* 1. 智慧地球与其主题

让世界的运转更加智能化，让个人、企业、组织、政府、自然和社会之间的互动效率更高。

将物联网和互联网融合，把商业系统和社会系统与物理系统融合起来，形成新的、智慧的全面系统，并达到运行“智慧”状态，提高能源利用率和生产力水平，改善人与自然之间的关系

主题：新锐洞察，智能运作，动态架构，绿色节能。

* 1. 什么是云？

云计算是一种共享的网络交付信息服务的模式，云服务的使用者看到的只有服务本身，而不用关心相关基础设施的具体实现。

* 1. 云提出的原因？

1.数据爆炸与信息孤岛

2.新业务与发展需求

3.基础设施昂贵以及不够灵活

4.资源有限

* 1. 什么是物联网以及与云的关系？

把感应器嵌入和装备到各种真实物体上，通过互联网连接起来，进而运行特定的程序，达到远程控制。

关系：云计算是实现物联网的核心

云计算促进物联网和互联网的只能融合，从而构建智慧地球

* 1. 云计算的特征
1. 弹性收缩
2. 快速部署
3. 资源抽象
4. 按用量计费
	1. 云计算与其他计算的比较异同
5. 主机系统与集中计算

特点：资源集中，计算、存储集中，集中部署，统一管理

1. 效用计算

把服务器及存储系统打包给用户使用，按照用户实际使用的资源量对用户进行计算。

强调的是IT资源，倡导IT资源按需供应，用户按使用量付费

1. 个人计算与桌面计算

具备独立的存储空间和处理能力，桌面计算是个人计算即可以完成大部分的个人计算需求的模式，不易于管理和维护

1. 分布式计算

一个应用运行在多台计算机之上。依赖于分布式系统

通过网络连接的多台计算机组成，每台计算机拥有独立的处理器以及内 存。互相协作，共同完成一个目标或者计算任务

1. 并行计算

高性能计算或超级计算，将一个科学计算问题分解为多个小的计算任务，并将这些小任务在并行计算机上同时执行，利用并行处理的方式达到快速解决复杂运算问题的目的，是云计算的重要组成部分

1. 网格计算

利用互联网把分散在不同地理位置的计算机组成一台“虚拟的超级计算机”，其中每台参与计算的计算机就是一个“节点”，而整个计算是由成千上万个“节点”组成的“一堆网格”。

* 1. 云计算的优势

优化产业布局，推进专业分工，提升资源利用率，减少初期投资，减低管理开销

* 1. 云计算对不同角色的影响

分为对基础软件提供商，硬件提供商，云提供商，云服务提供商，应用提供商，个人用户以及企业机构用户的影响。

* 1. 了解IBM的云计算产品，了解其实用性，并将具体产品归纳入云计算的三层架构中

根据列举的产品进行分析，没有固定答案

* 1. 了解其他的云计算产品，并举例对其进行总结归纳其优点和缺点

没有固定答案。

* 1. 什么是SaaS？

软件是服务，即软件的服务化

* 1. 软件即服务的原因？

软件只有研发成本，没有生产成本.软件是无形商品，没有损耗的概念,软件是需要持续的服务、维护和升级，才能正擦很难过发挥其内在的价值

* 1. 比较SaaS与ASP模式？

看问题的角度不同，关注的目标不同。

* 1. SaaS针对用户和软件提供商的优势？

给用户的优势：

风险减小，成本降低，先天防毒，节省销售成本，节省维护成本，稳健的经营模式。

* 1. 提出SaaS的缺点？

依赖互联网，数据安全性，数据保密性.(也可以有其他的观点)

* 1. 什么是SaaS的规模效应？

使得每个客户的平均运行和维护成本将降低

* 1. 云计算的成熟度模型

定制开发，可配置，高性能的多租户架构，可伸缩的多租户架构。

* 1. 四级成熟度模型的优缺点比较？

可以根据自己的理解进行比较。

* 1. 根据项目选择成熟度模型的因素？

产品所面向的客户群的特征与需求，产品的租户数量级别，团队的开发能力与愿意付出的开发/改造成本。

* 1. 实现SaaS的渐进步骤？

实现多组合架构，实现多租户架构的高性能，实现可配置，实现伸缩性。

* 1. SaaS成熟度模型的渐进化过程

项目，产品，多租户，高性能的多租户，可配置的多租户，可伸缩的多租户。

* 1. 做项目和产品的比较？

项目面向单一客户，产品面向大众或者行业

* 1. 做项目的开发流程

可以根据其他的课程进行总结。

* 1. 什么是可配置性？

 兼顾多方的共性和个性化需求

* 1. 可配置化的技术

MDA（模型驱动架构）

利用元数据建模，可以方便灵活实现可配置化

* 1. 了解MDA模型

[MDA](http://www.omg.org/mda)是由[OMG](http://www.omg.org/%22%20%5Co%20%22OMG%22%20%5Ct%20%22_parent)（Object Management Group，国际对象管理集团）于2001年提出来的。其核心思想是抽象出与实现技术无关、完整描述业务功能的核心平台无关模型（PIM，Platform Independent Model），然后针对不同实现技术制定多个转换规则，通过这些转换规则及辅助工具将 PIM 转换成与具体实现技术相关的平台相关模型（PSM，Platform Specific Model），最后将经过充实的 PSM 转换成代码。通过PIM和PSM，[MDA](http://www.omg.org/mda%22%20%5Co%20%22MDA%22%20%5Ct%20%22_parent)的目的是分离业务建模与底层平台技术，以保护建模的成果不受技术变迁的影响。

* 1. 什么多租户？

 多个租户共用一个实例

* 1. 多租户的实现方案？

独立数据库：一个Tenant一个 Database

共享数据库，隔离数据结构：多个或所有租户共享Database，但一个Tenant一个Schema

共享数据库，共享数据结构：租户共享同一个Database、同一个Schema，但在表中通过TenantID区分租户的数据

* 1. 分别比较多租户实现方案的优缺点？

独立数据库的隔离界别高，共享级别低，安全性高。

共享数据库，隔离数据架构的隔离级别中等，共享级别中等，安全性中等。

共享数据库，共享数据架构的隔离级别低，共享级别高，但是安全性低。

* 1. 考虑多租户安全设计的实现
1. 系统级：使用HTTPS协议以SSL（Security Socket Layer）交换数据，增强通信安全；通过数字签名防止传输过程篡改；对用户身份识别的UserToken使用DES算法数据加密；业务数据定时自动备份。
2. 程序级：完整的权限配置，包括功能权限和数据权限；客户端输入校验，防止JS攻击、XSS攻击、SQL注入等；辅助安全设计，比如密码控件、图片验证码、手机确认码等
	1. 从哪些方面实现高性能

数据库层，应用层以及Web层实现性能优化。（也可以有自己的观点）

* 1. 数据层性能优化的方案以及其实现方案

建立合适的索引，消除大数据表连接，避免复杂SQL

* 1. 应用层的性能优化方案以及针对解决的问题

 Cache：避免对数据库的频繁访问

统计和报表计算：改变实时计算

基于Tenant的搜索引擎：针对模糊查询

异步操作：提升用户体验，降低系统的并发负载

* 1. 使用Cache方案的特征

Cache的数据应该是读多写少

Cache数据易失，通过在应用中增加一层逻辑保证可靠性：当Cache中访问不到时，访问数据库，重新将数据载入Cache

* 1. Cache方案涉及到的技术并比较其优缺点

MemCached：1.基于C/S架构2.只有一份数据Copy，不需要数据同步

优势：不需要数据同步，避免复杂的多播等技术

Jboss Cache：基于Jgroup多播的分布式Cache

优势：Cache读取基于本地的Memory，性能更高

（可以有自己的观点，也可以从网上查资料）

* 1. Web层性能优化方案

Web开发的性能优化策略。Http服务器的性能优化策略。

* 1. 提供自己的性能优化方案，某一方面可以具体到实现方案

根据自己的理解提供答案。

* 1. 为什么实现性能监控

主动发现性能问题：能够实时统计出应用在各种场景下（不同地区、不同用户、不同页面）的性能数据并及时报警

* 1. 根据自己的理解性能监控应该实现哪些方面（无答案）
	2. 可配置性以及从哪些方面实现可配置性

数据可配置，功能可配置，界面可配置，流程可配置。

* 1. 数据可配置性的方案

定制字段，预分配字段，名称值对。

* 1. 比较数据可配置性方案的优缺点



* 1. 什么是功能可配置性

按需使用，按需付费，1.允许不同的用户购买不同的功能集合，同时在线使用。2.同一个用户再不同时期，有不同 需要，可以选择不他那个的功能集合并为之付费

* 1. 功能可配置性的实现方案

原子功能划分，功能包设计，功能使用校验。

* 1. 什么是功能包设计？以课件中的原子功能的划分为例，实现功能包的设计

功能包设计：就是根据用户的类型和系统的业务逻辑，综合考虑用户的使用场景和使用习惯，将原子功能进行组合，设计成功能包

* 1. 界面可配置性方案设计

系统菜单可配置，页面内容可配置。

* 1. 配置元数据管理的方面

配置元数据，租户配置数据，配置元数据服务。

* 1. 什么是可伸缩性
1. 性能、容量方面的可扩展
2. 系统的伸缩
	1. 应用层服务器层实现水平扩展的方案比较

 

* 1. 数据库层的水平扩展的方案比较



* 1. 为什么需要不同架构？

云计算需要清晰的架构来实现不同类型的服务及满足用户对这些服务的各种需求。

* 1. 共有那些架构？
	2. 基础设施层
	3. 平台层
	4. 应用层
	5. 什么是基础设施层？请举例说明
1. 是经过虚拟化后的硬件资源和相关管理功能的集合
2. Amazon EC2
	1. 4、基础设施层的基本功能
* 资源抽象
* 资源监控
* 负载管理
* 数据管理
* 资源部署
* 安全管理
* 计费管理
	1. 什么是平台层？请举例说明

交付给用户的是丰富的“云中间件”资源，这些资源包括容器、数据库和消息处理等

Google : App Engine

* 1. 平台层的基本功能
* 开发测试环境
* 运行时环境
* 运营环境
	1. 什么是应用层？请举例说明
* 云上应用软件的集合
* Yahoo :Hadoop
	1. 应用层的特征
* 这些应用能够通过浏览器访问，或者具有开放API，允许用户或者瘦客户端使用
* 用户在使用云服务时，不需要进行一次性投入，只需要在使用的过程中按照其实际的使用情况付费
* 云应用要求高度的整合，而且云应用之间的整合能力对于云应用的成功至关重要
	1. 应用层的分类并解释
* 标准应用：采用多租户技术为数量众多的用户提供相互隔离的操作空间，提供的服务是标准的、一致的。
* 客户应用：应用开发好的标准的功能模块，允许用户进行不限于界面的深度定制
* 多元应用：应用一般由独立软件开发商或者开发团队在公有云平台上搭建，是满足用户某一类特定需求的创新型应用
	1. 比较Amazon EC2、Google App Engine、Microsoft Azure、Yahoo Hadoop几种云服务的异同。

参考下图说明



* 1. 简述对于Google而言，研发云计算的源动力
* 海量用户+海量数据
* 需要具备较强的可伸缩性
* 如何又快又好地提供服务
	1. 什么是“浏览器=操作系统”的思想？
* 应用向互联网迁移
* 数据向互联网迁移
* 计算能力向互联网迁移
* 存储空间向互联网迁移
	1. 简述Google云计算平台架构
* 文件存储，Google Distributed File System，GFS
* 并行数据处理MapReduce
* 分布式锁Chubby
* 结构化数据表BigTable



* 1. 什么是MapReduce？

一个软件架构，是一种处理海量数据的并行编程模式

* 1. 什么是Map？什么是Reduce？
* Map把一个函数应用于集合中的所有成员，然后返回一个基于这个处理的结果集
* Reduce对结果集进行分类和归纳
	1. 什么是Hadoop？

Hadoop是一个分布式系统基础架构，由Apache基金会开发。用户可以在不了解分布式底层细节的情况下，开发分布式程序。充分利用集群的威力高速运算和存储。

* 1. Hadoop与Google云框架的关系



* 1. Hadoop工作基本流程介绍



* 1. 列举至少5个Hadoop成员，并说明其作用
* Core：一系列分布式文件系统和通用I/O的组件和接口（串行化、Java RPC和持久化数据结构）
* Avro：一种提供高效、跨语言RPC的数据序列系统，持久化数据存储。
* MapReduce：一种 编程模型， 是处理和产生大规模数据集的一种整合实现
* HDFS：分布式文件系统
* Pig：一种数据流语言和运行环境，用以非常大的数据集。
* Hbase：一个分布式、列存储数据库。
* HDFS：一个分布式、高可用性的协调服务。
* Hive：分布式数据仓库
* Chukwa：分布式数据收集和分析系统。
* Mahout：机器学习和数据挖掘库
	1. 思考：为什么在安装Hadoop之前需要先配置无密码SSH连接？

为了让master通过ssh能顺利访问master和slaver

* 1. 为什么需要设计GFS?
* 需要一个支持海量存储的文件系统
* 节约开销，使用一堆廉价且不可靠的硬件
	1. 鉴于海量数据的分布式存取需求，与普通单机存取相比，有哪些特殊点？
* 硬件出错是正常而非异常
	+ - 系统应当由大量廉价、易损的硬件组成
		- 必须保持文件系统整体的可靠性
* 主要负载是流数据读写
	+ - 主要用于程序处理批量数据，而非与用户的交互或随机读写
		- 数据写主要是“追加写”，“插入写”非常少
* 需要存储大尺寸的文件
	+ - 存储的文件尺寸可能是GB或TB量级，而且应当能支持存储成千上万的大尺寸文件
	1. 绘制GFS原理图



* 1. 文件分块太大或者太少的缺点
* 太大：数据块过大，造成网络传输负担
* 太小：数据块过小，查找不方便，需要存储的元数据过大
	1. GFS架构的问题以及解决方法
* 单一Master问题
* 解决方法
	+ - 尽可能减少数据存取中Master的参与程度
		- 不使用Master读取数据，仅用于保存元数据
		- 客户端缓存元数据
		- 采用大尺寸的数据块（64M）
		- 数据修改顺序交由Primary Chunk Server完成
	1. Master节点的任务
* 存储元数据
* 文件系统目录管理与加锁
* 与ChunkServer进行周期性通信
* 数据块创建、复制及负载均衡
* 垃圾回收
* 陈旧数据块删除
	1. GFS架构的特点
* 采用中心服务器模式
* 不缓存数据
* 在用户态下实现
* 提供专用的访问接口
	1. GFS架构的容错方法
* Chunk Server容错
	+ - 每个Chunk有多个存储副本（通常是3个），分别存储于不通的服务器上
		- 每个Chunk又划分为若干Block（64KB），每个Block对应一个32bit的校验码，保证数据正确（若某个Block错误，则转移至其他Chunk副本）
* Master容错
	+ - 三类元数据：命名空间（目录结构）、Chunk与文件名的映射以及Chunk副本的位置信息
		- 前两类通过日志提供容错，Chunk副本信息存储于Chunk Server，Master出现故障时可恢复
	1. 什么是HDFS
* HDFS全名是Hadoop Distributed File System
* Hadoop是Apache的一个子计划
* HDFS是Hadoop的一个子计划
* 分布式文件系统
	1. HDFS各种节点及其作用
* Namenode
	+ - 负责处理来自client端的档案存取要求
		- 负责储存整个HDFS中各档案之metadata
* Secondary Namenode
	+ - 协助Namenode整并并跟新当中的metadata
		- 当Namenode挂点时，可手动令其做Namenode用
* Datanode
	+ - 为data本身存放之处
		- 听取Namenode的安排，然后让client存取档案
	1. 什么是函数式编程

函数式编程是种编程范型，它将电脑运算视为函数的计算。函数编程语言最重要的基础是 λ 演算（lambda calculus）。而且λ演算的函数可以接受函数当作输入（引数）和输出（传出值）。

* 1. 函数式编程特点
* Functional操作不修改数据结构
* –创建新数据结构
* 原始数据以原形式存在，不会被修改
* 对于程序设计，数据流是隐式的
* 操作执行的次序无关
	1. 简述MapReduce工作的步骤
1. 用户程序中的MapReduce库首先将输入文件分割成M份（一般情况下，每份有16MB到64MB大小）。然后在集群中的电脑上启动程序的多个副本。
2. 一个程序的副本比较特殊 -- --master。其余的worker由master分配。一共有M个map任务和R个reduce任务需要分配。master挑选出空闲的worker，并非配给它们map任务或者reduce任务。
3. 分配了map任务的worker读取相应的输入数据块。它从输入数据中解析出key-value元组并将每个元组传送到用户定义的map函数中。由map函数得到的中间元组缓存在内存中。
4. 系统周期性地将缓存的元组写入到本地磁盘中，由分组函数将其分成R组。这些缓存元组在本地磁盘中的位置被传送到master中，master负责将这些位置传到相应的reduce worker中。
5. 当reduce worker被master告知这些位置之后，它通过远程过程调用去读取存储在map worker本地磁盘上的这些缓存数据。当一个reduce worker已经读取完所有的中间元组之后，它整理这些元组，使得所有具有相同key值的元组能够被分到一起。分组是必须的因为一般来说，许多不同key值的元组被分配到相同的reduce任务中。如果中间元组的数目太大以至于不可以存储到内存之中，则启用额外的整理过程。
6. reduce worker迭代器遍历所有排序后而具有相同key值的元组，将key值以及相应的一组中间值传送到reduce函数中去。系统将reduce函数的输出写入到最终的输出文件中去。
7. 当所有的map和reduce任务都结束之后，系统通知用户程序。MapReduce过程结束。
8. 当以上过程成功之后，mapreduce操作的结果被写在R个输出文件中。一般来说，用户不需要将这些文件组装在一起，它们经常作为输入数据传送到另外一个mapreduce过程中去，或者被其他能够处理多个文件的分布式应用调用。
	1. 没有Reducer时，整个MapReduce过程相当于什么操作

排序操作

* 1. 什么是Combiner？为什么需要有Combiner？

Combiner 就是一个Mini Reducer，它在执行Map 任务的节点本身运行，先对Map 的输出做一次简单Reduce，使得Map 的输出更紧凑，更少的数据会被写入磁盘和传送到Reducer。

原因：解决单个map任务产生大量相同key的中间元组，从而阻塞网络的问题

* 1. 是不是所有MapReduce任务都适宜启动Combiner以便于节省时间？

不是。有的操作如果运行两次就会产生错误。

* 1. 如何解决worker故障问题

Master 周期性的ping每个worker。如果master在一个确定的时间段内没有收到worker返回的信息，那么它将把这个worker标记成失效

重新执行该节点上已经执行或尚未执行的Map任务

重新执行该节点上未完成的Reduce任务，已完成的不再执行

* 1. 如果解决master故障问题
* 定期写入检查点数据
* 从检查点恢复
	1. 面对有可能因为某个worker一直工作不完成导致任务无法结束的问题，怎么解决？
* 在即将完成时，备份任务
* 多个worker同时进行相同的任务
* 任何一个完成均可
	1. 简述本地处理优化原理
* 向GFS询问获得输入文件blocks副本的位置信息
* Map tasks 的输入数据通常按 64MB来划分 (GFS block 大小)
* 按照blocks所在的机器或机器所在机架的范围 进行调度
	1. 什么是Chubby？
* 主要用于解决分布式一致性问题
	+ 在一个分布式系统中，有一组的Process，它们需要确定一个Value。于是每个Process都提出了一个Value，一致性就是指只有其中的一个Value能够被选中作为最后确定的值，并且当这个值被选出来以后，所有的Process都需要被通知到
* 粗粒度的分布式锁服务
	+ Chubby是Google为解决分布式一致性问题而设计的提供粗粒度锁服务的文件系统
	+ 其他分布式系统可以使用它对共享资源的访问进行同步
	1. Chubby的设计目标
* 高可用性
* 高可靠性
* 支持粗粒度的建议性锁服务
* 支持小规模文件直接存储
	1. 为什么说“Chubby系统本质上就是一个分布式、存储大量小文件的文件系统”
* Chubby中的锁就是文件
* 在GFS的例子中，创建文件就是进行“加锁”操作，创建文件成功的那个server其实就是抢占到了“锁”
* 用户通过打开、关闭和存取文件，获取共享锁或者独占锁；并且通过通信机制，向用户发送更新信息
	1. Chubby的应用
* 主节点选举
* 独占锁
* 共享锁
* 数据存取应用
	1. 什么是BigTable？
* 基于GFS和Chubby的分布式存储系统
* 对数据进行结构化存储和管理
	1. 简述BigTable数据模型中的行
* 每行数据有一个可排序的关键字和任意列项
* 字符串、整数、二进制串甚至可串行化的结构都可以作为行键
* 表按照行键的“逐字节排序”顺序对行进行有序化处理
* 表内数据非常‘稀疏’，不同的行的列的数完全目可以大不相同
* URL是较为常见的行键，存储时需要倒排
	1. 简述BigTable数据模型中的列
* 特定含义的数据的集合，如图片、链接等
* 可将多个列归并为一组，称为族（family）
* 采用 族:限定词 的语法规则进行定义
* 同一个族的数据被压缩在一起保存
* 族是必须的，是BigTable中访问控制的基本单元
	1. 在Google云架构中Chubby的作用
* 为GFS提供锁服务，选择Master节点；记录Master的相关描述信息
* 通过独占锁记录Chunk Server的活跃情况
* 为BigTable提供锁服务，记录子表元信息（如子表文件信息、子表分配信息、子表服务器信息）
* （可能）记录MapReduce的任务信息
* 为第三方提供锁服务与文件存储
	1. 在Google云架构中GFS的作用
* 存储BigTable的子表文件
* 为第三方应用提供大尺寸文件存储功能
* 文件读操作流程
	+ - API与Master通信，获取文件元信息
		- 根据指定的读取位置和读取长度，API发起并发操作，分别从若干ChunkServer上读取数据
		- API组装所得数据，返回结果
	1. 在Google云架构中BigTable的作用
* 为Google云计算应用（或第三方应用）提供数据结构化存储功能
* 类似于数据库
* 为应用提供简单数据查询功能（不支持联合查询）
* 为MapReduce提供数据源或数据结果存储