54](#_Toc436727713)

[6 ORACLE数据缓存技术 54](#_Toc436727714)

[6.1 空闲缓存块 55](#_Toc436727715)

[6.2 命中缓存块 55](#_Toc436727716)

[6.3 脏缓存块 56](#_Toc436727717)

[7 挖掘算法 56](#_Toc436727718)

[7.1 聚类 56](#_Toc436727719)

[7.2 分类 57](#_Toc436727720)

[7.3 关联 57](#_Toc436727721)

[7.4 预测 58](#_Toc436727722)

## 图形数据库

### 简介

    ->图形数据库是一种非关系型数据库，它应用图形理论存储实体之间的关系信息。最常见的一个例子，就是社会网络中人与人之间的关系。关系型数据库用于存储“关系型”数据的效果并不好，其查询复杂、缓慢、超出预期，而图形数据库的独特设计恰恰弥补了这个缺陷。

    ->Neo4j基于Java实现，兼容ACID特性，保证数据的最终一致性

    ->避免关系型数据库中存在的联结大量表、深度嵌套的SQL查询情况。

    ->Neo4j是一个用Java实现、完全兼容ACID的图形数据库。数据以一种针对图形网络进行过优化的格式保存在磁盘上。Neo4j的内核是一种极快的图形引擎，具有数据库产品期望的所有特性，如恢复、两阶段提交、符合XA等。自2003年起，Neo4j就已经被作为24/7的产品使用。通过联机备份实现的高可用性和主从复制目前处于测试阶段，预计在下一版本中发布。Neo4j既可作为无需任何管理开销的内嵌数据库使用；也可以作为单独的服务器使用，在这种使用场景下，它提供了广泛使用的REST接口，能够方便地集成到基于PHP、.NET和JavaScript的环境里。但本文的重点主要在于讨论Neo4j的直接使用。

     开发者可以通过[Java-API](http://api.neo4j.org/current/)直接与图形模型交互，这个API暴露了非常灵活的数据结构。至于象[JRuby/Ruby](http://github.com/andreasronge/neo4j)、[Scala](http://github.com/ept/neo4j-scala-template)、[Python](http://components.neo4j.org/neo4j.py/)、[Clojure](http://github.com/JulianMorrison/neo4j-clojure)等其他语言，社区也贡献了优秀的绑定库。Neo4j的典型数据特征：

* 数据结构不是必须的，甚至可以完全没有，这可以简化模式变更和延迟数据迁移。
* 可以方便建模常见的复杂领域数据集，如[CMS里的访问控制](http://wiki.neo4j.org/content/ACL)可被建模成细粒度的访问控制表，[类对象数据库的用例](http://code.google.com/p/jo4neo/)、[TripleStores](http://en.wikipedia.org/wiki/Triplestore)以及[其他例子](http://blog.neo4j.org/2009/12/holiday-fun-with-neo4j.html)。
* 典型使用的领域如[语义网和RDF](http://en.wikipedia.org/wiki/Semantic_Web)、[LinkedData](http://en.wikipedia.org/wiki/Linked_Data)、[GIS](http://resources.esri.com/help/9.3/arcgisengine/dotnet/e084da94-d4f7-4da7-86ed-7df684ff2144.htm)、[基因分析](http://code.google.com/p/pygr/)、[社交网络数据建模](http://blog.neo4j.org/2009/09/social-networks-in-database-using-graph.html)、[深度推荐算法](http://wiki.github.com/tinkerpop/gremlin/linkeddata-sail#recommendation)以及[其他领域](http://wiki.neo4j.org/content/Neo4j_In_The_Wild)。

    ->高性能

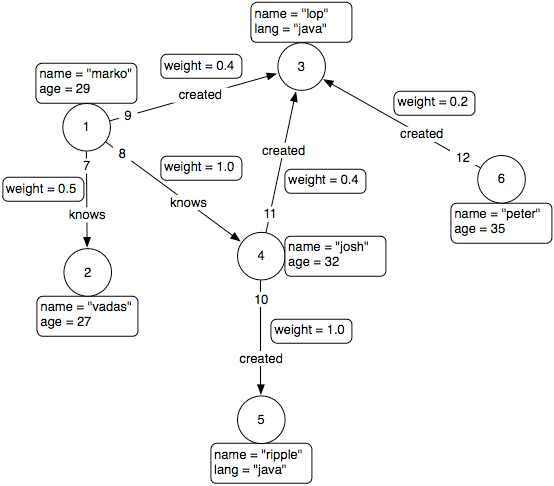
    要给出确切的性能基准数据很难，因为它们跟底层的硬件、使用的数据集和其他因素关联很大。自适应规模的Neo4j无需任何额外的工作便可以处理包含数十亿节点、关系和属性的图。它的读性能可以很轻松地实现每毫秒（大约每秒1-2百万遍历步骤）遍历2000关系，这完全是事务性的，每个线程都有热缓存。使用最短路径计算，Neo4j在处理包含数千个节点的小型图时，甚至比MySQL快1000倍，随着图规模的增加，差距也越来越大。

### 数据存储模式

    ->图数据库由一系列结点和边界组成；每一个结点代表了一个实体，每一个边界代表了两个结点之间的一种连接或者关系。图形数据库系统将世界看作由节点（对象，即人、地点和事物）和边（关联，即他们之间的关系）及其属性信息组成的图，实现了图数据结构存储。Neo4j为属性图形模型

    ->属性图形模型（用一张图反映这个模型），按照该模型，属性图里信息的建模使用3种构造单元：

* 节点（即顶点）
* 关系（即边） - 具有方向和类型（标记和标向）
* 节点和关系上面的属性（即特性）



### 应用场景

    ->除了社交应用之外，图数据库同很多应用可以工作。很多应用可以自然扩展使用图形类型的关系。比如，好多内嵌在社交应用中的推荐系统常常都是基于图形的系统。将校园内的所有人、事、物以节点形式、关系（老乡、校友）以边的形式存储到图形数据库中，将整个教育生态系统中涉及到的数据集建模成一个大型稠密的网络结构。

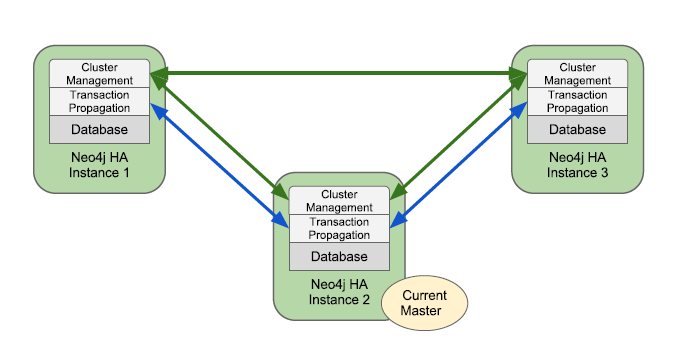
### Neo4j集群

    模拟Neo4j的高可用：Neo4j企业版本才提供高可用性功能。

    Neo4j HA主要提供以下两个功能：

* 容错数据库架构 保存多个数据副本，即使硬件故障，也能保证可读写。
* 水平方向扩展以读为主架构 读操作负载均衡。

    Neo4j HA模式总有单个master，零个或多个slave。与其他ms复制架构，Neo4j HA的slave可以处理写操作，而无需重定向写入到master。



## 分布式存储及分布式计算（Hadoop）

### 分布式文件系统

#### 简介

与目前常见的集中式存储技术不同，分布式存储技术并不是将数据存储在某个或多个特定的节点上，而是通过网络使用企业中的每台机器上的磁盘空间，并将这些分散的存储资源构成一个虚拟的存储设备，数据分散的存储在企业的各个角落。计算机通过文件系统管理、存储数据，而信息爆炸时代中人们可以获取的数据成指数倍的增长，单纯通过增加硬盘个数来扩展计算机文件系统的存储容量的方式，在容量大小、容量增长速度、数据备份、数据安全等方面的表现都差强人意。分布式文件系统可以有效解决数据的存储和管理难题：将固定于某个地点的某个文件系统，扩展到任意多个地点/多个文件系统，众多的节点组成一个文件系统网络。每个节点可以分布在不同的地点，通过网络进行节点间的通信和数据传输。人们在使用分布式文件系统时，无需关心数据是存储在哪个节点上、或者是从哪个节点从获取的，只需要像使用本地文件系统一样管理和存储文件系统中的数据。

#### 决定因素

文件系统最初设计时，仅仅是为局域网内的本地数据服务的。而分布式文件系统将服务范围扩展到了整个网络。不仅改变了数据的存储和管理方式，也拥有了本地文件系统所无法具备的数据备份、数据安全等优点。判断一个分布式文件系统是否优秀，取决于以下三个因素：

->数据的存储方式，例如有1000万个数据文件，可以在一个节点存储全部数据文件，在其他N个节点上每个节点存储1000/N万个数据文件作为备份；或者平均分配到N个节点上存储，每个节点上存储1000/N万个数据文件。无论采取何种存储方式，目的都是为了保证数据的存储安全和方便获取。

->数据的读取速率，包括响应用户读取数据文件的请求、定位数据文件所在的节点、读取实际硬盘中数据文件的时间、不同节点间的数据传输时间以及一部分处理器的处理时间等。各种因素决定了分布式文件系统的用户体验。即分布式文件系统中数据的读取速率不能与本地文件系统中数据的读取速率相差太大，否则在本地文件系统中打开一个文件需要2秒，而在分布式文件系统中各种因素的影响下用时超过10秒，就会严重影响用户的使用体验。

->数据的安全机制，由于数据分散在各个节点中，必须要采取冗余、备份、镜像等方式保证节点出现故障的情况下，能够进行数据的恢复，确保数据安全。

#### 具体产品HDFS

    常见的分布式文件系统有，GFS、HDFS、Lustre 、Ceph 、GridFS 、mogileFS、TFS、FastDFS等。各自适用于不同的领域。它们都不是系统级的分布式文件系统，而是应用级的分布式文件存储服务。本次调研选定的是Hadoop的分布式文件系统HDFS着重阐述。Hadoop分布式文件系统(HDFS)被设计成适合运行在通用硬件(commodity hardware)上的分布式文件系统。它和现有的分布式文件系统有很多共同点。但同时，它和其他的分布式文件系统的区别也是很明显的。HDFS是一个高度容错性的系统，适合部署在廉价的机器上。HDFS能提供高吞吐量的数据访问，非常适合大规模数据集上的应用。HDFS放宽了一部分POSIX约束，来实现流式读取文件系统数据的目的。HDFS在最开始是作为Apache Nutch搜索引擎项目的基础架构而开发的。HDFS是Apache Hadoop Core项目的一部分。

* 主控服务器，NameNode，整个文件系统的大脑，它提供整个文件系统的目录信息，并且管理各个数据服务器。
* 数据服务器，DataNode，分布式文件系统中的每一个文件，都被切分成若干个数据块，每一个数据块都被存储在不同的服务器上，此服务器称之为数据服务器。
* 数据块，Block，每个文件都会被切分成若干个块，每一块都有连续的一段文件内容，是存储的基本单位，在这里统一称做数据块。
* 数据包，Packet，客户端写文件的时候，不是一个字节一个字节写入文件系统的，而是累计到一定数量后，往文件系统中写入一次，每发送一次的数据，都称为一个数据包。
* 传输块，Chunk，在每一个数据包中，都会将数据切成更小的块，每一个块配上一个奇偶校验码，这样的块，就是传输块。
* 备份主控服务器，SecondaryNameNode，备用的主控服务器，在身后默默的拉取着主控服务器 的日志，等待主控服务器牺牲后被扶正。

##### 前提和设计目标

###### 硬件错误

硬件错误是常态而不是异常。HDFS可能由成百上千的服务器所构成，每个服务器上存储着文件系统的部分数据。我们面对的现实是构成系统的组件数目是巨大的，而且任一组件都有可能失效，这意味着总是有一部分HDFS的组件是不工作的。因此错误检测和快速、自动的恢复是HDFS最核心的架构目标。

###### 流式数据访问

运行在HDFS上的应用和普通的应用不同，需要流式访问它们的数据集。HDFS的设计中更多的考虑到了数据批处理，而不是用户交互处理。比之数据访问的低延迟问题，更关键的在于数据访问的高吞吐量。POSIX标准设置的很多硬性约束对HDFS应用系统不是必需的。为了提高数据的吞吐量，在一些关键方面对POSIX的语义做了一些修改。

###### 大规模数据集

运行在HDFS上的应用具有很大的数据集。HDFS上的一个典型文件大小一般都在G字节至T字节。因此，HDFS被调节以支持大文件存储。它应该能提供整体上高的数据传输带宽，能在一个集群里扩展到数百个节点。一个单一的HDFS实例应该能支撑数以千万计的文件。

###### 简单的一致性模型

HDFS应用需要一个“一次写入多次读取”的文件访问模型。一个文件经过创建、写入和关闭之后就不需要改变。这一假设简化了数据一致性问题，并且使高吞吐量的数据访问成为可能。Map/Reduce应用或者网络爬虫应用都非常适合这个模型。目前还有计划在将来扩充这个模型，使之支持文件的附加写操作。

###### “移动计算比移动数据更划算”

一个应用请求的计算，离它操作的数据越近就越高效，在数据达到海量级别的时候更是如此。因为这样就能降低网络阻塞的影响，提高系统数据的吞吐量。将计算移动到数据附近，比之将数据移动到应用所在显然更好。HDFS为应用提供了将它们自己移动到数据附近的接口。

###### 异构软硬件平台间的可移植性

HDFS在设计的时候就考虑到平台的可移植性。这种特性方便了HDFS作为大规模数据应用平台的推广。

##### Namenode 和 Datanode

HDFS采用master/slave架构。一个HDFS集群是由一个Namenode和一定数目的Datanodes组成。Namenode是一个中心服务器，负责管理文件系统的名字空间(namespace)以及客户端对文件的访问。集群中的Datanode一般是一个节点一个，负责管理它所在节点上的存储。HDFS暴露了文件系统的名字空间，用户能够以文件的形式在上面存储数据。从内部看，一个文件其实被分成一个或多个数据块，这些块存储在一组Datanode上。Namenode执行文件系统的名字空间操作，比如打开、关闭、重命名文件或目录。它也负责确定数据块到具体Datanode节点的映射。Datanode负责处理文件系统客户端的读写请求。在Namenode的统一调度下进行数据块的创建、删除和复制。



Namenode和Datanode被设计成可以在普通的商用机器上运行。这些机器一般运行着GNU/Linux操作系统(OS)。HDFS采用Java语言开发，因此任何支持Java的机器都可以部署Namenode或Datanode。由于采用了可移植性极强的Java语言，使得HDFS可以部署到多种类型的机器上。一个典型的部署场景是一台机器上只运行一个Namenode实例，而集群中的其它机器分别运行一个Datanode实例。这种架构并不排斥在一台机器上运行多个Datanode，只不过这样的情况比较少见。

集群中单一Namenode的结构大大简化了系统的架构。Namenode是所有HDFS元数据的仲裁者和管理者，这样，用户数据永远不会流过Namenode。

##### 文件系统的名字空间 (namespace)

HDFS支持传统的层次型文件组织结构。用户或者应用程序可以创建目录，然后将文件保存在这些目录里。文件系统名字空间的层次结构和大多数现有的文件系统类似：用户可以创建、删除、移动或重命名文件。当前，HDFS不支持用户磁盘配额和访问权限控制，也不支持硬链接和软链接。但是HDFS架构并不妨碍实现这些特性。

Namenode负责维护文件系统的名字空间，任何对文件系统名字空间或属性的修改都将被Namenode记录下来。应用程序可以设置HDFS保存的文件的副本数目。文件副本的数目称为文件的副本系数，这个信息也是由Namenode保存的。

##### 数据复制

HDFS被设计成能够在一个大集群中跨机器可靠地存储超大文件。它将每个文件存储成一系列的数据块，除了最后一个，所有的数据块都是同样大小的。为了容错，文件的所有数据块都会有副本。每个文件的数据块大小和副本系数都是可配置的。应用程序可以指定某个文件的副本数目。副本系数可以在文件创建的时候指定，也可以在之后改变。HDFS中的文件都是一次性写入的，并且严格要求在任何时候只能有一个写入者。

Namenode全权管理数据块的复制，它周期性地从集群中的每个Datanode接收心跳信号和块状态报告(Blockreport)。接收到心跳信号意味着该Datanode节点工作正常。块状态报告包含了一个该Datanode上所有数据块的列表。

###### 副本存放

副本的存放是HDFS可靠性和性能的关键。优化的副本存放策略是HDFS区分于其他大部分分布式文件系统的重要特性。这种特性需要做大量的调优，并需要经验的积累。HDFS采用一种称为机架感知(rack-aware)的策略来改进数据的可靠性、可用性和网络带宽的利用率。目前实现的副本存放策略只是在这个方向上的第一步。实现这个策略的短期目标是验证它在生产环境下的有效性，观察它的行为，为实现更先进的策略打下测试和研究的基础。

大型HDFS实例一般运行在跨越多个机架的计算机组成的集群上，不同机架上的两台机器之间的通讯需要经过交换机。在大多数情况下，同一个机架内的两台机器间的带宽会比不同机架的两台机器间的带宽大。

通过一个[机架感知](http://hadoop.apache.org/docs/r1.0.4/cn/cluster_setup.html" \l "Hadoop%E7%9A%84%E6%9C%BA%E6%9E%B6%E6%84%9F%E7%9F%A5)的过程，Namenode可以确定每个Datanode所属的机架id。一个简单但没有优化的策略就是将副本存放在不同的机架上。这样可以有效防止当整个机架失效时数据的丢失，并且允许读数据的时候充分利用多个机架的带宽。这种策略设置可以将副本均匀分布在集群中，有利于当组件失效情况下的负载均衡。但是，因为这种策略的一个写操作需要传输数据块到多个机架，这增加了写的代价。

在大多数情况下，副本系数是3，HDFS的存放策略是将一个副本存放在本地机架的节点上，一个副本放在同一机架的另一个节点上，最后一个副本放在不同机架的节点上。这种策略减少了机架间的数据传输，这就提高了写操作的效率。机架的错误远远比节点的错误少，所以这个策略不会影响到数据的可靠性和可用性。于此同时，因为数据块只放在两个（不是三个）不同的机架上，所以此策略减少了读取数据时需要的网络传输总带宽。在这种策略下，副本并不是均匀分布在不同的机架上。三分之一的副本在一个节点上，三分之二的副本在一个机架上，其他副本均匀分布在剩下的机架中，这一策略在不损害数据可靠性和读取性能的情况下改进了写的性能。

当前，这里介绍的默认副本存放策略正在开发的过程中。

###### 副本选择

为了降低整体的带宽消耗和读取延时，HDFS会尽量让读取程序读取离它最近的副本。如果在读取程序的同一个机架上有一个副本，那么就读取该副本。如果一个HDFS集群跨越多个数据中心，那么客户端也将首先读本地数据中心的副本。

###### 安全模式

Namenode启动后会进入一个称为安全模式的特殊状态。处于安全模式的Namenode是不会进行数据块的复制的。Namenode从所有的 Datanode接收心跳信号和块状态报告。块状态报告包括了某个Datanode所有的数据块列表。每个数据块都有一个指定的最小副本数。当Namenode检测确认某个数据块的副本数目达到这个最小值，那么该数据块就会被认为是副本安全(safely replicated)的；在一定百分比（这个参数可配置）的数据块被Namenode检测确认是安全之后（加上一个额外的30秒等待时间），Namenode将退出安全模式状态。接下来它会确定还有哪些数据块的副本没有达到指定数目，并将这些数据块复制到其他Datanode上。

##### 文件系统元数据的持久化

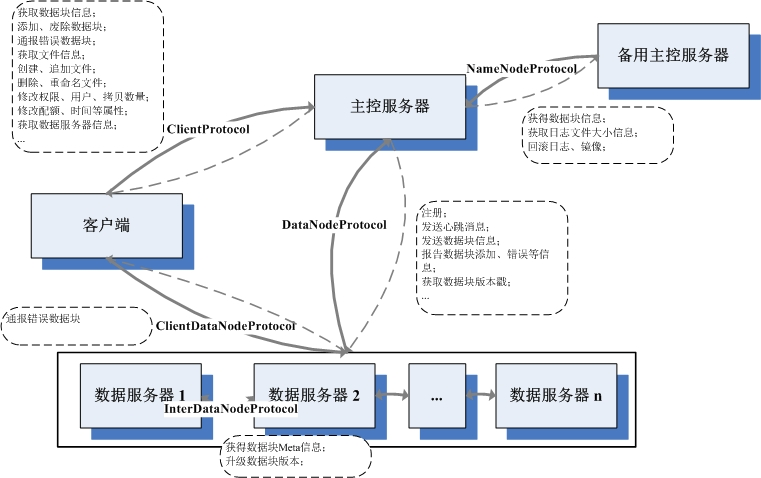
Namenode上保存着HDFS的名字空间。对于任何对文件系统元数据产生修改的操作，Namenode都会使用一种称为EditLog的事务日志记录下来。例如，在HDFS中创建一个文件，Namenode就会在Editlog中插入一条记录来表示；同样地，修改文件的副本系数也将往Editlog插入一条记录。Namenode在本地操作系统的文件系统中存储这个Editlog。整个文件系统的名字空间，包括数据块到文件的映射、文件的属性等，都存储在一个称为FsImage的文件中，这个文件也是放在Namenode所在的本地文件系统上。

Namenode在内存中保存着整个文件系统的名字空间和文件数据块映射(Blockmap)的映像。这个关键的元数据结构设计得很紧凑，因而一个有4G内存的Namenode足够支撑大量的文件和目录。当Namenode启动时，它从硬盘中读取Editlog和FsImage，将所有Editlog中的事务作用在内存中的FsImage上，并将这个新版本的FsImage从内存中保存到本地磁盘上，然后删除旧的Editlog，因为这个旧的Editlog的事务都已经作用在FsImage上了。这个过程称为一个检查点(checkpoint)。在当前实现中，检查点只发生在Namenode启动时，在不久的将来将实现支持周期性的检查点。

Datanode将HDFS数据以文件的形式存储在本地的文件系统中，它并不知道有关HDFS文件的信息。它把每个HDFS数据块存储在本地文件系统的一个单独的文件中。Datanode并不在同一个目录创建所有的文件，实际上，它用试探的方法来确定每个目录的最佳文件数目，并且在适当的时候创建子目录。在同一个目录中创建所有的本地文件并不是最优的选择，这是因为本地文件系统可能无法高效地在单个目录中支持大量的文件。当一个Datanode启动时，它会扫描本地文件系统，产生一个这些本地文件对应的所有HDFS数据块的列表，然后作为报告发送到Namenode，这个报告就是块状态报告。

##### 通讯协议

所有的HDFS通讯协议都是建立在TCP/IP协议之上。客户端通过一个可配置的TCP端口连接到Namenode，通过ClientProtocol协议与Namenode交互。而Datanode使用DatanodeProtocol协议与Namenode交互。一个远程过程调用(RPC)模型被抽象出来封装ClientProtocol和Datanodeprotocol协议。在设计上，Namenode不会主动发起RPC，而是响应来自客户端或 Datanode 的RPC请求。



##### 健壮性

HDFS的主要目标就是即使在出错的情况下也要保证数据存储的可靠性。常见的三种出错情况是：Namenode出错, Datanode出错和网络割裂(network partitions)。

##### 磁盘数据错误，心跳检测和重新复制

每个Datanode节点周期性地向Namenode发送心跳信号。网络割裂可能导致一部分Datanode跟Namenode失去联系。Namenode通过心跳信号的缺失来检测这一情况，并将这些近期不再发送心跳信号Datanode标记为宕机，不会再将新的IO请求发给它们。任何存储在宕机Datanode上的数据将不再有效。Datanode的宕机可能会引起一些数据块的副本系数低于指定值，Namenode不断地检测这些需要复制的数据块，一旦发现就启动复制操作。在下列情况下，可能需要重新复制：某个Datanode节点失效，某个副本遭到损坏，Datanode上的硬盘错误，或者文件的副本系数增大。

##### 集群均衡

HDFS的架构支持数据均衡策略。如果某个Datanode节点上的空闲空间低于特定的临界点，按照均衡策略系统就会自动地将数据从这个Datanode移动到其他空闲的Datanode。当对某个文件的请求突然增加，那么也可能启动一个计划创建该文件新的副本，并且同时重新平衡集群中的其他数据。这些均衡策略目前还没有实现。

##### 数据完整性

从某个Datanode获取的数据块有可能是损坏的，损坏可能是由Datanode的存储设备错误、网络错误或者软件bug造成的。HDFS客户端软件实现了对HDFS文件内容的校验和(checksum)检查。当客户端创建一个新的HDFS文件，会计算这个文件每个数据块的校验和，并将校验和作为一个单独的隐藏文件保存在同一个HDFS名字空间下。当客户端获取文件内容后，它会检验从Datanode获取的数据跟相应的校验和文件中的校验和是否匹配，如果不匹配，客户端可以选择从其他Datanode获取该数据块的副本。

##### 元数据磁盘错误

FsImage和Editlog是HDFS的核心数据结构。如果这些文件损坏了，整个HDFS实例都将失效。因而，Namenode可以配置成支持维护多个FsImage和Editlog的副本。任何对FsImage或者Editlog的修改，都将同步到它们的副本上。这种多副本的同步操作可能会降低Namenode每秒处理的名字空间事务数量。然而这个代价是可以接受的，因为即使HDFS的应用是数据密集的，它们也非元数据密集的。当Namenode重启的时候，它会选取最近的完整的FsImage和Editlog来使用。

Namenode是HDFS集群中的单点故障(single point of failure)所在。如果Namenode机器故障，是需要手工干预的。目前，自动重启或在另一台机器上做Namenode故障转移的功能还没实现。

##### 快照

快照支持某一特定时刻的数据的复制备份。利用快照，可以让HDFS在数据损坏时恢复到过去一个已知正确的时间点。HDFS目前还不支持快照功能，但计划在将来的版本进行支持。

##### 数据组织

###### 数据块

HDFS被设计成支持大文件，适用HDFS的是那些需要处理大规模的数据集的应用。这些应用都是只写入数据一次，但却读取一次或多次，并且读取速度应能满足流式读取的需要。HDFS支持文件的“一次写入多次读取”语义。一个典型的数据块大小是64MB。因而，HDFS中的文件总是按照64M被切分成不同的块，每个块尽可能地存储于不同的Datanode中。

###### Staging

客户端创建文件的请求其实并没有立即发送给Namenode，事实上，在刚开始阶段HDFS客户端会先将文件数据缓存到本地的一个临时文件。应用程序的写操作被透明地重定向到这个临时文件。当这个临时文件累积的数据量超过一个数据块的大小，客户端才会联系Namenode。Namenode将文件名插入文件系统的层次结构中，并且分配一个数据块给它。然后返回Datanode的标识符和目标数据块给客户端。接着客户端将这块数据从本地临时文件上传到指定的Datanode上。当文件关闭时，在临时文件中剩余的没有上传的数据也会传输到指定的Datanode上。然后客户端告诉Namenode文件已经关闭。此时Namenode才将文件创建操作提交到日志里进行存储。如果Namenode在文件关闭前宕机了，则该文件将丢失。

上述方法是对在HDFS上运行的目标应用进行认真考虑后得到的结果。这些应用需要进行文件的流式写入。如果不采用客户端缓存，由于网络速度和网络堵塞会对吞估量造成比较大的影响。这种方法并不是没有先例的，早期的文件系统，比如AFS，就用客户端缓存来提高性能。为了达到更高的数据上传效率，已经放松了POSIX标准的要求。

##### 流水线复制

当客户端向HDFS文件写入数据的时候，一开始是写到本地临时文件中。假设该文件的副本系数设置为3，当本地临时文件累积到一个数据块的大小时，客户端会从Namenode获取一个Datanode列表用于存放副本。然后客户端开始向第一个Datanode传输数据，第一个Datanode一小部分一小部分(4 KB)地接收数据，将每一部分写入本地仓库，并同时传输该部分到列表中第二个Datanode节点。第二个Datanode也是这样，一小部分一小部分地接收数据，写入本地仓库，并同时传给第三个Datanode。最后，第三个Datanode接收数据并存储在本地。因此，Datanode能流水线式地从前一个节点接收数据，并在同时转发给下一个节点，数据以流水线的方式从前一个Datanode复制到下一个。

##### 可访问性

HDFS给应用提供了多种访问方式。用户可以通过[Java API](http://hadoop.apache.org/core/docs/current/api/)接口访问，也可以通过C语言的封装API访问，还可以通过浏览器的方式访问HDFS中的文件。通过WebDAV协议访问的方式正在开发中。

### 分布式计算Map/Reduce

#### 概述

Hadoop Map/Reduce是一个使用简易的软件框架，基于它写出来的应用程序能够运行在由上千个商用机器组成的大型集群上，并以一种可靠容错的方式并行处理上T级别的数据集。

一个Map/Reduce 作业（job） 通常会把输入的数据集切分为若干独立的数据块，由 map任务（task）以完全并行的方式处理它们。框架会对map的输出先进行排序， 然后把结果输入给reduce任务。通常作业的输入和输出都会被存储在文件系统中。 整个框架负责任务的调度和监控，以及重新执行已经失败的任务。

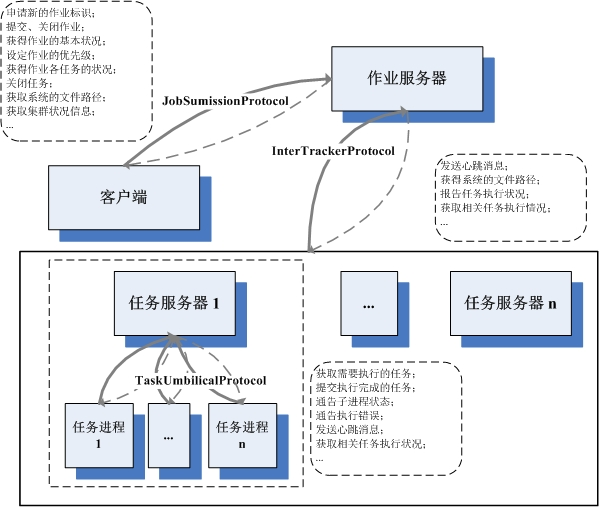
通常，Map/Reduce框架和[分布式文件系统](http://hadoop.apache.org/docs/r1.0.4/cn/hdfs_design.html)是运行在一组相同的节点上的，也就是说，计算节点和存储节点通常在一起。这种配置允许框架在那些已经存好数据的节点上高效地调度任务，这可以使整个集群的网络带宽被非常高效地利用。

Map/Reduce框架由一个单独的master JobTracker 和每个集群节点一个slave TaskTracker共同组成。master负责调度构成一个作业的所有任务，这些任务分布在不同的slave上，master监控它们的执行，重新执行已经失败的任务。而slave仅负责执行由master指派的任务。

应用程序至少应该指明输入/输出的位置（路径），并通过实现合适的接口或抽象类提供map和reduce函数。再加上其他作业的参数，就构成了作业配置（job configuration）。然后，Hadoop的 job client提交作业（jar包/可执行程序等）和配置信息给JobTracker，后者负责分发这些软件和配置信息给slave、调度任务并监控它们的执行，同时提供状态和诊断信息给job-client。

#### 通信机制

同作为Hadoop的实现，与HDFS的通信机制相同，Hadoop Map/Reduce也是用了协议接口来进行服务器间的交流。实现者作为RPC服务器，调用者经由RPC的代理进行调用，如此，完成大部分的通信，具体服务器的架构，和其中运行的各个协议状况，参见下图。从图中可以看到，与HDFS相比，相关的协议少了几个，客户端与任务服务器，任务服务器之间，都不再有直接通信关系。这并不意味着客户端就不需要了解具体任务的执行状况，也不意味着，任务服务器之间不需要了解别家任务执行的情形，只不过，由于整个集群各机器的联系比HDFS复杂的多，直接通信过于的难以维系，所以，都统一由作业服务器整理转发。

****

文件系统中很多的算法，都是很固定和直观的，不会由于所存储的内容不同而有太多的变化。而作为通用的计算框架，需要面对的问题则要复杂很多，在各种不同的问题、不同的输入、不同的需求之间，很难有一种包治百病的药能够一招鲜吃遍天。作为Map/Reduce框架而言，一方面要尽可能的抽取出公共的一些需求，实现出来。更重要的，是需要提供良好的可扩展机制，满足用户自定义各种算法的需求。Hadoop是由Java来实现的，因此通过反射来实现自定义的扩展，显得比较小菜一碟了。在JobConf类中，定义了大量的接口，这基本上是Hadoop Map/Reduce框架所有可定制内容的一次集中展示。在JobConf中，有大量set接口接受一个Class的参数，通常它都有一个默认实现的类，用户如果不满意，则可自定义实现。

#### 计算流程

如果一切都按部就班的进行，那么整个作业的计算流程，应该是作业的提交 -> Map任务的分配和执行 -> Reduce任务的分配和执行 -> 作业的完成。而在每个任务的执行中，又包含输入的准备 -> 算法的执行 -> 输出的生成，三个子步骤。

##### 作业提交

一个作业，在提交之前，需要把所有应该配置的东西都配置好，因为一旦提交到了作业服务器上，就陷入了完全自动化的流程，用户除了观望，最多也就能起一个监督作用，惩治一些不好好工作的任务。

基本上，用户在提交代码阶段，需要做的工作主要是这样的：

首先，书写好所有自定的代码，最起码，需要有Map和Reduce的执行代码。在Hadoop中，Map需要派生自Mapper接口，Reduce需要派生自Reducer接口。这里都是用的泛型，用以支持不同的键值类型。这两个接口都仅有一个方法，一个是map，一个是reduce，这两个方法都直接受四个参数，前两个是输入的键和值相关的数据结构，第三个是作为输出相关的数据结构，最后一个，是一个Reporter类的实例，实现的时候可以利用它来统计一些计数。除了这两个接口，还有大量可以派生的接口，比如分割的Partitioner接口。

然后，需要书写好主函数的代码，其中最主要的内容就是实例化一个JobConf类的对象，然后调用其丰富的setXXX接口，设定好所需的内容，包括输入输出的文件路径，Map和Reduce的类，甚至包括读取写入文件所需的格式支持类，等等。

最后，调用JobClient的runJob方法，提交此JobConf对象。runJob方法会先行调用到JobSubmissionProtocol接口所定义的submitJob方法，将此作业，提交给作业服务器。接着，runJob开始循环，不停的调用JobSubmissionProtocol的getTaskCompletionEvents方法，获得TaskCompletionEvent类的对象实例，了解此作业各任务的执行状况。

##### Map任务的分配

当一个作业提交到了作业服务器上，作业服务器会生成若干个Map任务，每一个Map任务，负责将一部分的输入转换成格式与最终格式相同的中间文件。通常一个作业的输入都是基于分布式文件系统的文件（当然在单机环境下，文件系统单机的也可以...），因为，它可以很天然的和分布式的计算产生联系。而对于一个Map任务而言，它的输入往往是输入文件的一个数据块，或者是数据块的一部分，但通常，不跨数据块。因为，一旦跨了数据块，就可能涉及到多个服务器，带来了不必要的复杂性。

当一个作业，从客户端提交到了作业服务器上，作业服务器会生成一个JobInProgress对象，作为与之对应的标识，用于管理。作业被拆分成若干个Map任务后，会预先挂在作业服务器上的任务服务器拓扑树。这是依照分布式文件数据块的位置来划分的，比如一个Map任务需要用某个数据块，这个数据块有三份备份，那么，在这三台服务器上都会挂上此任务，可以视为是一个预分配。

关于任务管理和分配的大部分的真实功能和逻辑的实现，JobInProgress则依托JobInProgressListener和TaskScheduler的子类。TaskScheduler，顾名思义是用于任务分配的策略类（为了简化描述，用它代指所有TaskScheduler的子类...）。它会掌握好所有作业的任务信息，其assignTasks函数，接受一个TaskTrackerStatus作为参数，依照此任务服务器的状态和现有的任务状况，为其分配新的任务。而为了掌握所有作业相关任务的状况，TaskScheduler会将若干个JobInProgressListener注册到JobTracker中去，当有新的作业到达、移除或更新的时候，JobTracker会告知给所有的JobInProgressListener，以便它们做出相应的处理。

任务分配是一个重要的环节，所谓任务分配，就是将合适作业的合适任务分配到合适的服务器上。不难看出，里面蕴含了两个步骤，先是选择作业，然后是在此作业中选择任务。和所有分配工作一样，任务分配也是一个复杂的活。不良好的任务分配，可能会导致网络流量增加、某些任务服务器负载过重效率下降，等等。不仅如此，任务分配还是一个无一致模式的问题，不同的业务背景，可能需要不同的算法才能满足需求。因此，在Hadoop中，有很多TaskScheduler的子类，像Facebook，Yahoo，都为其贡献出了自家用的算法。在Hadoop中，默认的任务分配器，是JobQueueTaskScheduler类。它选择作业的基本次序是：Map Clean Up Task（Map任务服务器的清理任务，用于清理相关的过期的文件和环境...） -> Map Setup Task（Map任务服务器的安装任务，负责配置好相关的环境...） -> Map Tasks -> Reduce Clean Up Task -> Reduce Setup Task -> Reduce Tasks。在这个前提下，具体到Map任务的分配上来。当一个任务服务器工作的游刃有余，期待获得新的任务的时候，JobQueueTaskScheduler会按照各个作业的优先级，从最高优先级的作业开始分配。每分配一个，还会为其留出余量，已被不时之需。举一个例子：系统目前有优先级3、2、1的三个作业，每个作业都有一个可分配的Map任务，一个任务服务器来申请新的任务，它还有能力承载3个任务的执行，JobQueueTaskScheduler会先从优先级3的作业上取一个任务分配给它，然后再留出一个1任务的余量。此时，系统只能在将优先级2作业的任务分配给此服务器，而不能分配优先级1的任务。这样的策略，基本思路就是一切为高优先级的作业服务，优先分配不说，分配了好保留有余力以备不时之需，如此优待，足以让高优先级的作业喜极而泣，让低优先级的作业感慨既生瑜何生亮，甚至是活活饿死。

确定了从哪个作业提取任务后，具体的分配算法，经过一系列的调用，最后实际是由JobInProgress的findNewMapTask函数完成的。它的算法很简单，就是尽全力为此服务器非配且尽可能好的分配任务，也就是说，只要还有可分配的任务，就一定会分给它，而不考虑后来者。作业服务器会从离它最近的服务器开始，看上面是否还挂着未分配的任务（预分配上的），从近到远，如果所有的任务都分配了，那么看有没有开启多次执行，如果开启，考虑把未完成的任务再分配一次（后面有地方详述...）。

对于作业服务器来说，把一个任务分配出去了，并不意味着它就彻底解放，可以对此任务可以不管不顾了。因为任务可以在任务服务器上执行失败，可能执行缓慢，这都需要作业服务器帮助它们再来一次。因此在Task中，记录有一个TaskAttemptID，对于任务服务器而言，它们每次跑的，其实都只是一个Attempt而已，Reduce任务只需要采信一个的输出，其他都算白忙乎了

##### Map任务的执行

与HDFS类似，任务服务器是通过心跳消息，向作业服务器汇报此时此刻其上各个任务执行的状况，并向作业服务器申请新的任务的。具体实现，是TaskTracker调用InterTrackerProtocol协议的heartbeat方法来做的。这个方法接受一个TaskTrackerStatus对象作为参数，它描述了此时此任务服务器的状态。当其有余力接受新的任务的时候，它还会传入acceptNewTasks为true的参数，表示希望作业服务器委以重任。JobTracker接收到相关的参数后，经过处理，会返回一个HeartbeatResponse对象。这个对象中，定义了一组TaskTrackerAction，用于指导任务服务器进行下一步的工作。系统中已定义的了一堆其TaskTrackerAction的子类，有的对携带的参数进行了扩充，有的只是标明了下ID，具体不详写了，一看便知。

当TaskTracker收到的TaskTrackerAction中，包含了LaunchTaskAction，它会开始执行所分配的新的任务。在TaskTracker中，有一个TaskTracker.TaskLauncher线程（确切的说是两个，一个等Map任务，一个等Reduce任务），它们在痴痴的守候着新任务的来到。一旦等到了，会最终调用到Task的createRunner方法，构造出一个TaskRunner对象，新建一个线程来执行。对于一个Map任务，它对应的Runner是TaskRunner的子类MapTaskRunner，不过，核心部分都在TaskRunner的实现内。TaskRunner会先将所需的文件全部下载并拆包好，并记录到一个全局缓存中，这是一个全局的目录，可以供所有此作业的所有任务使用。它会用一些软链接，将一些文件名链接到这个缓存中来。然后，根据不同的参数，配置出一个JVM执行的环境，这个环境与JvmEnv类的对象对应。

接着，TaskRunner会调用JvmManager的launchJvm方法，提交给JvmManager处理。JvmManager用于管理该TaskTracker上所有运行的Task子进程。在目前的实现中，尝试的是池化的方式。有若干个固定的槽，如果槽没有满，那么就启动新的子进程，否则，就寻找idle的进程，如果是同Job的直接放进去，否则杀死这个进程，用一个新的进程代替。每一个进程都是由JvmRunner来管理的，它也是位于单独线程中的。但是从实现上看，这个机制好像没有部署开，子进程是死循环等待，而不会阻塞在父进程的相关线程上，父线程的变量一直都没有个调整，一旦分配，始终都处在繁忙的状况了。

真实的执行载体，是Child，它包含一个main函数，进程执行，会将相关参数传进来，它会拆解这些参数，并且构造出相关的Task实例，调用其run函数进行执行。每一个子进程，可以执行指定个数量的Task，这就是上面所说的池化的配置。但是，这套机制在我看来，并没有运行起来，每个进程其实都没有机会不死而执行新的任务，只是傻傻的等待进程池满，而被一刀毙命。也许是我老眼昏花，没看出其中实现的端倪

##### Reduce任务的分配与执行

比之Map任务，Reduce的分配及其简单，基本上是所有Map任务完成了，有空闲的任务服务器，来了就给分配一个Job任务。因为Map任务的结果星罗棋布，且变化多端，真要搞一个全局优化的算法，绝对是得不偿失。而Reduce任务的执行进程的构造和分配流程，与Map基本完全的一致，没有啥可说的了。

但其实，Reduce任务与Map任务的最大不同，是Map任务的文件都在本地隔着，而Reduce任务需要到处采集。这个流程是作业服务器经由此Reduce任务所处的任务服务器，告诉Reduce任务正在执行的进程，它需要的Map任务执行过的服务器地址，此Reduce任务服务器会于原Map任务服务器联系（当然本地就免了...），通过FTP服务，下载过来。这个隐含的直接数据联系，就是执行Reduce任务与执行Map任务最大的不同了

##### 完成作业

当所有Reduce任务都完成了，所需数据都写到了分布式文件系统上，整个作业才正式完成了。此中，涉及到很多的类，很多的文件，很多的服务器，所以说起来很费劲，话说，一图解千语，说了那么多，我还是画两幅图，彻底表达一下吧。

首先，是一个时序图。它模拟了一个由3个Map任务和1个Reduce任务构成的作业执行流程。我们可以看到，在执行的过程中，只要有人太慢，或者失败，就会增加一次尝试，以此换取最快的执行总时间。一旦所有Map任务完成，Reduce开始运作（其实，不一定要这样的...），对于每一个Map任务来说，只有执行到Reduce任务把它上面的数据下载完成，才算成功，否则，都是失败，需要重新进行尝试

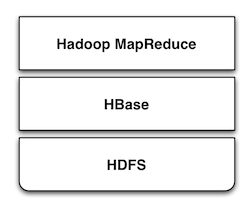
#### 应用场景

  诸如水控、电控、无线wifi类时时刻刻都在产生数据，数据增长速度较快、数据容量较大的数据可以采用分布式存储及分布式计算的方式进行，在学校大数据建设过程中可以付诸实施。

### 分布式数据库Hbase

#### 简介

HBase是Apache Hadoop中的一个子项目，Hbase依托于Hadoop的HDFS作为最基本存储基础单元，通过使用hadoop的DFS工具就可以看到这些这些数据 存储文件夹的结构,还可以通过Map/Reduce的框架(算法)对HBase进行操作，如图所示：



#### 数据模型

HBase 不同于一般的关系数据库,它是一个适合于非结构化数据存储的数据库.所谓非结构化数据存储就是说HBase是基于列的而不是基于行的模式，这样方面读写你的大数据内容。

HBase是介于Map Entry(key & value)和DB Row之间的一种数据存储方式。就点有点类似于现在流行的Memcache，但不仅仅是简单的一个key对应一个 value，你很可能需要存储多个属性的数据结构，但没有传统数据库表中那么多的关联关系，这就是所谓的松散数据。

简单来说，你在HBase中的表创建的可以看做是一张很大的表，而这个表的属性可以根据需求去动态增加，在HBase中没有表与表之间关联查询。你只需要 告诉你的数据存储到Hbase的那个column families 就可以了，不需要指定它的具体类型：char,varchar,int,tinyint,text等等。但是你需要注意HBase中不包含事务此类的功 能。

Apache HBase 和Google Bigtable 有非常相似的地方，一个数据行拥有一个可选择的键和任意数量的列。表是疏松的存储的，因此用户可以给行定义各种不同的列，对于这样的功能在大项目中非常实用，可以简化设计和升级的成本。

#### HBase优缺

###### 优点

1 列的可以动态增加，并且列为空就不存储数据,节省存储空间。  
2 Hbase自动切分数据，使得数据存储自动具有水平scalability。  
3 Hbase可以提供高并发读写操作的支持。

###### 缺点

1 不能支持条件查询，只支持按照Row key来查询.

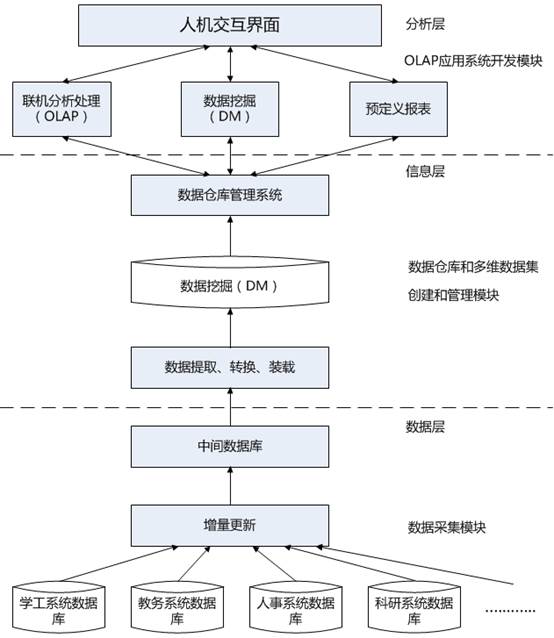
2 暂时不能支持Master server的故障切换,当Master宕机后,整个存储系统就会挂掉.

#### 应用场景

    对于传统数据库，增加列对于一个项目来讲，改变是非常大的。但是对于nosql，插入列和删除列，跟传统数据库里面的增加记录和删除记录类似。非常符合学校高级表的实现。

## 数据仓库与数据联机分析

### 数据挖掘系统关键技术



系统总体架构分为三层：

1. 数据层

数据挖掘管理系统最底层是数据层，数据层是构建数据仓库的基础。数据层包含各种数据源如学工系统、教务系统、人事管理系统、科研管理系统、一卡通等所有学校在用信息系统的数据库。通过建立中间数据库，将OLTP数据库上与数据仓库有关的数据增量更新到中间数据库，使对OLTP数据库的效率影响降低到最低。

1. 信息层

为了解决教学决策及评估等主题中需求目标难以量化的特点，信息层作为数据访问层和源数据层的桥梁，对数据进行处理，形成不同层次的信息，以便分析层在此基础上建模。信息层的核心就是与各主题相关的数据仓库。数据仓库通过对数据源的原始数据进行收集、分类、筛选和抽取形成能够对决策分析提供支持的多重粒度级的数据库，实现对数据源的统一管理和信息转换。同时为分析层的各种应用提供接口，构造应用于主题分析的更高粒度级的数据集市，或者直接为数据访问层提供数据支持。信息层通过管理工具对数据仓库进行管理和维护。以中间数据库为数据源，按分析主题建立数据仓库。最后将数据仓库中的数据按照一定方式聚合汇总，构成信息分析的多维视图，以一定的存储模式（如模式文件）将这些多维视图存储在OLAP服务器中。

1. 分析层

分析层由通用分析平台和基于主题的应用分析系统组成。通用分析平台是为了适应决策需求的不稳定的特点，提供一些基本数据挖掘手段和简单分析模型，这样在很多情况下不必专门的建立应用分析系统，大大提高了系统的普遍适用性和灵活性。针对复杂且相对稳定的决策需求，需要从数据仓库抽取数据并建立专题数据集市或是利用已有的数据集市，开发专用的分析系统。联机分析处理模块实现对多维数据集中立方体信息、维度信息、层次信息、度量信息的显示；实现钻取、切片、切块，旋转等多维分析操作，提供多种查询方式，查询结果以WEB页面的形式展现，以实现B/S形式的联机分析，全面支持决策分析。

三层结构化的体系结构适应决策挖掘系统需求的特点。三层之间通过数据仓库衔接，有效实现了数据驱动，并且任何一层的修改、添加或删除功能模块给整体系统的影响都很小。

#### 数据仓库技术

**简介：**

   数据仓库综合和合并多维空间的数据.构造数据仓库涉及数据清理,数据集成和数据变换,可以看作数据挖掘的一个重要预处理步骤.

    此外,数据仓库提供联机分析处理(OLAP)工具,用于各种粒度的多维数据的交互分析,有利于有效的数据泛化和数据挖掘.

    构造数据仓库和OLAP已经成为知识发现过程的基本步骤.

       数据仓库中的数据时进行决策分析的基础，因此构造数据仓库是构建系统的首要任务。在数据仓库中，由于注重的是查询而不是事务，因此数据是按照决策分析的学生主题、教师主题、科研主题、一卡通主题、资产主题、设备主题等来组织。为了使数据仓库中的数据是面向对象的，必须将数据按要求的主题进行收集、归纳后重新存放；由于需要能进行多视角观察、分析数据，因此，对应各主题的数据必须按一定得属性进行组织；另外，针对数据的综合程度不同，应该对数据按不同的汇总程度存放。因此，数据仓库采用多维数据模型对各主题的数据进行组织。对应以上数据组织的要求，数据仓库定义了相应的结构元素来实现数据的结构模型：事实、维、粒度等。

         由于数据仓库中的数据是静态的，很少对其进行插入或更新操作，而且用数据仓库中的数据创建立方体十分耗费时间，所以要建立较多的索引以加快信息检索和更新立方体的速度。

         抽取与加载数据：数据仓库建立之后，就可以从信息系统中加载数据，这需要将多个数据转移到数据仓库。由于不同数据源的数据存在很大的异构性，因此在数据加载过程中，需要完成数据库验证，数据迁移，数据净化和数据转换等使数据一致，集成的操作。这一过程将通过ETL工具ODI来完成。

**面向主题的**

    是相对于传统数据库的面向应用而言的。所谓面向应用，指的是系统实现过程中主要围绕着一些应用或功能。而面向主题则考虑一个个的问题域，对问题域涉及到的数据和分析数据所采用的功能给予同样的重视。

**集成的**

 最重要的特点。应用问题的设计人员制定出不同的设计决策，且表示方法不同。例如编码、命名习惯、实际属性和属性度量等方面不一致。

    数据进入数据仓库时，需要消除各种不一致性。例如，数据仓库中顾客“性别”的编码，可采用“男/女”或“m/f”，采用哪种方式并不重要，重要的是在数据仓库中应该统一编码。如果应用数据编码为“X/Y”，则进入数据仓库时需要进行转换。此外，对所有应用所涉及的问题都要考虑一致性。例如命名习惯、键码结构、属性度量以及数据特点等。

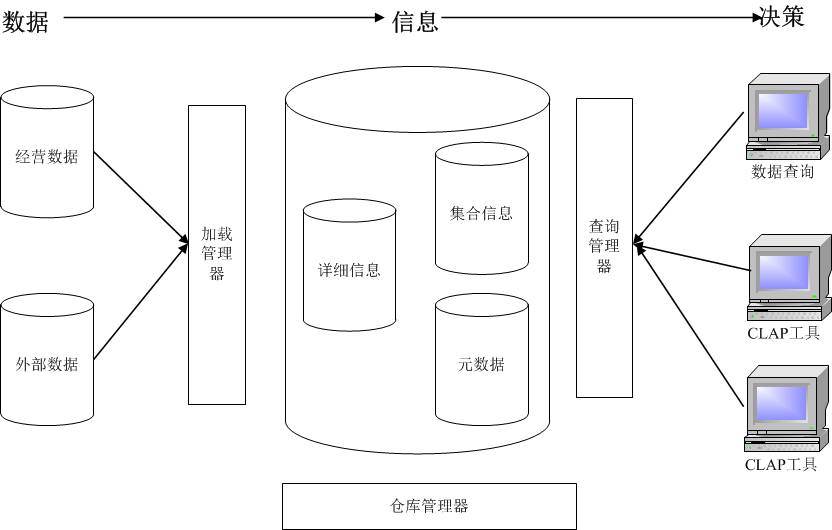
**与时间相关的**

 数据仓库以维的形式对数据进行组织，时间维是数据仓库中很重要的一个维度。并且数据仓库中的数据时间跨度大，从几年甚至到几十年，称为历史数据。

**不可修改的**

面向应用的事务数据库需要对数据进行频繁的插入、更新操作，而对于数据仓库中数据的操作仅限于数据的初始导入和记录查询。

操作型数据是一次访问和处理一个记录，可以对操作型数据库中的数据进行更新。但数据仓库中的数据则不同，通常是一起载入与访问的，在数据仓库环境中并不进行一般意义上的数据更新。



**数据模型**

数据模型是对现实世界的一种抽象，根据抽象程度的不同，可形成不同抽象层次上的数据模型。与数据库的数据模型相类似，数据仓库的数据模型也分为三个层次：

* 概念模型

客观世界到计算机系统的一个中间层次，它最常用的表示方法是E－R法（实体－关系）。

目前，数据仓库一般是建立在关系型数据库的基础之上，所以其概念模型与一般关系型数据库采用的概念模型相一致。

* 逻辑模型

指数据的逻辑结构，如多维模型、关系模型、层次模型等。数据仓库的逻辑模型描述了数据仓库的主题的逻辑实现，即每个主题对应的模式定义。

* 物理模型

是逻辑模型的具体实现，如物理存取方式、数据存储结构、数据存放位置以及存储分配等。在设计数据仓库的物理模型时，需要考虑一些提高性能的技术，如表分区，建立索引等。

目前，对数据仓库模型的讨论大多集中在逻辑模型，其中最常用的是多维模型。

* 维

人们观察数据的特定角度。例如，企业常常关心不同销售数据随时间的变化情况，所以时间就是一个维；

* 维的层次

人们观察数据的某个特定角度还可以存在细节程度不同的多个描述，这就是维的层次。一个维往往有多个层次，比如描述时间维时，可以从年份、季度、月份、天等不同层次来描述，那么年份、季度、月份和天就是时间维的层次；

* 维成员

维的一个取值称为该维的一个成员。如果一个维是多层次的，那么该维的成员就是在不同层次取值的组合。例如时间维分为年、月和天三个层次则分别在其上各取一个值组合起来即得到日期维的一个成员，即“某年某月某日”；

* 度量

描述了要分析的数值，例如话费、用户数量等；

* 粒度

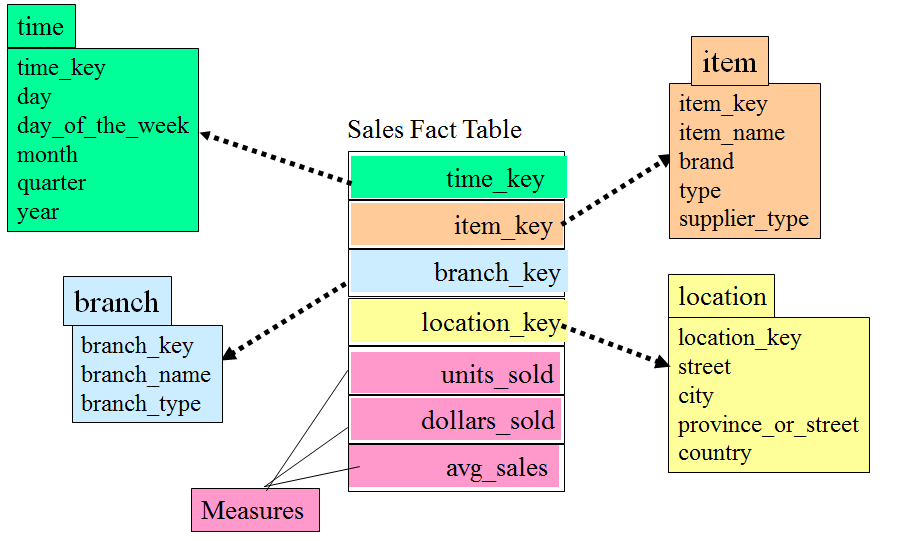
粒度是指数据仓库的数据单位中保存数据的细化或综合程度的级别。细化程度越高，粒度级就越小;相反，细化程度越低，粒度级就越大。

粒度深深地影响存放在数据仓库中数据量的大小，同时影响数据仓库所能回答的查询类型。

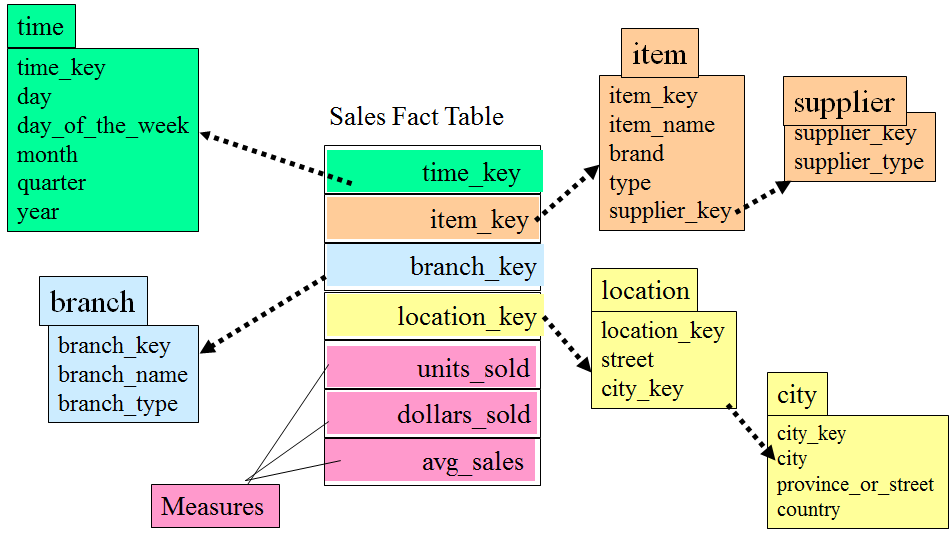
在数据仓库中的数据粒度与查询的详细程度之间要做出权衡。

* **星型结构**

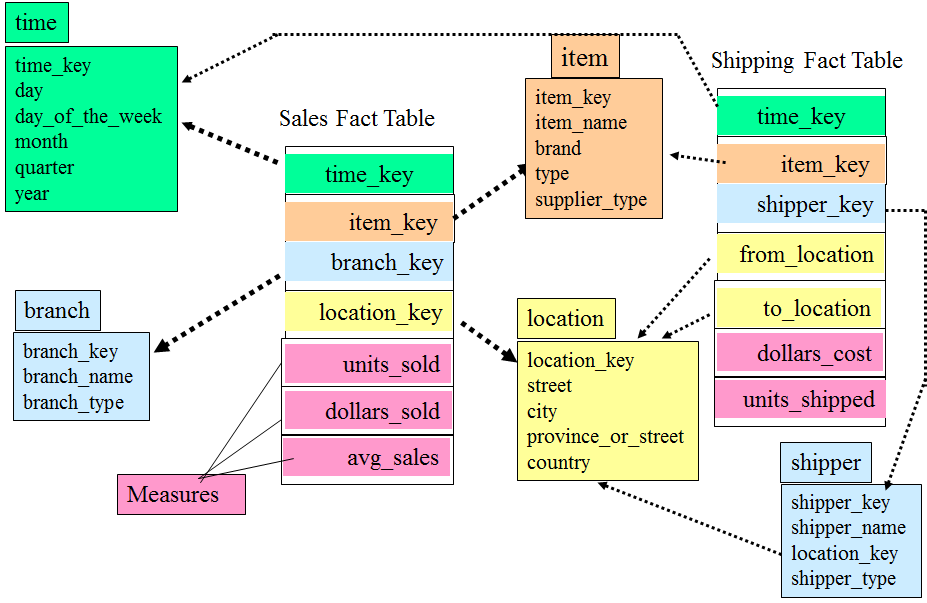
星型架构的维度表只会与事实表生成关系，维度表与维度表之间并不会生成任何的关系。雪花型架构描述可合并在一起使用的维度数据。



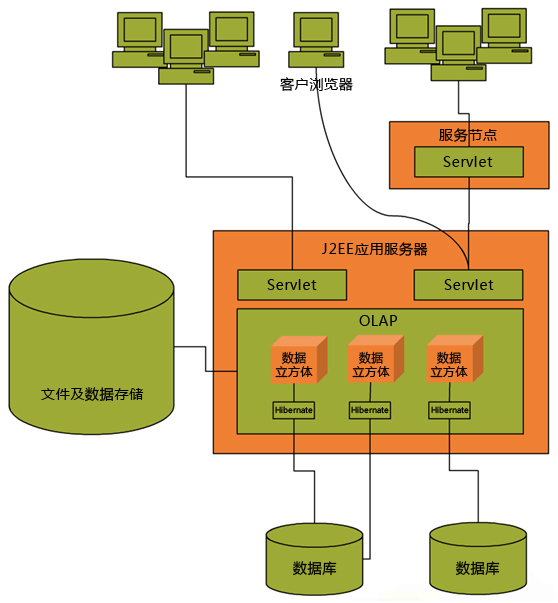
* **雪花型结构**



* **星型雪花型结构**



#### 联机分析技术OLAP



         联机分析处理OLAP是针对特定问题的联机数据访问。分析人员、管理人员通过对多维数据的很多可能的观察形势进行快速、稳定一致和交互性的存取，允许管理决策人员对数据进行深入观察。

1. 维

维是人们观察数据的特定角度，是考虑问题时的一类树形，树形集合构成一个维。例如，教学质量监督部门关系教学质量随时间推移而产生的变化情况，此时时间就是一个维，即时间维。

1. 维的层次

人们观察数据的某个特定角度还可以存在细节程度不同的多个描述方面，我们称这多个描述方面为维的层次。例如描述时间维，可以从学期、学年等不同的层次来描述，那么学期、学年等就是时间维的层次。

1. 维成员

维的一个取值。例如，学生对象这个维，这个维有院系、专业、年级、学生类型这四个层次，分别在四个层次上取一个值组合起来，就得到了开课对象维的一个成员，即“信息科学与技术学院计算机应用技术专业2005级硕士生”

1. 多维数组

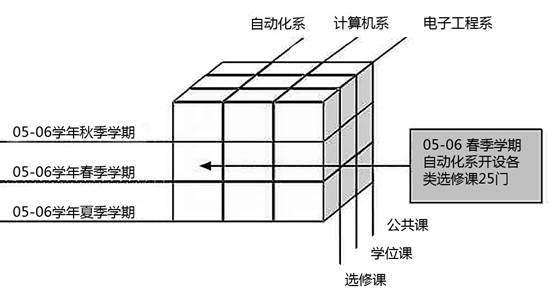
维1，维2，……，维n，变量。例如按照课程编号、院系、专业、年级组织起来的思维结构，加上学分，就组成了一个多维数组（课程编号，对象院系、对象专业、对象年级、学分）。

1. 数据单元

多维数据的取值。当多维数组的各个维都选中一个维成员，这些维成员的组合就确定了唯一的变量值。

1. OLAP立方

立方代表数据，立方的维代表事实表中的维。典型的数据立方体如图所示。



1. 切片、

在多维数据的各维上分别选定某个对应维成员的操作称为切片操作，即在确定每个维的成员后，所得的多维数组的子集称为是对其的一个切片。欺骗操作如下图所示。



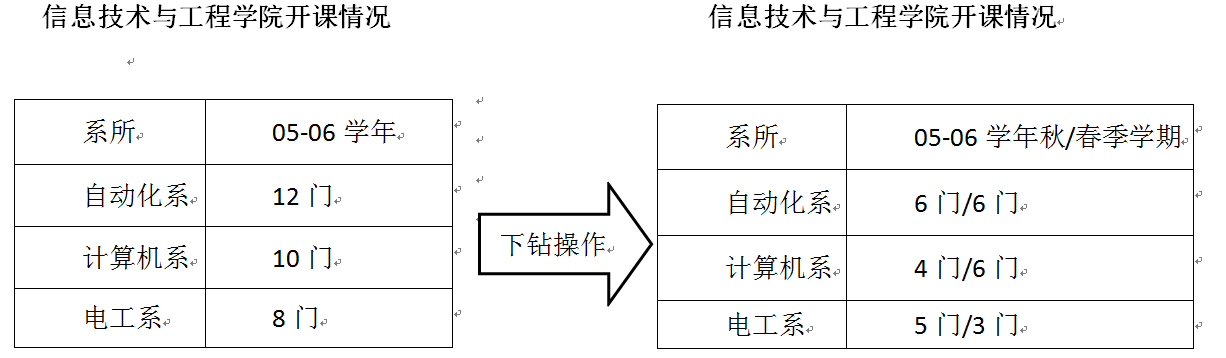
1. 旋转

改变一个报表或页面显示的维方向的操作。如可以交换行和列，或把某一行维移到列维去，或把页面显示中的一个维和页面外的维进行交换。旋转操作如图所示。



1. 钻取

钻取操作是使用户在数据仓库的多层数据中能通过导航信息而获得更多的细节数据。通常包括下钻和上翻两种操作。



联机分析处理可以快速的、多维、更高层次、全局的角度的分析来展示全校全方位的数据。

数据挖掘（DM）就是从大量的、不完全的、有噪声的、模糊的、随机的实际应用数据中，提取隐含在其中的、人们视线不知道的、但是又是潜在的有用信息和知识的过程。

由于学校现有业务系统大多为结构化的数据，数据挖掘技术方法主要采用关联分析、聚类分析、分类、预测、时序模式和偏差分析等。

1. 关联分析（AssociationAnalysis）

两个或两个以上数据项的取值之间存在某种规律性，就称为关联，可以建立起这些数据项的关联规则。数据关联是数据库中存在的一类重要的、可被发现的知识，它反映一个事件和其他事件之间的依赖和关联。如果两项或多项树形之间存在关联，那么其中一项的属性值就可以依据其他属性值进行预测。

1. 聚类分析（Clustering）

聚类是把数据按照它们的相似性归纳成若干类别，同一类别中的数据距离较小、彼此相似，不同类别中的数据距离偏大、彼此相异聚类分析可以建立宏观的概念，发现数据的分布模式，以及可能的数据树形之间的相关关系。在聚类分析法中可以给定距离阈值，各样本按阈值进行聚类。

1. 分类（Classification）

分类是数据挖掘中应用最多的任务。分类就是找出一个类别的感念描述，并用这种描述来构造模型。类别的概念描述代表着这类数据的整体信息，也就是该类的内涵描述。类的内涵描述分为：特征描述和辨别性描述。特征描述是对类中对象的共同特征的描述。辨别性描述是对两个或多个类之间的区别的描述。

1. 预测（Predication）

预测是利用历史数据找出变化规律，建立模型，并由此模型对未来数据的种类及特征进行预测，典型的预算方法是回归分析，即利用大量的历史数据，以时间为变量建立线性或非线性回归方程。预测时，只要输入任意的时间值，通过回归方程就可求出该时间的状态。

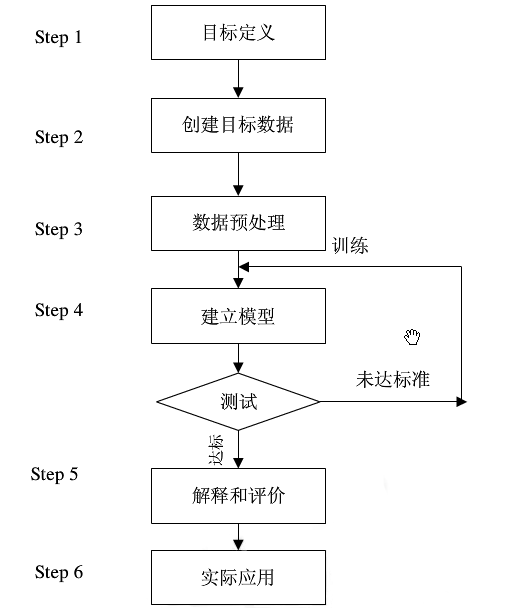
1. 时序模式（Time-SeriesPattern）

时序模式是指通过时间序列搜索出的重复发生概率较高的模式。与回归一样，它也是用一致的数据预测未来的值，但这些数据的区别是变量所处的时间不同。在时序模式中，需要找出在某个最小时间内出现比率一直高于某一最小百分比的规则。这些规则会随着形势的变化作适应的调整。时序模式中，一个有重要影响的方法是“相似时序”。用“相似时序”的方法，要按时间顺序查看时间事件数据库，从中找出另一个或多个相似的时序事件。

1. 偏差分析（Deviation）

数据库中的数据存在很多异常情况，发现数据库中数据存在的异常情况是非常重要的。偏差包括很多潜在的知识，如分类中的反常实例、不满足规则的特例、观察结果与模型预测值的偏差、量值随时间的变化等。偏差检测的基本方法是，寻找观察结果与参照值之间有意义的差别。

         数据挖掘的基本步骤可以粗略分为问题定义，数据选取和处理，数据挖掘，结果的解释和评估以及最后实际应用。如下图



1. 目标定义

数据挖掘是为了在大量数据中发现有用的令人感兴趣的信息。在问题定义过程中，数据挖掘人员和最终用户紧密协作，一方面明确实际工作对数据挖掘的要求；另一方面通过对各种学习算法的对比，今儿确定可用的学习算法。后续的学习算法选择和数据集准备都是在此基础上进行的。

1. 创建目标数据

在一个或多个知识发现工具的帮助下，选择一组要进行分析的初始数据。

1. 数据预处理

数据预处理是数据挖掘过程中的一个重要步骤，所花的时间和精力要占整个数据挖掘项目的50%-90%，尤其是在对包含有噪声、不完整，甚至是不一致数据进行数据挖掘时，更需要进行数据的预处理，以提高数据挖掘对象的质量，并最终达到提高挖掘所获模式知识的质量。

数据预处理分为4个步骤：数据清洗（Data Cleaning）、数据集成（DataIntegration）、数据转换（Data Transformation）、数据消减（Data Reduction）。

1. 建立模型

通过应用一个或多个数据挖掘算法来创建表示数据的最佳模型，数据挖掘算法执行阶段首先根据对问题的定义明确挖掘的任务或目的。确定挖掘任务后选择算法，进而建立模型，在多个可供选择的模型中找出最佳模型。

1. 结果解释和评估

数据挖掘阶段发现出来模式，经过评估可能存在冗余或无关的模式，这时需要将其剔除，也有可能模式下不满足用户要求，这时则需要回退到发现过程的前面阶段，如重新选取数据，采用新的数据变换方法，设定新的参数值，甚至换一种挖掘算法等。另外，数据挖掘由于最终是面向人类用户的，因此可能要对发现的模式进行可视化，或者把结果转换为用户易懂的另一种表示，如把分类决策树转换为“IF…THEN…ELSE …THEN…”规则。

1. 实际应用

如果发现的知识被认为是有用的，根据模型的解释和评估结果对实际要解决的问题进行实践指导。

#### 高速检索技术

         由于数据仓库中的数据非常大，而且分析性查询操作也比较复杂，因此，提高数据仓库中数据的访问速度称为关键要求和必然之举。系统采用基于数据仓库的索引技术、内存管理和并行控制来保证数据检索的快速便捷。

我们主要采用索引技术方法来解决数据快速检索的问题。

1. B树索引

B树索引是一种动态调节的平衡树，较适合字段值分散且重复值少的字段。一个B树索引中包含一个由高层节点和相继低层结点组成的层次结构。B树索引有两类结点：一是分支节点；二是叶子节点。B树索引具有间接性、易维护性以及高速检索的特性。

1. 位图索引

位图索引是一组0和1组成的字符，在表的任何一列中，每一种可能存在的值都有它唯一的位图。表中的每一条记录都会分配一个位图中的相对位置，用来表明所考虑的记录是否出现了不同的值。下面通过一个例子来说明：

|  |  |
| --- | --- |
| 男 | 女 |
| 0 | 1 |
| 1 | 0 |
| 1 | 0 |
| 0 | 1 |

表1 学生信息表                          表2 位图索引值表

|  |  |
| --- | --- |
| 姓名 | 性别 |
| 莉莉 | 女 |
| 张三 | 男 |
| 李四 | 男 |
| 王燕 | 女 |

1. 链接索引

链接索引的索引项是链接的并且满足链接要求的表的元组标志，它的每一个元组包括所有要链接的表的元组标志。链接索引的显著特点是它能提高多表链接的性能，它能够把所有符合链接条件的链接记录下来。

索引的优势：

1. 建立索引可以在很大程度上提高信息检索的效率。
2. 创建索引后，可以使索引和信息形成一对一的映射式对应关系，从而使数据具有唯一性。
3. 能够提高表与表之间的链接速度，提高了数据的参考完整性。
4. 可以减少检索过程中的分组及排序时间。
5. 使用优化隐藏器，提高整个信息检索挖掘的性能。

## 数据集成交换

高校如今经过多年信息化建设的积累，加上近几年的数字校园建设，目前已建成并投入使用的各类业务和管理信息系统，涵盖教务、学工、人事、劳资、科研、后勤（一卡通）、财务、OA、迎新、图书馆、实验室、教学档案、网络教学、毕业论文、工作量统计等方面。

通过数字校园的建设，已通过三大平台（数据中心、统一身份认证、门户系统）在大部分管理系统之间，实现了数据集成（数据平台）和应用集成（门户平台和身份认证集成），初步实现了系统间的互联互通，在一定程度上消除了信息孤岛的现象，大部分基础数据已经可以通过数据集成平台进行积累和沉淀，并通过数据平台对外共享，通过集成门户进行集中展现。目前已经初步具备了大规模应用数据挖掘和信息进一步加工利用（信息融合、大数据、高价值应用等）的基础。数据集成过程中需要使用到的技术及工具如etl工具kettle或者odi

### 集成中心库模式

模式以教育部《高等学校管理信息标准CELTS-33》为基础，结合精华教育过去多年集成经验进行设计。从总体上对高校数据进行梳理，在高校信息管理的数据层面按照面向对象进行设计。

高校往往将数据信息按照不同角度不同侧重点，分散在各个业务系统中进行管理，而集成中心库模式则将这些数据信息抽离出来，进行对象和业务类的拆分，将其分为对象类和活动类。

对象类是指数据对象，主要包括实体对象和逻辑对象。实体对象是指传统的数据对象，如人、活动等；而逻辑对象则是指一些非传统的数据对象，如院系信息等。

活动类包括对象管理活动和综合管理活动，可以按照对象的生命周期进行管理活动的设定。其中对象管理活动指基于对象的管理部分活动可以进行清晰的划分和归纳的数据，比如一些对实体对象的管理，；综合管理活动是指包含的内容和相关联系会随着应用、定位以及数据使用的角度不同而变化的活动。

### 参照模型标准及数据

参照标准集是一系列机构标准的集合，主要包括国家标准、教育部行业标准等。

### 执行标准模型及数据

执行标准集是学校自定义标准的集合，俗称“校标”，是学校信息化建设中实际使用的代码标准集。一般情况下，执行标准是高校参考了参照标准，再结合学校自身的需求而制定的。

### 集成接口支持

数据集成平台为多种数据源提供接口支持，如下：

* 支持主流关系型数据库，包括Oracle、IBM DB2 UDB、IBM DB2/400、Informix、Microsoft SQL Server、Sybase AS Enterprise、Sybase AS Anywhere。
* 支持非主流关系型数据库，包括MySQL、Derby、Hypersonic SQL、PostgreSQL。
* 支持ODBC数据源类型的接入，包括Microsoft Access、Microsoft Excel、Dbase、Visual Foxpro。
* 支持主题或者队列，包括JMS Topic、JMS Queue。
* 支持WebService
* 支持Tabled-Txt文件
* 支持XML文件
* 支持操作系统的网络协议，包括 FTP。

### 数据集成KM模块

数据集成平台提供数据集成KM库，定义了各种场景通用的知识模块（KM），包括了各种数据集成需求开发包共计100多项，大大减少了数据的集成的工作量。

### 集成设计工具

集成设计工具是设计阶段开发人员和元数据管理员的主要工具，系统所有项目的开发都是通过这个界面完成的。

用户使用该工具，为数据转换和数据完整性定义声明规则，然后将业务系统中数据库中的元数据导入或定义，为实际的生产环境生成方案。

### 集成查看工具

集成查看工具是运行阶段的主要工具，开发人员在设计阶段也可使用该工具进行调试。

用户使用该工具查看执行日志，包括错误数、被处理的行数、执行的统计信息和被执行的实际代码等等，实现对生产环境的管理以及ODI监控。

### 集成调度工具

用户通过集成调度工具实现对各个数据集成同步任务的调度控制，以此完成定制化的数据集成过程。

### 应用场景

第三方业务系统与数据中心库及分析基础库的集成工作都需要数据集成交换工具或平台的参与。

## 数据可视化技术 ECHARTS

### 混搭

混搭的图表会更具表现力也更有有趣味，ECharts提供的图表（共11类17种）支持任意混搭：折线图（面积图）、柱状图（条形图）、散点图（气泡图）、K线图、

饼图（环形图）、雷达图、地图、和弦图、力导布局图、仪表盘、漏斗图。

混搭情况下一个标准图表：包含唯一图例、工具箱、数据区域缩放、值域漫游模块，一个直角坐标系（可包含一条或多条类目轴线，一条或多条值轴线，最多上下左右四条）

### 拖拽重计算

拖拽重计算特性（专利）带来了数据统计图表从未有过的用户体验，允许用户对统计数据进行有效的提取、整合，甚至在多个图表间交换数据，赋予了用户对数据进行挖掘、整合的能力。

### 数据视图

如果你所呈现的数据足够让用户所关心，那么他们将不满足于查看可视化的图表，要去逐一迎合他们下载保存，数据分享，加工整合已有数据等等需求,如果你的用户足够的高端，你甚至可以打开数据视图的编辑功能，跟拖拽重计算相比，这可是批量的数据修改！

### 动态类型切换

很多图表类型本身所表现的能力是相似的，但由于数据差异、表现需求和个人喜好的不同导致最终图表所呈现的张力又大不一样，比如折线图和柱状图的选择,系列数据是堆叠还是平铺总是让人头疼。ECharts提供了动态类型切换，让用户随心所欲的切换到他所需要的图表类型和堆叠状态。

### 图例开关

多系列数据的同时展现呈现出丰富内容，但如何让用户切换到他所关心的个别系列上,ECharts提供了方便快捷的多维度图例开关，可以随时切换到你所关心的数据系列。

### 数据区域选择

数据可以是无限的，但显示空间总是有限的，数据区域选择组件提供了大数据量中漫游的能力，让用户选择并呈现他所关心的数据区域。配合随动的均值（极值）标线，标注展现强大的数据剖析能力

### 多图联动

多系列数据在同一个直角系内同时展现有时候会产生混乱，但他们又存在极强的关联意义不可分离,ECharts提供了多图联动的能力（connect），能做的可不仅仅是鼠标划过的详情显示，连接的多个图表会共享组件事件并且实现了保存图片时的自动拼接。

### 值域漫游

基于坐标的图表（如地图、散点图）通过色彩变化表现数值的大小能直观形象的展示数据分布。

### 大规模数据模式

如何展现成千上百万的数据？貌似除了用专业的统计工具（如MATLAB）外别无选择，

在拥有如此多交互特性下ECharts依然可以做到直角系图表（折、柱、散点、K线）20万数据秒级内渲染完成，这对于常规的应用，用现代浏览器就足以轻松展现百万规模的数据！

### 动态数据添加

如果你需要展示有实时变化的数据，相信这个动态接口会对你很有帮助。

### 标线辅助

趋势线、平均线、上升通道、支持位，提供标线辅助在K线图中可是必要的功能！ ECharts中的任何图表都可以使用

### 多维度堆积

支持多系列，多维度的数据堆积，配合自动伸缩的图形实体和直角坐标系，能呈现出更有内涵的统计图表~

### 标注 & 标线

添加额外的标注内容是常用的功能，如地图上标注某些特定位置，折线图上标注极值点或者柱形图里标线出变化趋势，ECharts全系列图表支持标注标线功能，并且与生俱来的可以响应图例开关、值域漫游等组件的交互功能。

### 个性化定制

超过600个可配置选项结合多级控制设计满足高度定制的个性化需求。

### 事件交互

可以捕获的用户交互和数据变化事件实现图表间或者与外界的联动

### 百搭时间轴

时空数据分析是信息可视化里一个相当重要的方向！ECharts提供可与任意图表搭配使用的时间轴控件以展现时空数据变化。

## ORACLE数据缓存技术

Oracle数据库的运行速度与效率，在同类数据库中是名列前茅的，特别是对大量数据进行访问时，更加有出色的表现。Oracle数据库作为复杂运算的首选数据库，其首先是通过所谓的数据高速缓存来实现对数据的高速运算与操作的。

数据高速缓存跟操作系统的缓存类似，其存储最近从数据文件中读取的数据块，其中的数据可以被所有的用户所访问。如当我们利用Select语句从数据库中查询员工信息的时候，其首先不是从数据文件中去查询这个数据，而是从数据高速缓存中去查找，而没有这个必要再去查询磁盘中的数据文件了。只有在数据缓存中没有这个数据的时候，数据库才会从数据文件中去查询。这是由于数据库在读取数据的时候，读取内存的速度比读取磁盘的速度要快很多倍，所以这种机制可以提高数据的整体访问效率。

### 空闲缓存块

当我们重新启动数据库后，系统就会为数据库分配一些空闲的缓存块。空闲缓存块中是没有任何数据的，他在那边默默的等着别写入记录。当Oracle 数据库从数据文件中读取数据后，数据库就会寻找是否有空闲的缓存块，以便将数据写入其中。

一般来说，数据库在启动的时候，就会在内存中预先分配这些缓存块。所以，Oracle数据库在启动的时候，会占用比较多的内存。但是，这可以免去在实际需要时向内存申请的时间。所以，有时候Oracle数据库虽然已启动，内存的占用率就很高，但是，其后续仍然可以正常运行的原因。而其他数据库虽然刚启动的时候内存占用率不是很高，但是，但系统内存到达80%以上时，在进行数据处理就会受到明显的影响。

### 命中缓存块

当SELECT语句先从数据库文件中读取数据后，会把取得的数据放入到这个命中缓存块中。也就是说，当我们利用查询语句从数据库查询处员工信息后，这个信息就会被保存在高速缓存中。直道高速缓存消耗完毕等原因，这个空间才会被释放。如此的话，下次用户在查询员工信息的时候，就不需要从数据库文件中再次查询相关信息，而直接从数据高速缓存中提取数据，从而提高数据库的访问效率。

### 脏缓存块

当我们利用SELECT查询语句把员工信息的数据查询出来后，数据库会把这个数据所存储的空缓存块做标记，表示该缓存块已经存有数据，使命中缓存块。此时，我们若在利用数据更新语句UPDATE对其中某条记录进行更新时，如要把张三的名字改为张四。运行UPDATE语句后，数据库也首先从高速缓存中查找是否有这条记录，若存在这条记录的话，就直接更改这条记录，并且把该缓存块标记为赃缓存块。如此的话，就可以保持数据的一致性。

## 挖掘算法

### 聚类

    从纷繁复杂的数据中，根据最大化类内相似性、最小化类间相似性的原则进行聚类或分组。 即使得在一个簇内的对象具有高相似性，而不同簇间的对象具有低相似性的过程。

#### 基于划分的聚类方法

* K均值算法
* K中心点算法
* C均值算法

#### 基于层次的聚类方法

* CURE算法
* CHAMELEON算法等

#### 基于密度的聚类方法

* DBSCAN
* OPTICS
* DENCLUE等

#### 基于网格的聚类方法

* STING
* WareCluster
* CLIQUE等

#### 基于模型的聚类方法

* COBWEB
* 神经网络算法等

### 分类

分类的目的是获得一个分类函数或分类模型（也常常称作分类器），该模型能把数据库中的数据项映射到某一个给定类别。分类可用于提取描述重要数据类的模型或预测未来的数据趋势。

*  决策树算法
*  CLS算法
*  ID3算法
*  C45算法
*  Autoclass算法

### 关联

自然界中某种事物发生时其他事物也会发生，则这种联系称之为关联。反映事件之间依赖或关联的知识称为关联型知识（又称依赖关系）。关联的类型分为简单关联、时序关联、因果关联。

* AIS 算法（R. Agrawal等提出）
* Apriori算法（及变种AprioriTid和AprioriHybrid)）
* SETM 算法（M. Houtsma等提出）
* DHP 算法（J. Park等提出）
* PARTITION 算法（A.Savasere等提出）
* Sampling 算法（H.Toivonen提出）
* FP-growth 算法（Jiawei Han提出）

### 预测

#### 传统的预测方法

这些算法的原理简单，理论相对成熟，能用标准的技术来分析模型中的参数.

* 趋势外推法

通常用描散点图的方法定性地确定变化趋势，再按照该变化趋势对未来情况作出预测，特点是不对其中的随机成分作统计处理。

* 时间序列法

将因变量（预测目标）和自变量（影响预测目标的因素）均看成随机变量。实际问题中，多数预测目标的观测值构成的序列表现为（广义）平稳的随机序列或可以转化为平稳的随机序列。虽然在某一给定时刻预测目标的观测值是随机的，但从整个观测序列看，却呈现出某种随机过程（如平稳随机过程）的特性。随机时间序列方法正是依据这一规律性来建立和估计产生实际序列的随机过程的模型，然后用这些模型进行预测。

* 回归分析法等

假定同一个或多个独立变量存在相关关系，寻找相关关系的模型。不同于时间序列法的是：模型的因变量是随机变量，而自变量是可控变量。分为线性回归和非线性回归。目前多用多元线性回归模型。

#### 新的研究方法

* 人工神经网络(ANN)预测方法
* 专家系统预测方法
* 模糊预测方法
* 小波分析预测方法
* 优选组合预测方法等