第 1 章 绪论

1. 数据(data)、信息(information)、知识(knowledge)与价值(value)这四个词在信息科学中既相关联、又具有不同的含义。请举例说明四个概念的关联与区别。

参考答案：数据体现的是一种过程、状态或结果的记录，这类记录数字化（digitalized）后可以被计算机存储和处理。信息则是包含在数据之中的能够为人脑理解和思维推理和结论，比如，" 01001000 01100101 01101100 01101100 01101111 00100000 01110111 01101111 01110010 01101100 01100100 00100001"是一串二进制数值，是一组能被计算机识别、存储和处理的数据。经计算机程序识别转换（ASCII 码值字符转换），我们知道它代表“Hello world！”这样一个字符串，包含了向世界问好的特殊信息。更进一步，在计算机编程语言世界，Hello world！实际上是一个约定俗成的机器或程序语言启动显示语句，这就上升为了知识。最终，如果有人把这一固有的显示方法拿去注册了专利并因此获利，于是就产生了价值。

2. 数据科学家的主要知识技能包括哪几方面？

参考答案：数据科学家的主要知识技能包括如下学科领域（按重要性依次排列）：统计学、数学、计算机科学、机器学习、数据可视化、沟通能力、行业知识

3. 阐述大数据的四大基本特征？

参考答案：4V (Volume, Velocity, Variety and Value)特性

1）大数据的超大规模 (Volume)特点使得它处理的数据量级超过了传统的 GB 规模，达到了 PB 甚至更高量级。超大规模的数据量对数据存储架构、计算模型和应用软件系统都提出了全新的挑战。在后面可以看到，传统的基于行键(row key)表格存储格式的关系型数据库（RDBS）已很难适应大数据海量存储和快速检索查询的需要，基于分布式文件系统的分布式数据库设计越来越多地用于大数据存储与管理系统。

2）Velocity 特征意指大数据的计算处理速度是其可用性、效益性的一个重要衡量指标。

3）Variety 特征指大数据来源、种类的多样性、异构性。大数据的类型按照其结构特征可以分为结构化/半结构化/非结构化数据；按时效性又可分为离线非实时数据/在线实时数据。

4）大数据的 Value 特点是指它的价值低密度、或者说碎片数据毫无价值但大规模整体数据就体现价值的特性。

4. 大数据计算与传统统计学方法的差别？

参考答案：传统统计学是对样本空间基于独立同分布(independent and identically distributed)原理随机抽取一个样本集进行统计分析，而大数据计算是以样本空间整体或完整数据集（也可能不是完整数据集，而只是研究者手中现在掌握的全部数据）作为计算对象。

统计研究者记录下样本的观察数据，根据样本特征推断总体的情况。采样的方法多种多样，有些采样方法会存在偏差，使得样本失真，而不能被视为一个缩小版的总体，去推断总体的特征。当这种情况发生时，基于样本分析所推断出来的结论常常是失真或完全错误的。”这表明传统统计分析方法的正确性和可信性很大程度上依赖于所选取样本集对整个样本空间的代表性，而这不是一个容易的任务。

大数据计算可以处理整个数据集（或研究者手中现在掌握的全部数据），这就避免了只计算一个数据子集（样本集）带来的难题，而可以专注于改进计算模型和算法来提高计算结果的可靠性。

传统统计分析所采用的计算公式或方法是固定的，即统计学家首先建立一个确定的数学模型，再通过选定的样本集测算模型的参数，然后用这个模型去预测总体空间的结果。在这一过程中，所采用的数学模型是确定的、不变的。

大数据计算则主要采用机器学习方法(machine learning)，其特点是预测结果的精度改进是一个动态过程，需要一定规模的数据计算来训练和改进预测算法(prediction algorithm)，这与统计学一开始就确定数学模型不同。具体而言，机器学习是从输入数据中学习(learning)或训练(training)预测算法，通过训练数据集(training set)的大量计算来改进预测算法的性能，使其逐步逼近正确的结果。这一过程中另有一个学习算法(learning algorithm)来控制对预测模型的改进和测试。显然，大数据计算更看重预

测算法的输出结果，并通过训练数据集的反复迭代计算来提高预测输出结果的精度。

5. 大数据计算系统与传统数据库系统的区别？

参考答案：传统的关系型数据库系统(RDBS)主要围绕关系型模型构建，数据存储采用基于主键(primary key)的行存储格式(row-based structure)，一个 SQL 查询会涉及到多个（在大型数据库中会达到数百个）表单，这就限制了关系型数据库处理超大规模数据的能力，因为几十到数百个表单的连接(join)是一个非常耗时的操作。关系型数据库遇到的另一个挑战是处理大量的非结构性数据或异构数据，关系型模型(RDBS schema)在构建这些没有统一数据格式的表格时会遇到很大困难。另外，尽管现代关系型数据库产品也支持分布式部署和计算，但关系型关联模型的特点决定了多数情况下仍然是集中部署，在支持分布式计算时数据集的划分和数据同步都是高成本的开销。

大数据计算采用的是分布式文件系统(distributed file system)及在此基础上构建的 NoSQL (Not Only SQL) 非关系型数据库，通常会在原始数据文件之上建立相关的索引表(index table), 采用哈希表(Hash table) 映射方法来支持快速查询。分布式数据库的特点也能够很好地支持分布式系统部署、对超大规模数据集完成快速查询操作。

而 NoSQL 数据库采用的是基于键值对(key pair) 的列存储格式(columnar storage structure)。针对学生记录属性查询的问题，NoSQL 数据库是把学生记录的属性归类进行存储。比如，所有学生的成绩都存入树状结构的某一分枝（不同课程的成绩进入更低层的分枝）。假设该校共开出 2000 门课，全校共有 100 个专业，每个专业学生人数最多为 1000。NoSQL 数据库首先会搜索进入该门课的分枝（最坏情况下查询次数 2000），然后在该分枝内搜索该专业（最多查询次数 100），然后完成符合条件的学生成绩的读取（最多读取 1000 次），这样，总的操作次数为 2000 + 100 + 1000 = 3100 次。与关系型数据库比较，同样的计算任务，NoSQL 数据库的总查询次数仅为前者的 1/484，这充分体现了基于列存储的非关系型数据库在处理大规模数据上的优势。

第 2 章 大数据计算体系

1. 阐述大数据计算系统涉及到的三个基本系统及其含义。

参考答案：三个基本系统：数据存储系统、数据处理系统、数据应用系统。

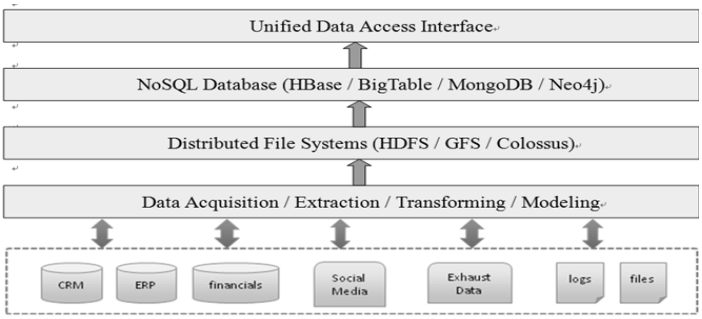
1）数据存储系统包括数据采集层（系统日志、网络爬虫、无线传感器网络、物联网、以及各种数据源）；数据清洗、抽取与建模（将各种类型的结构化、非结构化、异构数据转化为标准存储格式数据，并定义数据属性及值域）

2）数据处理系统包括针对不同类型数据的计算模型（如针对非结构化数据的 MapReduce 批处理模型、针对动态数据流的流计算（Stream Computing）模型、针对结构化数据的大规模并发处理（MPP）模型、基于物理大内存的高性能内存计算（In-memory Computing）模型等）；针对应用需求的各类数据分析算法（回归分析、聚合算法、关联规则算法、决策树算法、贝叶斯分析、机器学习算法等）

3）数据应用系统则是基于上述存系统和计算处理平台提供各行业的大数据应用技术解决方案。

2. 大数据存储架构的构成，并用图展示说明。

参考答案: 目前的大数据存储架构主要由数据层、分布式文件系统、非关系型数据库（NoSQL）、以及一个统一标准的数据读取界面组成，有些设计中还会在 NoSQL 数据库之上加一个提供数据挖掘和分析功能的数据仓库层，如图所示。



3. 美国国家标准学会把数据模型定义为三个层次，分别为哪三个层次，并阐述各层次含义。

参考答案：概念模型主要基于用户的数据功能需求产生，通过与客户的交流获得对客户业务要素、功能和关联关系的理解，从而定义出该业务领域内对应于上述业务要素和功能的实体类（entity class）。概念建模阶段并不拘泥于实体的实现细节或存储方式，重点是表达能够反映客户数据需求和支撑业务流程的数据实体及其相互间的关联关系。

逻辑模型则给出更多的数据实体细节，包括主键、外键、属性、索引、关系、约束、甚至是视图，

以数据表、数据列、值域、面向对象类(object-oriented class) 、XML 标签等形式来描述。在有些建模实践中把概念模型与逻辑模型合为一个模型也是可以的。

物理模型（有时又被称为存储模型）则是考虑数据的存储实现方式，包括数据拆分(partition)、数据表空间、数据集成。有的数据建模工具（如 SparX Enterprise Architect）在此阶段还可按照上述逻辑模型生成与实体对应的 SQL 代码段，用于随后的数据库表格设计。

4. 关系型数据库面临的挑战有哪些？

参考答案：

- 大数据超大规模数据（PB 量级）的存储和管理要求系统具有很好的弹性、在分布式环境中可方便地扩展，传统关系型数据库对数据一致性完整性的强调和集中部署方式使得其扩展性较差、难以适应数据量爆炸式增长的场景。
- RDBMS 基于严格定义的键索引和数据表的存储模式适合结构化数据的存储管理，并能提供高效的基于 SQL 语言的查询机制。但对于非结构化或半结构化数据，RDBMS 就难以处理，查询效率也大大降低。
- 大数据计算处理要求数据存储结构能够很好地支持上层的计算模型。比如 MapReduce 计算模型采用的是分治策略(Divide-and-Conquer)，即先把一个大数据集划分(split)为多个子集，然后每个子集运行 Map 程序进行处理；在完成子集的处理后，再运行 Reduce 程序完成计算结果的汇总，这就要求下层的数据存储结构能够支持这种数据集（或数据表结构）的划分和融汇功能。关系型数据库由于严格的数据一致性完整性要求，难以对数据表进行这种分割处理，因此很难支持大数据计算的各种计算模型。

5. 按照存储架构设计，NoSQL 数据库有哪四种分类？

参考答案：NoSQL 数据库可分为键值数据库(key-value store database)、列存储数据库(column family-oriented database)、文档数据库(document-oriented database)和图形数据库(graph-oriented database)四个大类。

键值（key-value）数据库是基于键值对（key, value）实现对数据的存储和查询。其基本思想是：数据值（value）是通过键（key）来查询，键可以是字符串类型，值可以是任意类型的数据，比如整型、字符型、数组、列表、集合等。底层结构则可采用哈希表（Hash Table）对键进行索引和管理，支持快速查询。但键值数据库不支持基于数据值的查询。

列存储数据库采用列存储结构（columnar storage structure），也有人称之为 DSM（decomposition storage model）存储模型，即数据是基于值域（列）进行检索、存储和管理。列式存储仍然使用键（key），但键是指向列。列存储数据库支持高压缩比，大规模数据下对数据字段的查询非常高效，但不适合于实时删除或更新整条记录，也不支持数据表的 join 操作。

文档数据库是围绕一系列语义上自包含的文档来组织数据管理的，文档没有模式（schema free），也就是说并不要求文档具有某种特定的结构。一个文档数据库实际上是一系列文档的集合，文档其实是一个数据记录，这个记录能够对包含的数据类型和内容进行“自我描述”，XML 文档、HTML 文档和 JSON 文档就属于此类。

图形数据库将社交关系等数据描述为点（Vertex）和边（Edge）及他们的属性（Property），每一张图（Graph）都可以看做是一个结构化数据。

6. 列举两种在大数据计算分析中主要用到的计算模型。

参考答案：数据计算分析主要用到的计算模型有 MapReduce (离线批处理)，流计算(Streaming)。

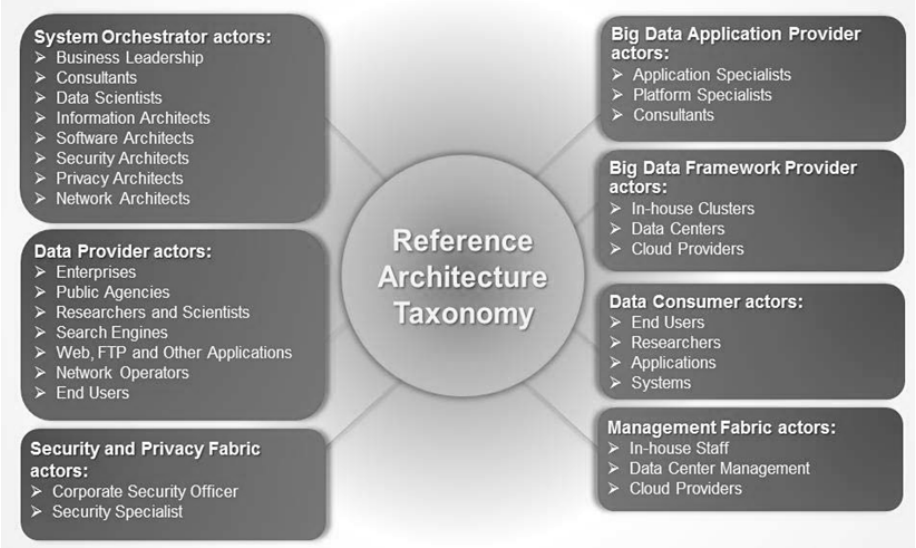
MapReduce 是一种支持分布式计算环境的并行处理模型。MapReduce 程序运行在由多台计算机组成的 Master/Slave 集群架构上（一个 Master 节点，多个 Slave 节点；Master 节点负责任务调度和管理，Slave 节点执行具体的计算任务）。MapReduce 模型实际上采用了分治策略（divide-and-conquer），即将一个大数据集分割为多个小尺度子集（split），然后让计算程序靠近每个子集，同时并行完成计算处理。MapReduce 编程界面简便易用，能够在普通商业计算机集群上有效地处理超大规模数据，是目前大数据计算的一个主流计算模型。MapReduce 的不足之处是硬盘数据读取频繁，处理时效性较差，不适合于要求快速响应的在线智能分析。

流计算（Stream Computing）是一种处理实时动态数据的计算模型。流计算过程：数据采集系统将实时数据（消息队列 MetaQ 或 Socket 导入数据、前端业务数据、Log 监控数据等）通过平台的数据接入层导入 Storm 平台；Storm 实时处理系统则承担数据实时计算分析任务；计算结果则导入数据落地层（Hadoop 的 HDFS 存储系统、MySQL 数据库或 Lustre 文件系统），提供对用户的实时查询服务。另外，Storm 还有一个元数据管理器统一协调前端业务数据写入，定义实时数据类型及描述格式，并指导数据落地层如何处理结果数据。应当注意，Storm 实时处理系统并不是将所有的实时数据都导入落地层，大部分无用的实时数据在完成计算后即丢弃。

第 3 章 大数据标准与模式

1. 描述大数据参考架构主要角色

参考答案：如下图



2. 阐述大数据标准体系中六个类别的标准。

参考答案：数据标准体系由七个类别的标准组成，分别为：基础标准、数据标准,技术标准、平台和工具标准、管理标准、安全和隐私标准、行业应用标准。其具体含义为：

1) 基础标准

为整个标准体系提供包括总则、术语和参考模型等基础性标准。

2) 数据处理标准

数据处理类标准包含数据整理、数据分析和数据访问三种类型的标准。

3) 数据安全标准

数据安全作为数据标准的支撑体系，贯穿于数据整个生命周期的各个段。抛开传统的网络安全和系统安全，大数据时代下的数据安全标准主要包括通用要求、隐私保护两类标准。

4) 数据质量标准

该类标准主要针对数据质量提出具体的管理要求和相应的指标要求，确保数据在产生、存储、交换和使用等各个环节中的质量，为大数据应用打下良好的基础。并对数据全生命周期进行规范化管理。主要包括元数据质量、质量评价和数据溯源三类标准。

5) 产品和平台标准

该类标准主要针对大数据相关技术产品和应用平台进行规范。包括关系型数据库产品、非结构化数据管理产品、商务智能工具、可视化工具、数据处理平台和测试规范六类标准。

6) 应用和服务标准

应用和服务类标准主要是针对大数据所能提供的应用和服务从技术、功能、开发、维护和管理等方面进行规范。主要包括开放数据集、数据服务平台和领域应用数据三类标准。其中开放数据集标准主要对向第三方提供的开放数据包中的内容、格式等进行规范；数据服务平台标准是针对大数据服务平台所提出的功能性、维护性和管理性的标准；领域应用数据指的是各领域根据其领域特性产生的专用数据标准。

3. 根据数据规模、时延性、计算模型、系统结构、关键技术五个维度，对比离线批处理计算、在线交互式计算、及大内存计算的区别。

参考答案：总结如下图

	离线批处理计算	在线交互式计算	大内存计算
数据规模	PB以上	TB～PB	GB～TB
时延性	离线计算(分钟～小时)	在线分析(秒～分钟)	实时计算(秒级)
计算模型	MapReduce Pregel HAMA	Dremel Drill PowerDrill	MemCloud HANA
系统结构	分布式体系	分布式体系	集中式结构
采用技术	大数据迭代循环 硬盘读写次数多	提高数据内存存驻率 data locality columnar data structure	内存一次加载 硬件成本高

第 4 章 数据采集方法

1. 什么是日志采集？日志采集的主要目的是什么？

参考答案：Web 日志包含各种前端 Web 服务器产生的用户访问日志，以及各种 Web 应用程序输出的日志。日志采集是指对这些信息的汇总。日志采集的主要目的是为了进行日志分析。Web 日志中包含了大量人们感兴趣的信息。例如，我们可以从日志记录中获取网站每个页面的页面访问量、访问用户的独立 IP 数；此外，我们还可以获取一些较为复杂的信息。例如，统计出关键词的检索频次排行榜、用户停留时间最长的页面，甚至可获取更复杂信息，包括构建广告点击量模型、用户行为特征分析等。

2. 日志采集的主要过程是什么？传输协议有哪些？

参考答案：日志数据的采集是通过设备中的日志记录子系统实现的，这个子系统能够在必要的时候生成日志消息。当然，具体的日志信息采集方式取决于设备。例如，我们可以对设备进行手工配置，也可以通过硬编码让设备自身生成一系列的预设消息。此外，我们必须使用日志主机来接收日志消息。日志主机是一个基于 Unix 或者 Windows 的服务器系统，它用来集中存储日志消息。日志主机可以集中存储来自多个数据源的日志消息，可以对系统日志信息进行备份，也可以分析日志数据。

3. 请简述网络爬虫的工作原理。

参考答案：网络爬虫往往从一个初始网页的 URL 开始工作，首先获得初始网页上的 URL。在抓取网页的过程中，需要根据网页分析算法过滤与主题无关的链接，保留有用的链接并将其放入等待抓取的 URL 队列中。然后，网络爬虫根据某种搜索策略从队列中选择下一次要抓取的网页 URL，并重复上述过程，直到达到系统的某一停止条件，例如搜索时长或搜索页面数量达到某一阈值。另外，所有被爬虫抓取的网页会自动被系统存储，并建立索引，以便之后的查询和检索。

4. 网络搜索的方法有几种？请简述每种网络搜索的原理，并比较不同搜索算法的优缺点。

参考答案：网页的搜索策略按搜索次序不同，可以分为深度优先、广度优先和最佳优先三种搜索策略。

深度优先的搜索策略表述如下：首先跳转进入起始网页的 URL 链接，分析这个网页中所包含的 URL 链接，选择其中一个 URL 链接进入。如此一个链接一个链接地选择并跳转进入，直到访问完路径中的最后一个 URL。深度优先搜索策略存在如下问题：起始网页通常是网站主页，其提供的链接往往最具价值，浏览和点击量最高。随着每一层 URL 的深入，网页的价值和点击量都会相应地有所下降。这表明重要网页通常距离起始网页的跳转次数较少，而多次跳转抓取到的网页价值往往很低。相对于其他搜索策略而言，深度优先的搜索策略在实际搜索过程中很少被使用。

广度优先的搜索策略和深度优先策略不同。它在抓取 URL 的过程中，只有完成当前层级的搜索

后，才跳转到下一层级进行搜索。广度优先算法的复杂度较高。

最佳优先搜索策略是基于降低广度优先搜索策略的算法复杂度而进行优化的。最佳优先搜索策略按照特定的网页分析算法，预测候选 URL 与主题的相关性，筛选并抓取最相关的某些 URL。

5. RESTful Web 是基于哪些资源进行定义的？

参考答案：RESTful Web 服务（也称为 RESTful Web API）是一个使用 HTTP 并遵循 REST 原则的 Web 服务。它基于以下三方面资源进行定义：

- URI，例如 http://example.com/resources/。
- Web 服务接收与返回的互联网媒体类型，比如：JSON、XML 等。
- Web 服务所支持的一系列资源请求方法（比如：POST，GET，PUT 或 DELETE）。

第 5 章 数据清晰与规约方法

//描述大数据平台采用怎样的方法策略主数据管理，元数据管理。

主数据管理：实现多域主数据管理有两种方法：MDM 应用程序法 和 MDM 平台法。

MDM 应用程序法：MDM 应用程序法附带特定的数据模型、业务逻辑或功能以及图形用户界面，非常适合解决单一、明确界定的业务问题。这类似于购买现成的销售团队的自动化应用程序以管理销售渠道，或购买一个采购应用程序，以管理为供应链购买的直接或间接材料。

MDM 平台法：借助 MDM 平台法，组织可以灵活定义其自身的数据模型，基于定义的模型产生逻辑和功能，并支持基于有关功能配置图形用户界面。

MDM 应用程序法与 MDM 平台法的不同：其他 MDM 方法如应用程序法和 MDM 平台法都可以快速满足组织采纳 MDM 的初始需求，然而应用程序法将不可避免地导致 MDM 孤岛和成本超支。尽管有时确实存在这种情况，即公司有必要寻求能快速实施的 MDM 方案，以在有限范围内解决迫切的业务难题，但当遇到扩展该 MDM 实施以解决其他业务需求或满足将来的不时之需时，平台法无疑是降低总拥有成本和加快实现价值的最佳途径。

MDM 两种方法的对比：应用方法从用户界面开始，然后是业务逻辑，然后是数据模型，以及 Informatica 的平台解决方案，从数据模型开始，然后是业务逻辑，最后才到用户界面。虽然这两种方法都可以使 IT 团队解决眼前的业务问题，但前一种方法会使他们局限于构建 MDM 孤岛来解决每一个后续业务问题。相比之下，后一种方法使他们能够充分利用他们投资的时间、资源和预算，以解决每个后续的业务问题。

其他 MDM 方法如应用程序法的主要不足：

1.每个数据域都有独立的 MDM 应用程序，比如，客户域使用客户数据集成（CDI），产品数据使用产品信息管理（PIM）等；

2.重大业务流程改造采用“大爆炸”方法，往往需要多年时间才能启动；

3.互操作性仅限于同一品牌的应用程序，将客户锁定在可能不适合他们的更广泛的业务需求的产品上；

4.由数据管理员独自管理，使业务用户无法享受自行创建和使用主数据的好处

多领域数据管理：MDM 解决方案能是单一平台上成熟的、灵活多领域的，使整个企业可以迅速部署和轻松扩展，以解决多个部门和地区的业务问题。金融服务、生命科学、制造、医疗保健、政府和各行各业的很多最大型的企业都利用 MDM 来满足其战略要求。

在单一平台上支持所有的 MDM 要求

1. 访问 - 分散的数据源和应用程序带来不一致和重复的主数据
2. 发现 - 抱着发现重复、错误和不一致的宗旨检查数据的一致性和结构
3. 清理 - 解决错误和不完整的字段
4. 掌握 - 将企业相同数据的多个版本合并为一个真实版本或“黄金记录”，并管理内部的层次和关系
5. 交付 - “黄金记录”同步到下游应用程序和数据仓库

主数据管理
通过单一平台上成熟的多领域 MDM 集中主数据的管理，从而消除点对点集成，简化您的结构，降低维护成本，改进数据治理。 MDM（主数据管理）能够通过以下步骤帮助企业成功进行多领域主数据管理：
1. 建模：用灵活的数据模型定义任意类型的主数据
2. 识别：快速匹配和准确识别重复项目
3. 解决：合并以创建可靠、唯一的真实来源
4. 联系：揭示各类主数据之间的关系
5. 治理：创建、使用、管理和监控主数据

MDM 提供业务用户和数据管理员可访问的强大接口，从而实现完整的数据管理和数据异常处理，使您可以轻松浏览不同主数据实体中的多层次结构。

1. 数据预处理的主要任务有哪些？
参考答案：数据预处理的主要任务有：（1）数据清洗：填补缺失数据、消除噪声数据等。数据清洗的原理，就是通过分析“脏数据”的产生原因和存在形式，将“脏数据”转化为满足应用要求的数据，从而提高数据集的数据质量。（2）数据集成：将所用的数据统一存储在数据库、数据仓库或文件中形成一个完整的数据集，这一过程主要用于消除冗余数据。（3）数据转换：主要是对数据进行规格化操作，如将数据值限定在特定的范围之内。（4）数据归约：剔除无法刻画系统关键特征的数据属性，只保留部分能够描述关键特性的数据属性集合。

2. 数据清洗技术按照解决问题的需求可以分为哪几类？请详细阐述每一类问题。
参考答案：包括重复数据处理、消除噪音数据、缺失值处理等三类。

重估数据处理：数据可能存在数据输入错误的问题，如数据格式、拼写上存在的差异（例如，Apple 公司、apple 公司、苹果公司是同一实体的多条记录）。这些差异会导致不能正确地识别出标识同一实体的多条记录，且对于同一实体，在数据仓库中会有多种不同的表示形式，即同一实体对象可能对应多条记录。重复记录会导致错误的分析结果，因此有必要去除数据集中的重复记录，以提高分析的精度和速度。

消除噪音数据：噪声数据是一组测量数据中由随机错误或偏差引起的孤立数据，噪声数据往往使得数据超出了规定的数据域，对后续的数据分析结果造成不良的影响。

缺失值处理：现实世界中，存在大量的不完整数据。造成缺失数据的原因有很多，包括由于人工输入时的疏忽而漏掉，或者在填写调查问卷时，调查人不愿意公布一些信息等。在数据集中，若某记录的属性值被标记为空白、“Unknown”或“未知”时，则认为该记录存在缺失值，是不完整的数据。这些不完整、不准确的数据会影响数据分析结果的准确性，影响信息服务的质量。

3. 清洗数据缺失值的技术有哪些？请比较各种技术的优劣。
参考答案：缺失值清洗的方法，这些方法大致可分为两类：1．忽略不完整的数据值；2．填充缺失数据值的方法。第一类方法操作较为容易，往往通过删除含有不完整数据的属性或实例来去除不完整数据，但这种方法会损失很多数据信息。第二类方法是采用填充算法对不完整的数据进行填充，大多是通过分析其他完整部分的数据对缺失数据进行填充。

4. 数据规约技术有哪些？并详细阐述每种技术的特点。
参考答案：当数据集含有大量的数据属性时，数据的实例数量也非常庞大，这使得此类分析是不可行的。数据归约技术可以降低所需分析数据的数量，且仍接近于保持原数据的完整性。因此，在归约后的数据集上分析会更有效。数据归约的技术较多，主要包括维归约、属性选择和离散化技术。

维归约是通过减少数据集不相关属性的方法，降低数据集的维度，从而提高数据分析算法的效率。维归约方法主要的思路是属性构造，即通过合并已有的属性来构造新的属性，最常用的属性构造方法是根据领域专家的意见来合并已有的属性。

属性选择方法可以减少数据集中的不相关属性。不同于维归约中采用领域知识直接将属性去掉，属性选择通过分析所有可能的属性子集，从而找到最佳的属性子集。

离散化技术可以用于数据转换。比如，对数据集使用分类算法时，需要把数据转换成离散的形式；而对于关联规则发现算法，则需要变为二元变量的属性格式。因此，有时需要从连续型数据转换为离散型数据，而有时需要把连续型和离散型的数据转换为二元变量形式。另外，如果离散数据的值较大，或某些值出现的频率较低，则可以通过合并这些数值来达到对离散数据归约的目的。

5. 常用的数据清洗工具有哪些？请分析每一类工具的应用场景。

参考答案：专用的数据清洗工具往往应用于特定的业务领域、特定的数据清洗阶段或者特定的数据质量问题。这些工具往往依靠某些规则库来指导数据转换过程，或者通过与人的交互来完成数据转换过程。

目前存在较多的和地址相关的数据清洗工具。比如，IDCentric (FirstLogic)，Pureintegrate，QuickAddress (QASSystems)，ReUnion (PitneyBowes)，NADIS，Trillium (TrilliumSoftware) 等都是这类工具。它们提供的技术包括抽取地址信息并将它们转换为符合标准的形式，从而验证城市、邮编、街道等各种信息是否正确。

此外，还有许多工具用于标示或去除重复记录。这些工具包括 DataCleanser (EDD)，Merge／PurgeLibrary (Sagent／QMSoftware)，MatchlT (HelplTSystems)，MasterMerge (PitneyBowes) 等。通常这些工具都要求目标数据源已经过一定的数据清洗，具备了较好的数据质量，不会影响记录匹配过程，因此，这些工具往往需要其他 ETL 工具的配合。大量的商业化工具支持数据的 ETL 过程 (Extraction，Transformation，Loading)，比如 CopyManager，DataStage，Extract，SagentSolutionPlatform，WarehouseAdministrator 等许多工具，这些工具往往利用 DBMS 来统一管理所有的元数据信息，比如数据源信息、目标数据模式、映射关系、脚本程序等。

第 6 章 数据分析算法

1. 简述决策树的原理及过程。

参考答案：决策树是一种类似流程图的树结构，其中每个内部节点（非树叶节点）表示在一个属性上的测试，每个分枝代表一个测试输出，而每个树叶节点存放一个类标号。一旦建立好了决策树，对于一个未给定类标号的实体，决策树会选择一条从根节点到叶节点的路径，该实体的预测结果就存放在该叶节点中。决策树的优势在于不需要任何领域的知识或参数设置，适合于探测性的知识发现。

首先，进行属性选择，需运用属性选择度量又称分裂规则，因为它们决定给定节点上的实体属性如何分裂。属性选择度量提供了每个属性描述给定训练实体数据的秩评定，具有最好度量得分的属性被选作给定元组的分裂属性。通常来说，选择具有最高信息增益的属性来作为节点 N 的分裂属性，该属性使结果划分中的元组分类所需信息量最小。目前比较流行的属性选择度量包括信息增益、增益率和 Gini 指标。

在创建决策树时，由于数据中的噪声点较多，许多分枝反映的是训练数据中的异常点，而剪枝方法是用来去除异常数据的常用方法。通常剪枝方法都使用统计度量，剪去最不可靠的分枝。一般来说，剪枝主要分为两种方法：先剪枝和后剪枝。

2. 阐述 k-均值的算法原理。

参考答案：k-均值法是一种广泛使用的聚类方法。它将 n 个实体分成 k 个簇，保证簇内的相似度尽可能高，且簇间的相似度尽可能低。

k-均值法基于误差平方和准则，随机选择 k 个实体，每个实体代表一个簇的初始均值。对于簇中的每个实体，根据它与各个簇的均值的距离，将该实体指派到最相似的簇中（即与簇中心的距离最小），并计算每个簇的新的均值。此过程不断重复，直至准则函数收敛（即簇分类不变）。误差平方和的定义如下：

$$E=\sum_{i=1}^{k}\sum_{p\in C_i}\left|p-m_i\right|^2 \tag{7}$$

其中，E 是数据集中所有实体的平方误差和；p 是空间中的点，表示给定的一个实体；m_i 表示簇 C_i

的均值。E 所代表的就是所有实体到其所在聚类中心的距离之和。对于不同的聚类方式，E 的大小通常是不一样的。因此，使 E 最小的聚类是误差平方和准则下的最优结果。

k-means 算法选择初始种子点的基本思想就是：初始的聚类中心之间的相互距离要尽可能的远。

- 从输入的数据点集合中随机选择一个点作为第一个聚类中心；
- 对于数据集中的每一个点 x，计算它与最近聚类中心(指已选择的聚类中心)的距离 D(x)；
- 选择一个新的数据点作为新的聚类中心，选择的原则是：D(x)较大的点，被选取作为聚类中心的概率较大；
- 重复 2 和 3 直到 k 个聚类中心被选出来；
- 利用这 k 个初始的聚类中心来运行标准的 k-means 算法。

3. 阐述 k-邻近的算法原理。

参考答案：基于实体的学习方法中最基本的是 *k*-近邻算法。这个算法假定每个实体对应于 *n* 维欧氏空间 Â^n 中的一个点，两个实体的之间的距离是根据标准欧氏距离定义的。更精确地讲，把任意的实体 *x* 表示为下面的特征向量：

$$< a_1(x)，a_2(x)，...，a_n(x)>$$

其中 $a_r(x)$ 表示实体 *x* 的第 *r* 个属性值。那么两个实体 x_i 和 x_j 间的距离定义为 $d(x_i, x_j)$，其中：

$$d(x_i, y_i) \equiv \sqrt{\sum_{r=1}^{n}(a_r(x_i) - a_r(x_j))^2} \qquad (18)$$

基于实体距离的概念，*k* - 近邻算法就是找出某个实体周围最靠近的 k 个实体，从而让初始的某个实体找到属于自己/最靠近自己的一类。

4. 描述 k-均值与 k-邻近算法的区别。

参考答案：

KNN	K-Means
1.KNN 是分类算法 2.监督学习 3.喂给它的数据集是带 label 的数据，已经是完全正确的数据	1.K-Means 是聚类算法 2.非监督学习 3.喂给它的数据集是无 label 的数据，是杂乱无章的，经过聚类后才变得有点顺序，先无序，后有序
没有明显的前期训练过程，属于 memory-based learning	有明显的前期训练过程
K 的含义：来了一个样本 x，要给它分类，即求出它的 y，就从数据集中，在 x 附近找离它最近的 K 个数据点，这 K 个数据点，类别 c 占的个数最多，就把 x 的 label 设为 c	K 的含义：K 是人工固定好的数字，假设数据集合可以分为 K 个簇，由于是依靠人工定好，需要一点先验知识

5. 简述 Adaboost 的计算过程。

参考答案：

可以看到 Adaboost 算法的详细计算过程，总结如下：

1）Adaboost 算法由一系列迭代组成，每次迭代会改变样本的分布。

2）样本分布的改变取决于样本是否被正确分类：

- 赋予分类正确的样本以较低的权值
- 赋予分类错误的样本以较高的权值（通常是分类边界附近的样本）

3）最终的结果是弱分类器的加权组合，其中权值表示该弱分类器的性能。

第 7 章 文本读写技术

//哪种大数据开发语言是最优的

如果说 R 语言是一个神经质又可爱的高手，那么 Python 是它随和又灵活的表兄弟。作为一种结合了 R 语言快速对复杂数据进行挖掘的能力并构建产品的更实用语言，Python 迅速得到了主流的吸引力。Python 是直观的，并且比 R 语言更易于学习，以及它的生态系统近年来急剧增长，使得它更能够用于先前为 R 语言保留的统计分析。

“这是这个行业的进步。在过去的两年时间中，从 R 语言到 Python 已经发生了非常明显的转变，”Butler 说。

在数据处理中，在规模和复杂性之间往往会有一个权衡，于是 Python 成为了一种折中方案。IPython notebook 和 NumPy 可以用作轻便工作的一种暂存器，而 Python 可以作为中等规模数据处理的强大工具。丰富的数据社区，也是 Python 的优势，因为可以提供了大量的工具包和功能。

美国银行使用 Python 在银行的基础架构中构建新的产品和接口，同时也用 Python 处理财务数据。“Python 广泛而灵活，因此人们趋之若鹜，”O’Donnell 说。不过，它并非最高性能的语言，只能偶尔用于大规模的核心基础设施，Driscoll 这样说道。

1. 读取文本常用的函数有哪些?

参考答案：

open()函数: 第一个参数是打开文本文件的路径，第二个参数

r 代表读取模式

w 代表写入模式

a 代表追加模式

r+代表读写模式

read()表示读取到文件尾，size 表示读取大小。

seek(0)表示跳到文件开始位置。

readline()逐行读取文本文件。

readlines()读取所有行到列表中，通过 for 循环可以读出数据。

close()关闭文件。

2. 如何将 csv 文件直接读取到一个 Python 的 DataFrame 对象里面？

参考答案：首先导入 pandas 库

```
>>> import pandas as pd
```

然后通过

```
>>> df=pd.read_csv('test.csv')
```

将 test.csv 存储到 df 这个 DataFrame 里面。

3. 如何将 Python 内容写入文本文件中？

参考答案：要把数据写入 txt 文件，我们就必须先创建 file 对象。但是，在这情况下，必须用 'w' 模式标记指定要写入的文件。首先，我们创建一个名叫 myfile 的文件。

```
>>> mydata = ['Date', 'Time']
>>> myfile = open('testit.txt', 'w')
>>> for line in mydata:
        myfile.write(line + '\n')
```

把 mydata list 的内容写入文件后，关闭文件。

```
>>> myfile.close()
```

4. Python 中如何读取二进制文本？

参考答案：在创建 file 对象时，通过把 'b' 添加到文件模式中，就可以容易地用 Python 处理二进制数据。

```
>>> myfile = open("testit.txt", "wb")
>>> for c in range(50, 70):
```

```
        myfile.write(chr(c))
>>> myfile.close()
```

5. Python 中如何与数据库进行连接？

参考答案：引入数据处理模块之后，我们就需要和数据库进行连接了，具体的实现代码如下：

```
db = MySQLdb.connect("localhost","root","123456","myciti" )
```

上面的代码中含有四个关键的参数：第一个参数是服务器的地址；第二个参数是用户名；第三个参数是 dbms 密码；第四个参数是需要访问的数据库名称。

第 8 章 数据处理技术

1. 当两个数据集的索引全部或部分重叠时，它们的数据组合问题就不能用简单的合并（merge）或连接（concatenation）运算来处理。用 python 代码举例说明如何解决以上问题。

参考答案：

下面实现完全重叠的两个数据集的合并，当第一个数据集非空时，取第一个数据集的值，否则取第二个数据集的值：

```
In [1]: a
Out[2]:
f    NaN
e    2.5
d    NaN
c    3.5
b    4.5
a    NaN
dtype: float64
```

```
In [3]: b
Out[4]:
f    0
e    1
d    2
c    3
b    4
a    NaN
dtype: float64
```

```
In [5]: np.where(pd.isnull(a), b, a)
Out[6]: array([ 0.,  2.5,  2. ,  3.5,  4.5,  nan])
```

2. 下面(左)的数据中有多行存在重复的数据。请只针对 k1 和 k2 列，进行去重。

	k1	k2	v1
0	one	1	0
1	one	1	1
2	one	2	2
3	two	3	3
4	two	3	4
5	two	4	5
6	two	4	6

参考答案：

```
In [1]: data.drop_duplicates(['k1', 'k2'], take_last=True)
Out[2]:
    k1   k2   v1
1   one   1    1
2   one   2    2
4   two   3    4
6   two   4    6
```

3. 连续数据常常被离散化。假设有一组人员数据，希望将它们划分为不同的年龄组：

ages = [20, 22, 25, 27, 21, 23, 37, 31, 61, 45, 41, 32]

如果我们想要将这些数据划分为“18 到 25”、“26 到 35”、“35 到 60”以及“60 以上”。用 python 代码实现以上分档需求，并设置每档的名称，将 labels 选项设置为['Youth', 'YoungAdult', 'MiddleAged', 'Senior']。

参考答案：

```
In [1]: bins = [18, 25, 35, 60, 100]

In [2]: cats = pd.cut(ages, bins)

In [3]: cats

Out[4]:
(18, 25]
(18, 25]
(18, 25]
(25, 35]
(18, 25]
(18, 25]
(35, 60]
```

```
   (25, 35]                                    Youth
  (60, 100]                                    Youth
   (35, 60]                                YoungAdult
   (35, 60]                                    Youth
   (25, 35]                                    Youth
                                           MiddleAged
In [5]: group_names = ['Youth',            YoungAdult
'YoungAdult', 'MiddleAged', 'Senior']          Senior

                                           MiddleAged
In [6]: pd.cut(ages, bins,                 MiddleAged
labels=group_names)                        YoungAdult
Out[7]:                                 Levels (4): Index(['Youth', 'YoungAdult',
      Youth                             'MiddleAged', 'Senior'], dtype=object)
```

4. 简述正则表达式的含义。

参考答案：正则表达式（regex）提供了一种灵活的在文本中搜索或匹配字符串模式的方式，它根据正则表达式语言编写字符串。Python 内置的 re 模块负责对字符串应用正则表达式。

第 9 章 数据分析技术

1. 用 matplotlib 工具包创建直方图。

参考答案：

```
import numpy
import pylab
# Build a vector of 10000 normal deviates with variance 0.5^2 and mean 2
mu, sigma = 2, 0.5
v = numpy.random.normal(mu,sigma,10000)
# Plot a normalized histogram with 50 bins
pylab.hist(v, bins=50, normed=1)          # matplotlib version (plot)
pylab.show()
# Compute the histogram with numpy and then plot it
(n, bins) = numpy.histogram(v, bins=50, normed=True)   # NumPy version (no plot)
pylab.plot(.5*(bins[1:]+bins[:-1]), n)
pylab.show()
```

2. 阐述 Dataframe 的定义。

参考答案：DataFrame 是一种表格类型的数据结构，它含有一组有序的列。每一列可以是不同类型的值（例如数值、字符串、布尔值等）。DataFrame 既可以按行索引，也可以按列索引，因而可以被视为由 Series 组成的字典。与其他数据结构相比，DataFrame 中对行操作和对列操作基本上是平衡的。其实，DataFrame 中的数据是通过一个或多个二维块进行存放的。

3. 用 Scikit-Learn 工具包实现逻辑回归。

参考答案：

```
from sklearn import metrics
from sklearn.linear_model import LogisticRegression
model = LogisticRegression()
model.fit(X, y)
print(model)
# make predictions
expected = y
```

```
predicted = model.predict(X)
# summarize the fit of the model
print(metrics.classification_report(expected, predicted))
print(metrics.confusion_matrix(expected, predicted))
```

4. 用 Scikit-Learn 工具包实现 CART 决策树算法。

参考答案：

```
from sklearn import metrics
from sklearn.tree import DecisionTreeClassifier
# fit a CART model to the data
model = DecisionTreeClassifier()
model.fit(X, y)
print(model)
# make predictions
expected = y
predicted = model.predict(X)
# summarize the fit of the model
print(metrics.classification_report(expected, predicted))
print(metrics.confusion_matrix(expected, predicted))
```

5. 用 Scikit-Learn 工具包实现朴素贝叶斯算法。

参考答案：

```
from sklearn import metrics
from sklearn.naive_bayes import GaussianNB
model = GaussianNB()
model.fit(X, y)
print(model)
# make predictions
expected = y
predicted = model.predict(X)
# summarize the fit of the model
print(metrics.classification_report(expected, predicted))
print(metrics.confusion_matrix(expected, predicted))
```

第 10 章 数据可视化技术

1. 简述 Matplotlib 支持哪些功能？

参考答案：Matplotlib 来自于由 John Hunter 在 2002 年启动的一个用于创建图表的绘图项目，其目的是为 Python 构建一个与 Matlab 之间进行交互的绘图接口。Matplotlib 可以支持各类操作系统上的 GUI 后端，也可以将图片存储为各类格式的图片：包括 PDF、JPG、PNG、GIF 等。此外，Matplotlib 还支持许多插件工具，包括用于 3D 绘图的 mplot3d，用于地图描绘的 basemap 等。

2. 简述 Mayavi2 有哪些特征？

参考答案：Mayavi2 可以使用 Python 语言编写，因此 Mayavi2 既是一个便捷的可视化软件，又是一个可以通过 Python 编写扩展的工具。它可以自动嵌入到用户自主编写的 Python 程序中，又可以直接使用脚本的 API 来实现快速绘图。Mayavi2 的 mlab 模块提供了便捷的绘图函数。数据准备好之后，通过调用一次 mlab 的函数就可以呈现数据的显示效果图，比较适合在 IPython 的中交互式界面中使用。

3. 除了 matplotlib 和 Mayavi 以外，Python 中还可以采用哪些类库实现图表图形的绘制？

参考答案：

(1) Cairoplot：http://linil.wordpress.com/2008/09/16/cairoplot-11/。Cairoplot 在网页上的展示效果强于 flex 中的图表实现能力，但此工具目前更多地在 linux 平台上使用，与 windows 平台兼容性较差。

(2) Chaco：http://code.enthought.com/chaco/。Chaco 是一个二位的绘图工具，其中文教程可参考：http://hyry.dip.jp/pydoc/chaco_intro.html。

(3) Python Google Chart：http://pygooglechart.slowchop.com/。Python Google Chart 是对 Google chart API 的一个完整封装。

(4) PyCha：https://bitbucket.org/lgs/pycha/wiki/Home。PyCha 是 Cairo 类库的一种封装形式，它可以为实现轻量级的应用，还可以实现一系列的参数优化。

(5) PyOFC2：http://btbytes.github.com/pyofc2/。PyOFC2 是 Open Falsh Library 的 Python 类库，其实现的图形具有 Flash 效果，可以通过鼠标拖动来动态地显示图标信息。

(6) Pychart：http://home.gna.org/pychart/。PyChart 可以用来创建高品质封装的图表库，包括 PDF、PNG、SVG 等格式的图表库。

(7) PLPlot：http://plplot.sourceforge.net/。PLPlot 是一种可以用来创建科学图表的开源跨平台软件开发包。此开发包以 C 类库作为核心，支持 C 类库中的各种编程语言，包括 C, C++, Fortran, Java, Python, Perl 等。

(8) Reportlab：http://www.reportlab.com/software/opensource/。Reportlab 支持在 pdf 中画图表，它的实现可以参考 http://www.codecho.com/installation-and-example-of-reportlab-in-python/。

(9) Vpython：http://www.vpython.org/index.html。VPython 是 Visual Python 的简写，由卡耐基梅隆大学的在校学生 David Scherer 于 2000 年撰写的一个 Python 三维绘图模块。

4. 如何使用 Python 在一张图中绘制 2X2 的四幅图？

参考答案：可以用 plt.figure 这个函数来创建一个新的 Figure 对象：

```
fig = plt.figure()
```

采用 add_subplot 创建 subplot：

```
ax1 = fig.add_subplot(2, 2, 1)
ax2 = fig.add_subplot(2, 2,2)
ax3 = fig.add_subplot(2, 2, 3)
ax4 = fig.add_subplot(2, 2, 4)
```

5. 如何在 python 中添加图例？

参考答案：图例是一种用来识别图表中元素的方式，添加图例的方法有多种，最常用的就是在添加 subplot 的时候导入 label 参数。

第 11 章 Hadoop 生态系统

1. Hadoop 集群中可以用几种模式进行运行?每种模式有哪些特点?

参考答案：Hadoop 集群分为单机（本地）模式、伪分布式模式、全分布式模式三种。在单机模式（standalone）中不会存在守护进程，所有东西都运行在一个 JVM 上。这里同样没有 DFS，使用的是本地文件系统。单机模式适用于开发过程中运行 MapReduce 程序，这也是最少使用的一个模式。伪分布式（Pseudo）适用于开发和测试环境，在这个模式中，所有守护进程都在同一台机器上运行。全分布模式通常被用于生产环境，这里我们使用 N 台主机组成一个 Hadoop 集群，Hadoop 守护进程运行在每台主机之上。这里会存在 Namenode 运行的主机，Datanode 运行的主机，以及 task tracker 运行的主机。在分布式环境下，主节点和从节点会分开。

2. Hadoop 的核心配置是什么？拥有哪些配置文件？

参考答案：Hadoop 的核心配置通过两个 xml 文件来完成：1，hadoop-default.xml；2，hadoop-site.xml。这些文件都使用 xml 格式，因此每个 xml 中都有一些属性，包括名称和值，但是当下这些文件都已不复存在。Hadoop 现在拥有 3 个配置文件：1，core-site.xml；2，hdfs-site.xml；3，mapred-site.xml。这些文件都保存在 conf/子目录下。

3. 集群中的 Master 和 Slave 节点是如何组成？
参考答案：Masters 同样是主机的列表组成，每台一行，用于说明第二 Namenode 服务器。Slaves 由主机的列表组成，每台 1 行，用于说明数据节点。

4. 为什么 SSH 本地主机需要密码？如果在 SSH 中添加 key，是否还需要设置密码？
参考答案：在 SSH 中使用密码主要是增加安全性，在某些情况下也根本不会设置密码通信。即使在 SSH 中添加了 key，还是需要设置密码。

5. 如何重启 Namenode？
参考答案：点击 stop-all.sh，再点击 start-all.sh。 键入 sudo hdfs（Enter），su-hdfs （Enter），/etc/init.d/ha（Enter），及/etc/init.d/hadoop-0.20-namenode start（Enter）。

第 12 章 MapReduce 计算模型

1. 简述 Map 包含哪些步骤？
参考答案：（1）读取输入文件内容，解析成 key、value 对。对输入文件的每一行，解析成 key、value 对。每一个键值对调用一次 map 函数。（2） 写自己的逻辑，对输入的 key、value 处理，转换成新的 key、value 输出。（3）对输出的 key、value 进行分区。（4）对不同分区的数据，按照 key 进行排序、分组。相同 key 的 value 放到一个集合中。（5）分组后的数据进行归约。

2. 简述 Reduce 包含哪些步骤？
参考答案：（1）对多个 map 任务的输出，按照不同的分区，通过网络 copy 到不同的 reduce 节点。（2）对多个 map 任务的输出进行合并、排序。写 reduce 函数自己的逻辑，对输入的 key、value 处理，转换成新的 key、value 输出。（3）把 reduce 的输出保存到文件中。

3. MapReduce 中排序发生在哪几个阶段？这些排序是否可以避免，为什么？
参考答案：一个 MapReduce 作业由 Map 阶段和 Reduce 阶段两部分组成，这两阶段会对数据排序，从这个意义上说，MapReduce 框架本质就 是一个 Distributed Sort。在 Map 阶段，Map Task 会在本地磁盘输出一个按照 key 排序（采用的是快速排序）的文件（中间可能产生多个文件，但最终会合并成一个）；在 Reduce 阶段，每个 Reduce Task 会对收到的数据排序，数据便按照 Key 分成了若干组，之后以组为单位交给 reduce 处理。很多人的误解在 Map 阶段，如果不使用 Combiner 便不会排序，这是错误的，不管你用不用 Combiner，Map Task 均会对产生的数据排序（如果没有 Reduce Task，则不会排序， 实际上 Map 阶段的排序就是为了减轻 Reduce 端排序负载）。由于这些排序是 MapReduce 自动完成的，用户无法控制。因此，在 hadoop 1.x 中无法避免，也不可以关闭，但 hadoop2.x 是可以关闭的。

4. 编写 MapReduce 作业时，如何做到在 Reduce 阶段，先对 Key 排序，再对 Value 排序？
参考答案：该问题通常称为”二次排序“，最常用的方法是将 Value 放到 Key 中，实现一个组合 Key，然后自定义 Key 排序规则（为 Key 实现一个 WritableComparable）

5.如何使用 MapReduce 实现两个表 join，可以考虑一下几种情况：（1）一个表大，一个表小（可放到内存中）；（2）两个表都是大表。
参考答案：第一种情况比较简单，只需将小表放到 DistributedCache 中即可；第二种情况常用的方法有：map-side join（要求输入数据有序，通常用户 Hbase 中的数据表连接），reduce-side join，semi join（半连接）。

第 13 章 图并行计算框架

1. 为什么在 MapReduce 计算模型之外还需要图并行计算模型？图并行计算框架与 MapReduce 批处理模型的主要差别在哪里？

参考答案：现实世界中有很多应用数据更适合以网络或图的形式展现。比如 Facebook，Twitter，新浪微博，人人等大型社交网络中存在的 Web 社团，这种社交网络数据最适合用网络图来表示。这类以图（graph）形式表征的数据在大数据系统需要处理的数据量中占了相当一个比例（Google 公司提到其搜索引擎处理的数据量中有 20%是由图处理引擎完成），因此需要一个针对这类网络图计算问题的计算模型。

许多图计算问题算法都带有全局循环迭代（iteration）步骤（如求解单源最短路径问题的 Dijkstra 算法、最大流/最小割问题（Max-Flow Min-Cut）、数据聚类 K-means 算法等）,而 MapReduce 计算模型是一种典型的批处理（batch processing）模式，即数据计算是按照流水线方式执行，完成第一步，才会执行第二步；每一步内可能有大量的并行处理线程，但没有跨越很多步的迭代循环计算。因此图计算问题对于批处理计算模型是一个难题。

与 MapReduce 批处理模型比较，图计算模型具有如下特点：1）图计算问题具有全局性，计算过程中有大量节点间通信；2）图算法需要完成全局循环迭代步骤；3）图分割问题（即将一个大图的数据集划分为小图的子数据集）很复杂。

2. 图并行计算系统目前有三种技术方案：基于 BSP 模型的 Pregel 和 Hama，基于节点计算（vertex-program）的 GraphLab，及图数据库 Neo4j 和 InfiniteGraph，试论述三种技术方案的差异。

参考答案：基于 BSP 模型的图并行计算模式（Pregel, Hama）采用 Master/Slave 计算架构，与 Hadoop 平台架构能较好地匹配。它采用全局同步（Superstep 和 Synchronization Barrier）方式来实现计算同步与节点间通信，具有简便易行、编程界面友好的特点。

GraphLab 采用了基于节点计算（vertex-program）的 GAS（gather-apply-scatter）模型来处理图并行计算，其并行机制与 Pregel/HAMA 不一样，它把计算任务分解到各个独立计算节点（Map），但各计算节点与其相邻节点之间存在 data-dependency 和 computation-dependency，即某个节点计算的下一步还需取决于相邻节点的计算状态。由于 GraphLab 没有一个全局时钟来实现同步控制，因此每个节点的程序需要提供同步功能。

图数据库（Neo4j, InfiniteGraph）则是将社交关系图（graph）描述为点（vertex）、边（edge）及它的属性（property），这里“点”代表实体，比如人、企业、账户或其他任何数据项，类似于关系数据库中的数据记录或文档数据库中的文件；“边”则代表点与点之间的关系；“属性”则是“边”所包含的用户关注的特性。图数据库更多是以图的数据存储方式的角度去提供图数据查询解决方案。

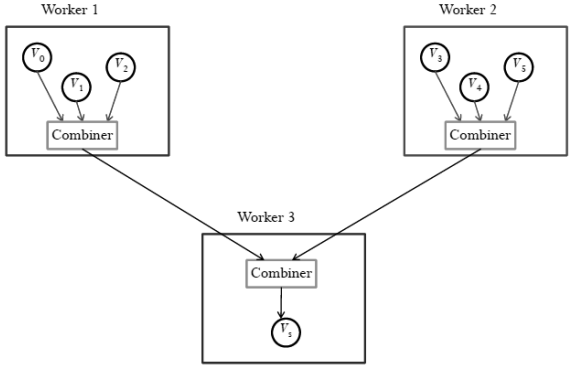
三者的主要差别在于前两者侧重于支持各种图算法（计算模型有差异），而第三者侧重于支持图数据存储与查询。

3. 为什么 Pregel 的节点间通信必须被局限在超步之间的障碍期（barrier）进行？不这样做有啥后果？

参考答案：Pregel 设计了各顶点的计算都在节点本地进行，各顶点计算是独立的，没有对其他顶点计算结果或计算逻辑上的依赖性，这就决定了没有顶点之外的资源竞争，因此避免了分布式异步计算系统中容易发生的 deadlock。顶点间的通信被局限在超步之间的 barrier 期间完成（每个顶点可以在超步内送出给其它顶点的消息，但这些消息不会马上处理。当这个超步结束时下一个超步开始前（即 barrier 期间），所有的顶点统一处理它们各自收到的消息）可使得：1）实现一种统一的顶点间全局通信机制（barrier 期间全部顶点都进行通信）；2）各顶点上的迭代循环计算不受到其他顶点的牵扯。

如果不采用这种统一的障碍期（barrier）全局通信机制，而容许顶点之间可以随时发送消息，接收到消息的顶点随时处理，势必造成顶点相互间的计算依赖关系，大大影响计算速度或导致顶点处的迭代循环无法进行。

4. 在 BSP 模型中，消息发送和接收的 Combiner 机制可在发送节点实现，也可以在接收节点实现。什么时候我们选择在发送节点实现 Combiner？什么时候选择在接收节点实现 Combiner？各自的目的是什么？



参考答案：在 BSP 模型中，多个顶点（vertex）可能同时向另一个顶点发送消息，如图所示，节点 Worker 1 上的顶点 V_0，V_1, V_2 都向 Worker 3 上的顶点 V_6 发送消息。但在某些算法设计中，接受顶点关注的并不是每一个发送顶

点的单独值，而可能是其中的最大值或是求和值。这种情况下，Pregel 提供了 Combiner 机制来合并发出消息，使得多个顶点发给同一目标点的多个消息可以先合并成一条消息，然后再发出。

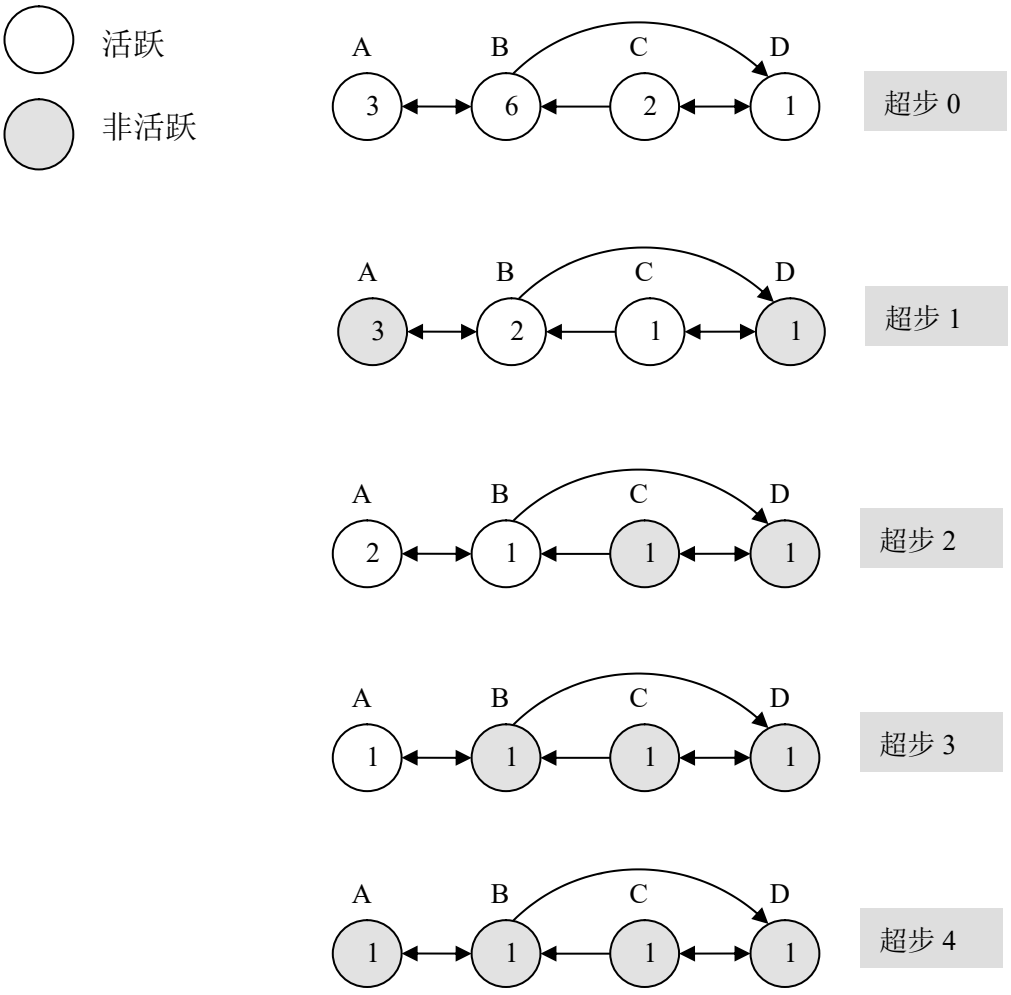
如果目的是减少消息传递开销、降低网络流量负担，这种 Combiner 功能可以在发送端实现，将多条消息合并为一条再发出；如果为了在接收端加快处理进度，可以将 Combiner 在接受端实现（图中的 Worker 3 的 Combiner）。需要注意，接收端 Combiner 并没有降低网络流量。

5. 节点通信中 Combiner 的使用是为了降低节点间网络通信开销，更有效地使用网络资源。但是不是所有节点计算都适用 Combiner？使用 Combiner 时需遵循的一条准则是什么？

参考答案：使用 Combiner 时需遵循的准则是：不管是在发送端还是接收端，对消息的 combine 处理（实际是一种预处理）都不能影响最终的计算结果的正确性。也即是，不管是否进行 combine 处理，其最终计算结果都是一样的（使用 Combiner 只是降低了网络流量或是加快了处理速度）。这就要求 combine 操作应该符合交换律和结合律。

6. 参照图 12-18 的最大值算例，若将问题改为需要将最小值传播到每个顶点，列出传播过程的各个超步步骤。

参考答案：如果图 12-18 的算例改为最小值问题，则问题变为：给定一个有向连通图，图中每个顶点都包含一个值，它需要将最小值传播到每个顶点。在每个步骤中，顶点会从接收到的消息中选出一个最小值，并将这个值传送给其所有的相邻顶点。当某个步骤已经没有顶点更新其包含值，那么计算就告结束。



按照 Pregel 的设计，所有的顶点值的更新都在超步内；每个顶点只在超步结束时向其所有邻接点发送消息（传送顶点值）。当一个顶点收到的消息中含有值比它目前值小，则用最小的一个值替换它目前值，状态设置为“活跃”，否则就将状态改为“非活跃”；当所有顶点状态为“非活跃”时，计算结束。

计算过程如下（参照上图）：

超步 0：A，B， C， D 四个顶点状态均设为“活跃”，各自包含一个初始值；
超步 1：A 向 B 传送值 3；B 接受 A 的消息值 3，用 3 替代目前值 6，B 保持为“活跃”；B 向 A 和 D 传送值 6，B 接受 C 的消息值 2，B 用 2 代替 3，B 状态为“活跃”；A 的值不变（因为 3<6），状态改变为“非活跃”；C 向 B 和 D 传送值 2，C 接受 D 的消息值 1，C 值改变为 1，状态为“活跃”；D 向 C 传送值 1，D 接受 A 的消息值 6，D 的值保持为 1，D 状态改变为“非活跃”；
超步 1 结束时，A，B， C， D 四个顶点的即时值分别为 3, 2，1，1，状态分别为“非活跃”，“活跃”，“活跃”，“非活跃”；
超步 2：我们只对状态是“活跃”的顶点执行操作，B 向 A 传送值 2，A 的值替换为 2，因此 A 状态变为“活跃”；B 向 D 传送值 2，D 收到后保持原来值 1 不变，D 的状态仍然为“非活跃”；C 向 B 发送值 1，B 收到后更换为 1，B 状态仍然为“活跃”；C 向 D 发送值 1，D 收到保持原来值 1 不变，C 状态变为“非活跃”。
超步 2 结束时，A，B， C， D 四个顶点的即时值分别为 2, 1，1，1，状态分别为“活跃”，“活跃”，“非活跃”，“非活跃”；
超步 3：此时有顶点 A 和 B 状态是“活跃”，只需对顶点 A 和 B 执行操作。A 向 B 传送值 2，B 收到不需更新，因此 B 状态又变为“非活跃”；B 向 A 发送消息值 1，A 替换为 1，A 状态更新为“活跃”；B 向 D 发送消息值 1，D 不做更新，D 状态保存为“非活跃”。
超步 3 结束时，A，B， C， D 四个顶点的即时值分别为 1, 1，1，1，状态分别为“活跃”，“非活跃”，“非活跃”，“非活跃”；
超步 4：此时只有顶点 A 状态是“活跃”，只需对顶点 A 执行操作。A 向 B 传送值 1，B 收到不作更新，因此 A 状态又变为“非活跃”。
超步 4 结束时，A，B， C， D 四个顶点状态全部为“非活跃”，且下一步无消息产生（只有“活跃”状态的顶点才能发消息），至此计算全部结束。

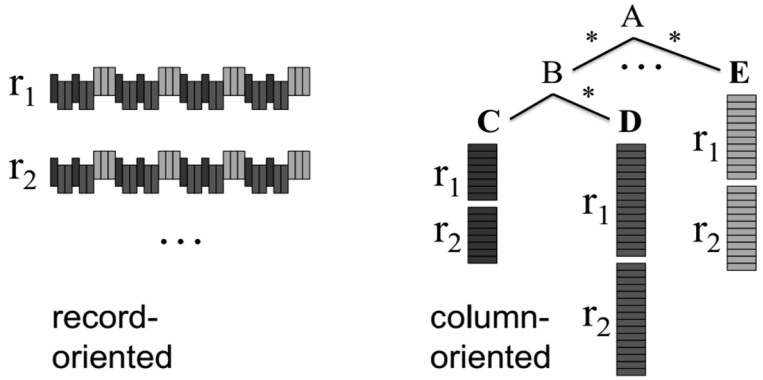
第 14 章 交互式计算模式

1. 为什么说交互式计算模式（interactive analysis）是界于 MapReduce 批处理计算和大内存计算之间的一个折衷解决方案？它主要依靠什么技术实现？
参考答案：以 MapReduce 为代表的批处理计算被证明是一种数据量大、性价比高、技术成熟的解决方案，但其计算延迟长（通常在数小时到数天的量级），不利于对时效要求高的在线实时分析类应用；以 RAMCloud 和 SAP HANA 为代表的内存计算模式处理速度快（通常在毫秒到秒量级），有利于实时智能分析，但系统硬件成本高，常需要专用硬件设备，与周边系统同步兼容性差。

以 Google 的 Dremel 为代表的交互式计算模式并非基于昂贵硬件平台的技术方案，而是运行在通用商业机器集群上，主要通过嵌套数据结构（nested data structure）、列存储（columnar storage）和查询树（query tree）等软件技术来实现 Web 规模数据查询性能的飞跃提升。因此，交互式计算模式可看作一种界于 MapReduce 批处理和内存计算之间的一个折衷方案，运行在廉价商业硬件平台上，通过特定的软件技术来实现超大规模数据的快速查询。

2. 什么是列存储结构（column-based storage structure）？为什么列存储结构的查询效率要远高于基于行存储结构（row-based storage structure）的关系型数据库？
参考答案：大多数需要存储的数据结构最后都可以表示为一张二维数据表，即数据表由多行（row）组成，而每一行包含多个值域（field）或字码段。多行数据的同一值域或字码段则构成一列（column）。

这个二维数据表存储时有两种方式：行存储（row-oriented storage）和列存储（column-oriented storage），如下图所示。行存储是以数据表的行键（RowKey）为基准、以数据记录（record）为单位进行存储，每一行数据包含了一个对象或事务的完整记录，每一行记录包含了多个值域（图左边 r1 和 r2 的不同颜色块表示不同的值域）。列存储则是将不同记录的相同值域（r1 和 r2 的相同颜色块）放入一个列中存



储，采用的是树状存储结构，如右边所示。

如果是行存储，在读取数据时（查找一条记录的某个值域）需要完成两个步骤：1）纵向按行键（RowKey）查找到该行；2）横向向右搜索，跳过不相关值域，直到找到查询项。这种存储方式使得每读一个 RowKey 后，都需要跳到下一个 RowKey 的位置，所有要搜索的字段都不是连续存放，且有些值域是变长度的字符串（repeated）,不能通过简单公式计算得到地址，查询起来效率非常低。

而如果按列存储方式，只需按树状结构找到需要查询的列所在分支的首地址，然后顺序读取数据（每个 record 对应值域的地址飘移值（offset）都记录在元数据表中），而不需要扫描其他不相干的分支。列存储格式不仅实现简单，而且磁盘顺序读取比随机读取要快得多，而且更容易进行优化（比如把临近地址的数据预读到内存，对连续同类型数据进行压缩存放），因而查询效率大大提高。

3. Dremel 将嵌套数据结构在实际存储时映射成一维存储结构，在计算过程中常常需要将内存中的一维存储结构恢复成原有的数据结构。Dremel 是通过什么方法实现数据结构的无损表达（lossless representation）和高速组装（quick re-assembly）的？试简述之。

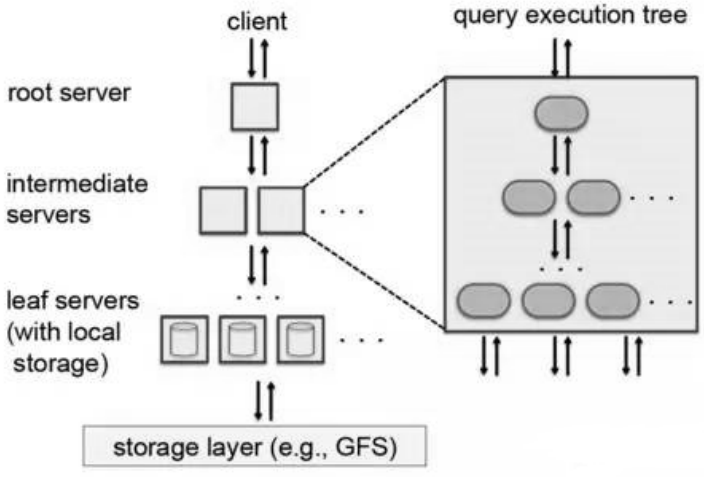
参考答案：Dremel 采用了基于值域的列存储模式，即将数据记录基于值域拆分成多个列存储表，每个列存储表（仅包含一列或一个字码段）存储了多个记录的相同值域（字码段）的值，在物理存储时将多个列存储表进行一维顺序存储。

Dremel 的列存储表中不光包含各记录的列值或字码段，还包含了其对应的 r 值（repetition level）和 d 值（definition level），r 值和 d 值的定义参加书中 14.2 节。有了包含 r 值和 d 值的列存储表，Dremel 可以对每个值域或字码段按照有限状态机（FSM）规则（图 14-8）读取顺序存储的物理表并按 14.2 节所描述的数据重构方法进行原数据记录的重构。

每次对一维顺序存储的物理表进行扫描和数据记录重建时，Dremel 并不需要扫描和重建全部数据，而可根据需要只扫描部分数据，重新组装感兴趣的值域（列）。

4. 根据第 14 章图 14-11 的 Dremel 查询树结构，说明为什么中间节点层（intermediate sever）的层次不宜太多（比如多于两层）？

参考答案：Dremel 采用的是多层服务树（serving-tree）查询架构，如右图所示。最上层的根服务器（root server）接收所有的客户端查询请求，并把查询语句分解，读取相关元数据，再把分解后的请求下发中间服务器（intermediate server）。中间服务器进一步把查询需求分发到它所属的下级叶节点服务器（leaf server）完成并行计算。数据记录存储在叶节点服务器的本地文件系统上，叶节点完成计算处理后，其返回计算结果的过程与上述步骤逆向而行。

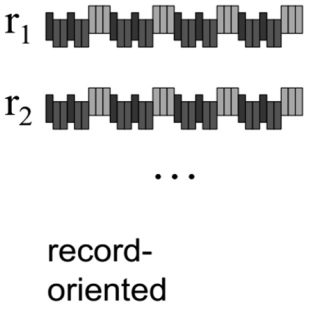


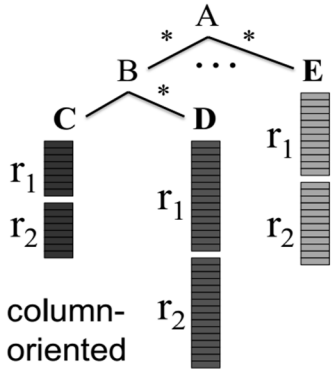
Dremel 查询服务树的层级数可以人为设定。比如，一个有 3000 个叶子服务器节点的系统，可以是两层（1：3000），即一个根服务器和 3000 个叶子服务器；也可以设置成三层（1：100：3000），即在根服务器和底层叶子服务器之间增加 100 个中间服务器；如果选择两层，根服务器很容易成为计算瓶颈（所有的叶子服务器的请求汇集到根服务器）。而当服务树有三层时（增加一层中间节点），整个系统的并行度得到提高，执行性能也大大优化（约 6 倍左右）；再多的中间层，比如三层以上，对于系统的性能提高并无太多帮助，反而使得系统软件结构过于复杂。

5. 从列存储结构和查询树并行模型的特点说明为什么交互式计算模式只适宜于数据查询业务，而不适宜于数据增删操作。

参考答案：列存储结构则是将不同记录的相同值域（如下图 r1 和 r2 记录的相同颜色块）放入一个列中存储，采用的是树状存储结构，如下图右边所示。这种树状列存储结构适合于快速查找记录的某个值域或字码段，因为不同记录的相同值域或字码段被存储在树状结构的某一叶节点分支内，只要快速找到这个叶节点，也就确定了所查询的值域或字码段的位置。

但这一存储结构并不利于数据库的增删操作。如果需要增加一条新记录，不同于行存储的只需要在二维表中增加一行（或是一维存储结构中增加一段空间），列存储需要一次找到该记录包含的各项值域（字码段）在树状存储结构中的各项分支并且嵌入到相关位置。这种遍历存储树结构的操作是一个非常耗时且高成本的操作。从列存储树结构中删掉一条完整记录的过程与上面相似，执行成本非常高。因此基于列存储结构的交互式计算模式只适宜于数据查询，而不适合于数据增删。





第 15 章 流计算系统

1. 流数据处理（stream data processing）有哪两种基本模式?从系统吞吐率和时间延迟性看，这两种模式各有什么特点?

参考答案：目前有两种主流的流计算模式：Native Stream Processing System（逐条处理模式）（图 15-3）和 Micro-batch Stream Processing System（打包处理模式）（图 15-4）。Native Stream Processing System 是基于数据按其读入顺序逐条进行处理，每一条数据到达即可得到即时处理（假设系统没有过载），延迟时间（delay time）较短，系统响应性好。Micro-batch Stream Processing System 是将数据流先作预处理，打包成含多条数据的 batch（批次）再交给系统处理。打包处理模式的特点是系统吞吐率（throughput）较高，容错处理更好。

2. Spark 的 micro-batch 模型 RDD (resilient distributed dataset)以 2 秒为单位截取数据流构成一个个数据包。如果以小于 2 秒或大于 2 秒为单位截取数据流构成 RDD，各有什么利弊?

参考答案：流数据处理计算以时间窗口截取生成 RDD 数据包，时间窗口的大小（如每 2 秒）将决定 RDD 数据包的大小。如果以小于 2 秒的时间窗口截取数据包，所形成的 RDD 可能尺寸太小。在后续的流计算（stream computing）中，RDD 数据包将进一步拆分成分区（partition），每一个分区将以多副本形式存储到 HDFS（Hadoop Distributed File System）存储系统上。RDD 尺度太小会导致分区碎片化，使得 HDFS 的管理效率降低，也导致整个流计算处理系统的吞吐率降低。

如果时间窗口取得太大（远大于 2 秒），所生成的 RDD 数据包尺寸较大，包含数据量大，只要拆分后分区（partition）的大小不超出 HDFS 的数据块（data block）的尺度，就可以得到有效处理。但时间窗口取得太大、RDD 数据包大的缺点是：尽管系统吞吐率有所提高，但对每条流数据而言，系统响应时间可能相对变长，不利于某些时间敏感性的应用要求。

3. 什么是 Spark 计算逻辑模型（抽象模型）Topology？什么是 Spark 计算物理模型（计算架构）？计算逻辑模型是如何映射到实际计算架构上的？

参考答案：Storm 并行计算是基于由 Spout（数据源）和 Bolt（处理节点）组成的有向拓扑图 Topology 来实现。这里，流数据是以 Tuple（基本数据单元，可看作一组各种类型的值域组成的多元组）的形式在 Spout 与 Bolt 之间流转。Spout 负责将输入数据流转换成一个个 Tuples，发送给 Bolt 处理。每个 Bolt 读取上游传来的 Tuples，向下游发送处理后的 Tuples。Storm 的 Topology 实际上定义了并行计算的逻辑模型（或者称抽象模型），也即定义了并行计算的步骤和流程。逻辑架构主要包含以下组件：

数据模型 Tuple、数据流 Stream、数据源 Spout、处理单元 Bolt、分发策略 Stream Grouping

Topology 只是一个 Storm 作业流程的逻辑设计，真正要实现这个逻辑设计，还需要 Storm 计算物理模型（计算架构）来支撑。这就涉及到如何在 Hadoop 平台上部署 Storm 的软件组件；如何将 Topology 定义的逻辑组件映射到物理节点上运行的进程或线程，实现多任务并行处理；如何提供系统容错和故障恢复功能。Storm 的计算架构（物理模型）包含如下组件：

Storm 主控程序 Nimbus、集群调度器 Zookeeper、工作节点控制程序 Supervisor、工作进程 Worker、执行进程 Executor、计算任务 Task

计算逻辑模型到实际计算架构的映射是通过系统为 Topology 分配工作节点（worker）、执行进程

（executor）和任务线程（Task）来实现的。即当一个计算作业的 Topology 提交给系统后，Storm 需要把物理计算资源（worker/executor/task）调度分配给 Topology 的 spout 和 bolt，已完成相应的计算步骤。

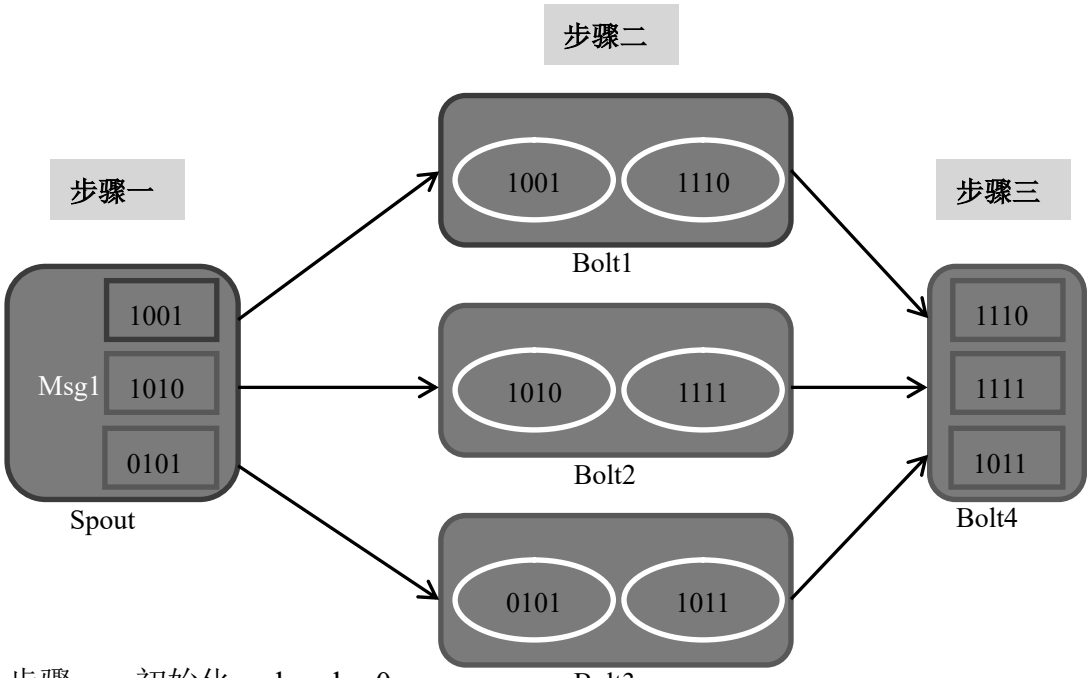
4. 根据 5.3 节的 Acker 工作机制，说明为什么 Acker 收到一条 Ack 消息使 ack-val=0 时，就意味着该条 tuple 的处理结束？

参考答案：Storm 可靠性要求发出的 Spout 每一个 tuple 都会完成处理过程，其含义是这个 tuple 以及由这个 tuple 所产生的所有后续的子 tuples 都被成功处理。Storm 设计了一个 Acker 机制来提供这个可靠性。其基本工作原理可简述为：

- Acker（一个独立运行的监督线程）设置 64-bit 校核参数 ack-val 的初始值为 0；
- Spout 发出的 tuple 都带有一个 64-bit 随机生成的 msgId；
- Bolt 处理完输入的 tuple，若创建了新的衍生 tuples 向下游发送，在向 Acker 发送消息确认输入 tuple 完成时，它会先把输入 tuple 的 msgId 与所有衍生 tuples 的 msgId（也是 64-bit 的全新 ID）作 XOR 运算，然后把结果 tmp-ack-val 发送给 Acker；
- Acker 每次收到新 tmp-ack-val，都会将其与目前的 ack-val 做 XOR 运算，并存储运算结果：ack-val = (ack-val) XOR (tmp-ack-val)，其结果是 ack-val 所含值总是目前 Tuple Tree 中所有 tuples 的 msgId 的 XOR 运算值
- Acker 如果发现运算后 ack-val=0（即 ack-val 目前值与 tmp-ack-val 值完全相同），就知道该 tuple 的计算结束。因为只有整个 Tuple Tree 在规定时间内（timeout）再无新的 tuple 产生，才会导致 tmp-ack-val 与 ack-val 值完全相同。

有无可能由于两个衍生 tuple 的 ID 值碰巧相同，造成 ack-val 在 Tuple Tree 处理完之前就变成 0？由于衍生 tuple 也是 64-bit 的随机数，两个 64-bit 随机生成的 ID 值完全一样的概率非常低，几乎可忽略不计，因此在 Tuple Tree 处理完之前 ack-val 为 0 的概率非常小，可忽略不计。

5. 图 15-30 的 Acker 算例如果扩展为如下的 1 个 Spout + 4 个 Bolts 情形，ack_val 的初始值仍然为 0。列出计算步骤验证：在步骤三结束时，ack_val = 0。



参考答案：步骤一：初始化 ack-val = 0;

ack-val = 1001 XOR 1010 XOR 0101 = 0110

步骤二：Bolt1 发送的 tmp-ack-val = 1001 XOR 1110 = 0111

收到 Bolt1 的 tmp-ack-val 后 ack-val 更新为：ack-val = (ack-val) XOR (tmp-ack-val) = 0110 XOR 0111 = 0001

Bolt2 发送的 tmp-ack-val = 1010 XOR 1111= 0101

收到 Bolt2 的 tmp-ack-val 后 ack-val 更新为：ack-val = (ack-val) XOR (tmp-ack-val) = 0001 XOR 0101 =

0100
Bolt3 发送的 tmp-ack-val = 0101XOR 1011= 1110
收到 Bolt3 的 tmp-ack-val 后 ack-val 更新为：ack-val = (ack-val) XOR (tmp-ack-val) = 0100 XOR 1110 = 1010
步骤三：Bolt3 发送的 tmp-ack-val =1110 XOR 1111 XOR 0111= 1010
收到 Bolt3 的 tmp-ack-val 后 ack-val 更新为：ack-val = (ack-val) XOR (tmp-ack-val) = 1010 XOR 1010 = 0
由此可知，步骤三结束时 ack-val=0，计算结束。

第 16 章 内存计算模式

1. 分布式缓存系统工作原理是?根据图 16-3 说明分布式缓存系统是如何大大提高系统访问速度的。
参考答案：分布式缓存系统的工作原理是：通过对存储数据的读写分离，实现对热点数据（即高频率访问数据）的快速读取。其具体做法是（参加图 16-3）：

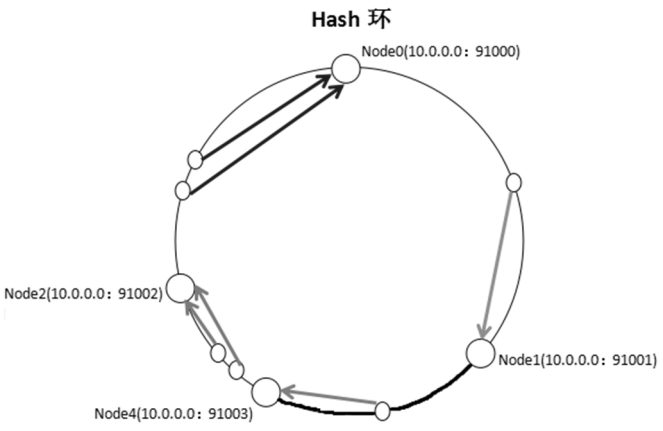
- 数据的读、写操作由不同的线程执行，写数据到数据库（RDBS），其操作的优先级别较低；
- 读数据首先从缓存系统（memcached）读取（绿色箭头路径）；
- 如果缓存 memcached 中没有找到所需数据，则从硬盘（RDBS）读出数据，放入缓存（蓝色箭头路径），然后读取数据；
- 运行一个缓存 memcached 数据的更新算法，使得尽可能多的数据可从缓存中读到。

分布式缓存系统实现了数据的读、写分离，将大部分读取数据操作主要放在速度极高的缓存上执行，从而大大提高了数据访问速度。

2. 分布式缓存系统有同步缓存（图 16-4）和异步缓存（图 16-5）两种模式，介绍它们各自的优缺点。
参考答案：同步缓存（图 16-4）的服务器集群中所有节点均保存一份相同的缓存数据，当某个节点有缓存数据更新的时候，会通知集群中其他机器更新内存或清除缓存数据。这种模式的优点是：应用程序和缓存常常部署在同一台服务器上，应用程序可从本地内存快速获取缓存数据，访问速度快，另外这种模式的数据一致性好，因为是同步更新。这种架构的问题是缓存数据量受限于单一服务器的内存空间，而且当集群规模增大时，同步更新集群所有节点的代价也昂贵。

异步模式（图 16-5）则采用一组专用缓存服务器，缓存与应用分离部署。在存放和访问缓存数据时，应用程序通过一致性 Hash 算法选择缓存节点，集群缓存服务器之间不通信，也不需要数据同步，因此集群规模可以很容易地实现扩容，具有良好的可伸缩性。但这种模式的数据一致性较差（因数据不同步），软件复杂度高。

3. 什么是一致性哈希算法（Consistent Hashing Algorithm）？试举例说明一致性哈希算法是如何解决扩容问题。
参考答案：异步模式下分布式缓存服务器节点之间无通信，不共享任何信息，只能被动地接受和执行来自于客户端的服务请求，因此数据缓存分布模式不是由服务器端控制，而是很大程度上由客户端的数据存储路由算法来实现。一致性哈希算法就是这样一种存储路径算法。



一致性哈希算法通过一个称做一致性 Hash 环的数据结构（如后图）实现键值对 Key 到缓存服务器 ID 的 Hash 映射，也即提供了存储时选择缓存服务器节点的方法。一致性 Hash 环跨越了长度区间[0, 2^{32}-1]，我们可以根据计算服务器节点 ID 计算其 HashCode（也落在[0, 2^{32}-1]区间），然后把该服务器节点定位在这个一致性 Hash 环上某一点（图中服务器 Node0，Node1，Node2 落在不同的环上不同位置）。

如果有一个数据项需要写入缓存区，首先对数据的 key 值计算得到其 HashCode（其分布也为[0, 2^{32}-1]），然后根据这个 HashCode 值在一致性 Hash 环上

顺时针查找距离这个数据项最近的服务器节点，这即是该数据项分配去的服务器节点。如果缓存服务器集群扩容，一样会影响到部分映射关系，但与余数 Hash 算法相比，一致性哈希算法有两个优势：
1）扩容时，一致性 Hash 算法只影响 Hash 环上新加入节点与顺时针方向它身后节点这个区间的数据项，也即是影响是局部的而非全局性的；
2）随着节点数增加，环上服务器节点排列越来越密，上述受影响区间会变得越来越小，原有映射关系保持正确性的概率越来越大，这就意味着服务器规模扩大反而使得一致性 Hash 算法的结果倾向稳定，这是算法的优势。

4. 与图 16-23 的读写分离数据库架构比较，图 16-24 的数据库集群架构解决了什么问题？图 16-25 的混合分区架构又解决了什么问题？

参考答案：图 16-23 的读写分离架构（读数据由内存数据库承担，内存中找不到才去访问磁盘数据库；写数据则是写入磁盘数据库，不影响内存数据库访问速度）解决了既保证高速访问速度、又能持久化存储数据的问题。与读写分离架构比较，图 16-24 的数据库集群架构解决了数据库的扩容问题。集群架构的每个服务器节点有自己独立的内存，与底层文件系统结合较好，使用了诸如列存储格式、数据压缩、分区等内存技术，计算能力更强，但技术复杂。

混合分区架构则在可扩容的集群架构之外解决了内存数据持久化问题。混合模式（Hybrid Shard）的每个分区由一个内存数据库节点和一个 MySQL 节点共同组成，形成水平方向的多分区、垂直方向的二级数据库（2-Level DB）。混合分区架构较好地解决了数据库扩容与持久化存储问题，分区计算模型能够支持超大规模数据的并行处理，内存数据库这一层也能保证高速访问。但系统结构和软件架构都趋于复杂，开发成本高。

5. RAMCloud 在 Master 节点内存中和 Backup 节点磁盘上存储有两套 Segment 体系。在进行内存清除提高内存使用率时，为何对 Master 内存和 Backup 磁盘要采用两种不同的清除机制（two-level Cleaning）？试解释说明这种不同的机制。

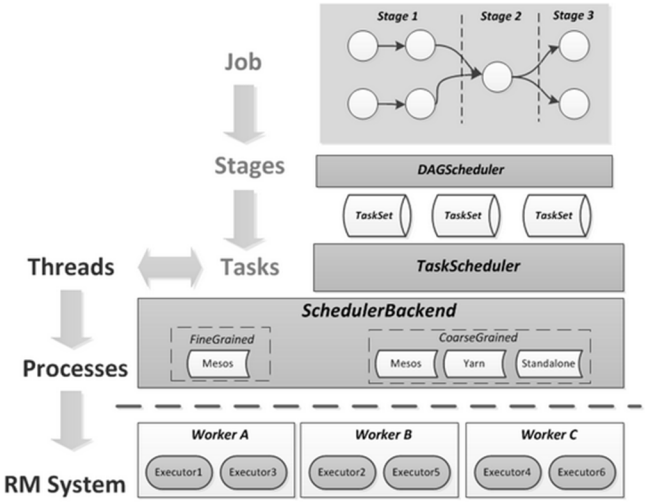
参考答案：RAMCloud 在 Master 内存和 Backup 磁盘上保留了两套 Segment 体系，因此，清除工作也需要对两套体系完成。如果把内存和磁盘的清除工作结合在一起进行会出现一个问题：当内存使用率高时（80~90%），与内存清除同时进行的磁盘清除会占用大量的网络带宽（这时磁盘使用率也高，迁移数据需要很大的开销），严重影响了集群写入数据的效率（Throughput）。因此，对读写性能差别很大的 Master 内存和 Backup 磁盘，RAMCloud 采用了两种不同的清除机制（two-level cleaning），其要点如下：

- First-level Cleaning：Segment Compaction（分区压缩）。在此阶段清除线程只对内存内 Segment 内部的 Logs 进行清理和压缩，释放清除后的内存空间供再次使用。注意，此阶段并未改变内存里的 Log 和 Segment 组成，哈希表中仍映射了各个 Log 和 Segment，只是改变了部分存储位置、压缩了空间。在此阶段不作磁盘清除，以保证磁盘写数据的带宽。从图 16-35 可看出，第一阶段清楚后的内存 Segment 状态与磁盘上的 backup Segment 状态不一致；
- Second-level Cleaning：Combined Cleaning（综和清除）。同时清除多个 Segments,并进行磁盘清除、同步。由于这之前内存 Segment 已经过压缩清理，因此这一阶段的操作不会占用过多网络或磁盘 I/O 带宽，不会对正常的写入操作带来太大影响。

6. 图 16-56 描绘了 Spark 的双层调度模型，即 Spark 的调度包括需求层（Application/Job/Stage/Task）和资源层（Worker/Executor/TaskThread）两层。试根据图 16-56 说明 Spark 调度算法是如何调度分配下层的计算资源满足上层的计算需求的。

参考答案：Spark 采用了双层调度模型，即整个调度架构分为计算需求调度（Application/Job/Stage/Task）和计算资源配置（Worker/Executor/TaskThread）两层。在上层需求调度层面又分为 Job 调度（由 DAGScheduler 承担）和 Task 调度（由 TaskScheduler 承担）；在下层计算资源配置层面则需决定每个节点 Worker 上启动多少 Executor 进程，分配多少资源，每个 Executor 内运行多少个 Task 线程等。

需求调度层与资源配置层之间是分离的，即下层的计算资源单位并不与上层的计算任务绑定。图16-54 清楚地描绘了上层计算单元包括 Job，Stage，Task 等，由 DAGScheduler 负责划分调派；下层包括 Worker，Executor，Thread 等计算资源单位，由 SchedulerBackend 负责分派，SchedulerBackend 可以是粗粒度（Standalone，YARN，Mesos 等模式下），也可以是细粒度（仅在 Mesos 模式下）。



上层计算任务的调度（即如何将具体的 RDD 分区映射到 Worker 上的 Task 线程，或者说如何将 Task 分发到集群的 Worker 节点上去执行）则是由 TaskSetManager 通过 TaskScheduler 与下层的计算资源管理器（SchedulerBackend）的协调来实现。如何分配下层的计算资源（Thread）去完成上层的计算任务（Task），则通过 TaskScheduler 的协调来完成。

其他

4. 简述决策树（C4.5）的算法原理。

参考答案：决策树 (Decision Tree）是直观运用概率分析的一种图解法。由于这种决策分支画成图形很像一棵树的枝干，故称决策树。决策树代表一类算法，C4.5 是其中比较典型的一种算法。C4.5 算法采用熵来选择属性，以构成决策分支；并采用后剪枝以抑制不必要的决策分支的生长。

5. 简述 Logistic 回归的算法原理。

参考答案：Logistic 回归是一种广义线性回归，因此与多重线性回归分析有很多相同之处。它们的模型形式基本上相同，都具有 wx+b 的形式，其中 w 和 b 是待求参数。logistic 回归则通过 logistic 函数（记为 L）将 wx+b 对应一个隐状态 p，p=L(wx+b),然后根据 p 与 1-p 的大小决定因变量的值。

5. 简述有序 Logistic 回归分析与线性回归分析的区别。

参考答案：在统计中，有序 Logistic（OL）回归被认为是由因变量二分法的线性回归模型的升级而来。OL 回归对因变量的分布没有任何假设，并且连续型或离散型的因变量皆可接受；其唯一的假设是比例优势假设，一般使用卡方检验考察模型的因变量是否能通过比例优势（proportional odds）假设检验。

而线性回归分析的假设比较苛刻，主要假设包括自变量之间不存在多重线性关系，以及因变量必须是正态分布。