1.什么是云计算？

- 指应用以服务形式通过互联网交付使用，数据中心的硬件和软件能提供这些服务。

- 一种能够便捷地按需访问共享可配制计算资源池(如网络、服务器、存储、应用、服务)的服务模式，并只需要很少的管理工作或与服务供应商的很少交互就可以快速提供和发布这些服务。

- 由一组互连的虚拟机组成的并行和分布式系统，依据服务供应商和消费者之间协商确定的服务等级协议，动态配置和提供一种或多种统一的计算资源。

2、云计算的特点
超大规模、虚拟化、高可靠、通用性、可伸缩、按需服务、及其廉价

3、云计算参考模型
IaaS、PaaS、SaaS

4、云计算部署
公有云、私有云、混合云

5.云计算核心技术

（1）分布式系统

分布式系统是一些独立计算机集合，对用户来说，系统就像一台独立计算机。体现了云计算的特性：扩展性、并发性和持续可用性。

（2）虚拟化

将一些计算的基本构件(如服务器、存储和网络等)进行抽象化的方法。提高硬件利用率、降低能耗、提高了IT运维效率，系统管理人员减少、操作系统和硬件的解耦。

（3） Web 2.0

Web是云计算提供服务的主要接口，实现交互信息的共享、协同，以用户为中心的设计和应用组合。 Web2.0用于应用程序的开发平台，使Web页面具有交互性和灵活性，桌面应用基于Web访问，提升了用户体验；Web2.0动态性和松耦合性；Web是支持云计算需求的成熟平台。Web 2.0应用程序和框架提供了丰富的互联网应用，并且IT基础设施也可以通过Web接口提供。

（4）面向服务的计算

云计算系统的核心参考模型，它将**服务**作为应用和系统开发的主体模块。支持快速、低成本、灵活、可交互和可扩展的应用和系统开发。**服务**具有松耦合、可重用、独立于编程语言和位置透明特性。

（5）效用计算

定义了一种计算服务的提供模式，将存储、计算能力、应用程序和基础设施等资源封装为服务，并基于使用量付费。

1.并行计算和分布式计算

（1）并行计算

- 将计算任务分配给几个共享相同内存的处理器的计算模式；
- 组件的同构性：每个处理器都是相同类型的，且拥有相同的处理性能；
- 基于共享内存这一概念的架构：物理内存系统;由库、特定的硬件和高效的网络基础设施组成的系统。

（2）分布式计算

- 将计算任务进行划分，并在不同计算单元中同时执行的架构或系统；
- 通常指计算单元位置不同，且这些单元在硬件和软件功能上也可能各不相同。

2、并行处理硬件架构

SISD、SIMD、MISD、MIMD（共享内存和分布式内存）

3、并行编程方法

数据处理（SIMD）、处理并行（多个操作在不同处理器上）、农场主和工作者

4、并行级别

大（进程/程序员）、中（函数或过程/程序员）、小（循环或指令块/编译器）、极小（指令/处理器）

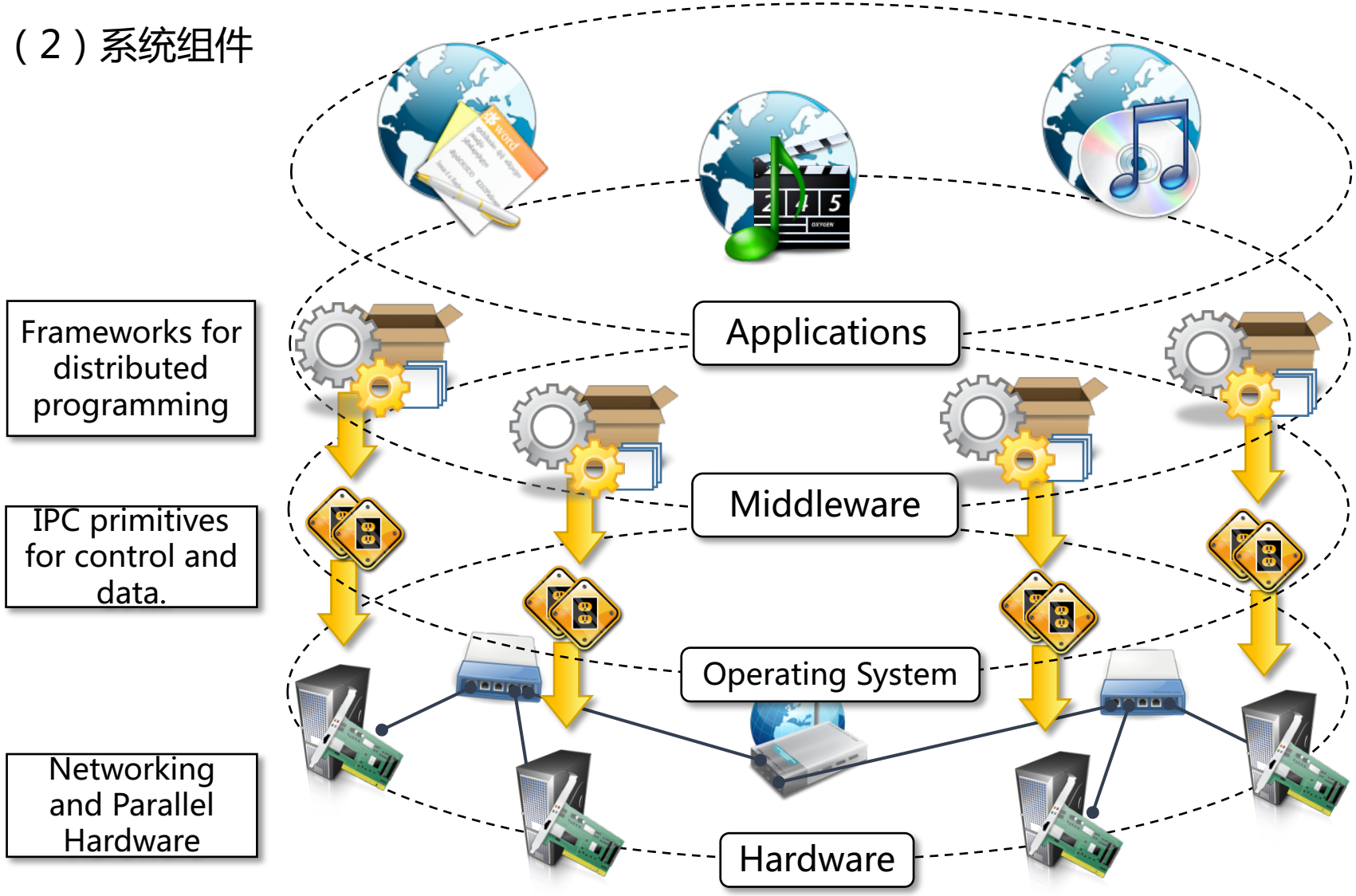
1.分布式计算

（1）定义

- 分布式系统是独立计算机的集合，对于用户来说是一个整体系统。多种类型的分布式计算系统，统一使用和集成分布式资源。

- 分布式系统组件位于仅能通过**传递消息**来通信和协调活动的网络计算机中。

（2）系统组件



- 底层计算机和网络硬件构成了物理基础设施，这些组件由操作系统直接管理。
- 操作系统负责提供基础服务，用于进程间通信、进程调度和管理、文件系统和本地设备的资源管理。
- 将网络和计算机这两层合并成一个平台，配置特定软件组成一个分布式系统。

（2）系统组件

- 把公认的标准应用到操作系统层甚至硬件网络层中，利用异构组件可以很容易地构造一个统一的集成系统.

- 中间件层利用操作系统层提供的服务，构建了一个开发和部署分布式应用程序的统一环境。如中间件层可开发协议、数据格式，以及用于开发分布式应用程序的编程语言或框架，为开发人员提供了统一接口，屏蔽了底层的异构性。

- 顶层是利用中间件设计和开发的应用或服务。应用层具有通过本地或Web浏览器可访问的图形用户接口(GUD) 。

- 硬件和操作系统层组成了一个或多个数据中心的最基本结构，其中服务器通过高速网络部署和连接在一起。硬件由操作系统管理，操作系统提供了基本的管理计算机和网络的能力。核心业务逻辑在管理虚拟化层的中间件上实现，虚拟化部署在物理机上，提供可定制的应用运行环境。

2、架构模式

- 架构模式主要用于决定**组件**和**连接器**，组件、连接器及其结合条件一起作为架构模式的实例。

- 组件代表一个封装了系统的功能或特性的软件单元。组件实例可以是程序、对象、进程、管道或过滤器。
- 连接器是一种允许组件之间进行合作和协调的通信机制。连接器不封装在单一实体中，而是以分布式方式由多个系统组件实现。

- 设计模式有助于软件工程师和开发者为如何在应用中构建组件关系，以及理解软件应用程序的内部结构达成共识。架构模式在软件系统的整体架构中起到相同的作用，包括软硬件部署等。
- 分布式系统的架构模式可以帮助我们理解系统中组件的不同作用，以及如何跨多机分配组件。

- 架构模式分为两大类：软件架构模式、系统架构模式。

3、软件架构模式

- 基于软件组件的逻辑结构。提供整个系统的直观视图，而忽略系统的物理部署。

- Garlan和Shaw给出软件架构：

1）数据中心架构

- 将数据作为软件系统的基本元素，可访问的共享数据是以数据为中心架构的核心特征。
- 仓库架构模式（两个组件）：表示系统当前状态的中央数据结构，以及操作中央数据的独立组件集合。
- 黑板架构（三个组件）：知识源（用于更新黑板中维护的知识库的实体）、黑板（表示在知识源中共享的、存储应用知识库的数据结构）、控制（触发器和过程的集合，负责管理与黑板的交互，更新知识库的状态）。

2）数据流架构

- 由数据从组件到组件的有序移动决定的，数据的移动也是组件之间的通信方式。
- 顺序批处理模式：由独立程序组成的有序序列的按序执行。
- 管道过滤器模式：顺序批处理模式的一种变形，它将软件系统的活动描述成数据变换的序列。加工链中的每个组件称为过滤器，用数据流表示过滤器之间的连接。

3）虚拟机架构

- 利用抽象的执行环境(通常称为虚拟机)模拟硬件或软件中不可见的功能。

- 基于规则的系统：将抽象执行环境看作一个推理机，以适当的规则或谓词的形式表达程序。应用程序的输人数据通常表示为一组声明或事实，**推理机使用这些声明或事实激活规则或谓词**，处理数据。

- 解释器：一个引擎将伪代码解释成解释器可识别的格式，如Java虚拟机。

- 命令-语言处理器

4）调用和返回架构
一连串方法调用，这些方法调用的执行和组合标识了一个或多个操作的执行。

自上而下结构：以命令式编程开发的系统为代表，采用分面治之的方法解决问题。由一个主程序通过调用子程序或过程来完成任务。

面向对象结构：利用面向对象编程(OOP) 方法设计和实现系统， 包含的系统类型非常广泛。系统用类来声明，用对象实现。类通过定义组件的类型，指定数据的状态及对数据的操作。

分层结构：按层设计和实现软件系统，给出了系统抽象的不同级别。通常每一层最多与两层进行操作，其低一层的抽象层次和高一层的抽象层次。

4、系统架构

包括组件的物理组织结构以及分布式基础设施上的进程。

1)客户端/服务器
通信是单向的：客户端向服务器发出请求，服务器在处理请求之后返回响应结果
可以是多个客户端组件向一个处于被动等待状态的服务器发送请求。

客户端/服务器模型包括三部分：显示、应用逻辑和数据存储。
客户端和服务器端：客户端提供用户接口，主要处理显示层的所有功能；
服务器负责应用逻辑和数据存储层功能；

三层结构IN层结构：将数据显示、应用逻辑和数据存储分为三个层次

2）对等模型
一种对称的架构，模型中所有组件都称为对等点，具有相同的功能，包含了客户端/服务器模型中的客户端与服务器的功能。

5、进程间通信模型

分布式系统由一组通过网络交互的并发进程组成。因此进程间通信是分布式系统设计和实现的基础。

进程间通信既用于交换数据和信息，也用于协调进程活动。它将分布式系统的不同组件连接起来，使其成为一个系统。

进程之间交互的模型有多种，分别对应着不同抽象。最常见的包括共享存储、远程过程调用(RPC) 和消息传递。

在低层，进程间通信由网络编程工具实现。套接字(socket) 是最常用的分布式进程间，实现通信信道原语。

5、进程间通信模型

（1）基于消息的通信

消息在实现分布式计算的技术和模型演进过程中发挥了重要作用。

分布式系统定义为“**联网计算机上的组件通过传递消息，进行沟通和协调组件活动的系统**”。

消息在这里指任何从一个实体传递到另一个实体的信息。它包含任意形式的数据表示，数据有大小和时间限制，是对远程程序、序列化的对象实例、或通用消息，的调用返回的结果。因此， 基于消息的通信模型可以用来指代任何进程间通信模型。

消息在这里指任何从一个实体传递到另一个实体的信息离散量。它包含任意形式的数据表示，数据有大小和时间限制，是对远程过程，序列化的对象实例或通用消息的调用返回的结果。

编程模型：

- **消息传递**

  用消息的概念作为模型的主要抽象。**实体交换信息，以消息的形式对被交换的数据进行编码。**不同的模型，其结构和消息的内容有所不同。

  如：消息传递接口(MPI，实现**并行进程或线程**之间的数据交换和同步) 和OpenMP（用于**共享内存并行系统**的多处理器程序设计）。

- **远程过程调用(RPC)**

  在单一进程以外，扩展了过程调用的概念， 由此触发代码在远程进程中的执行。

- **分布式对象**

  RPC模型基于面向对象范式的实现， 将方法的远程调用看作对象。如通用对象请求代理架构(CORBA) 、组件对象模型(COM、DCOM和COM+) 、Java远程方法调用(RMI) 和远程NET。

- **分布式代理和活动对象**

  基于代理和活动对象的编程范式，涉及对实例的定义，不管它们是否是对象代理，也不考虑是否有请求。**对象都由自己的控制线程来执行动作。**

- **Web服务**

  RPC概念在HTTP上的实现， 允许不同技术开发的组件进行交互。**Web服务作为Web服务器上的一个远程对象， 在HTTP请求时被转化为方法调用**， 并使用特定的协议进行封装，如简单对象访问协议(SOAP) 或表征状态转移协议(REST) 。

6、分布式计算技术

•远程过程调用

RPC是执行客户请求过程的基本方法，允许在进程和单一内存地址空间之外完成过程调用。被调用的过程可以在同一个系统上，或是在网络内的不同系统上。封送处理 (marshaling，将参数和返回值转换成网络字节流)和解封是RPC的一个重要方面。

•分布式对象框架

允许对象分布在异构的网络中，框架的便捷性使其如同在同一地址空间内一样。

7、面向服务的计算

以服务的形式组织分布式系统，此服务表征了构建系统的主要抽象。

面向服务将应用系统表示为在面向服务架构(SOA) 中，协同工作的服务的集合。

Web服务是开发基于SOA系统的常用的方法。

Web服务作为实现云计算系统的重要组件， 通过互联网建立用户与系统之间的主要交互通道。

8、Web服务成为实现SOA的首选技术，主要原因有：

首先 Web服务允许跨越不同平台和编程语言进行互操作；

第二，Web服务基于众所周知的且与供应商无关的标准，如HTTP、SOAP ，XML和WSDL ；

第三， Web服务提供了一种直观和简单的方法来连接异构的软件系统，可以在一个分布式环境中快速组合服务；

最后,Web服务提供了在工业环境中使用的企业业务应用所需的功能。

9、虚拟化

- 虚拟化通常指硬件虚拟化，在为基础设施即服务(IaaS) 的云计算方案中起到关键作用。
- 在操作系统级、编程语言级和应用级实现了虚拟化环境。
- 存储、内存和网络的虚拟化

10、虚拟化技术受到关注

- 更高的性能和计算能力：PC有足够的资源承载虚拟机管理器。
- 未充分利用的硬件和软件資源：资源有限或零星使用，通常10-15%。
- 空间不足：硬件的利用不足，产生了所谓的服务器整合技术，虚拟化技术是其关键。
- 绿色节能措施：通过服务器整合来减少服务器的数量。
- 管理成本上升。减少了所需的服务器数量，从而降低了管理成本。

11、虚拟化环境特点

- 虚拟化是一个广义的概念，指的是构建硬件、软件环境、存储或网络的虚拟环境。
- 虚拟化环境中包括三个主要组件：客户机、主机和虚拟化层。
  - 客户机表示虚拟化的机器，与虚拟化层交互。
  - 主机管理客户机的原始环境。
  - 虚拟化层负责重新创建相同的或不同的客户机运行环境。

（1）安全性

- 以完全透明的方式控制客户机的执行，提供一个安全的、可控的执行环境。
- 虚拟机是客户机运行的模拟环境。虚拟机管理器控制和过滤客户机的活动。
- 主机公开的资源可被客户机隐藏或被简单地保护。

11、虚拟化环境特点

（2）执行管理

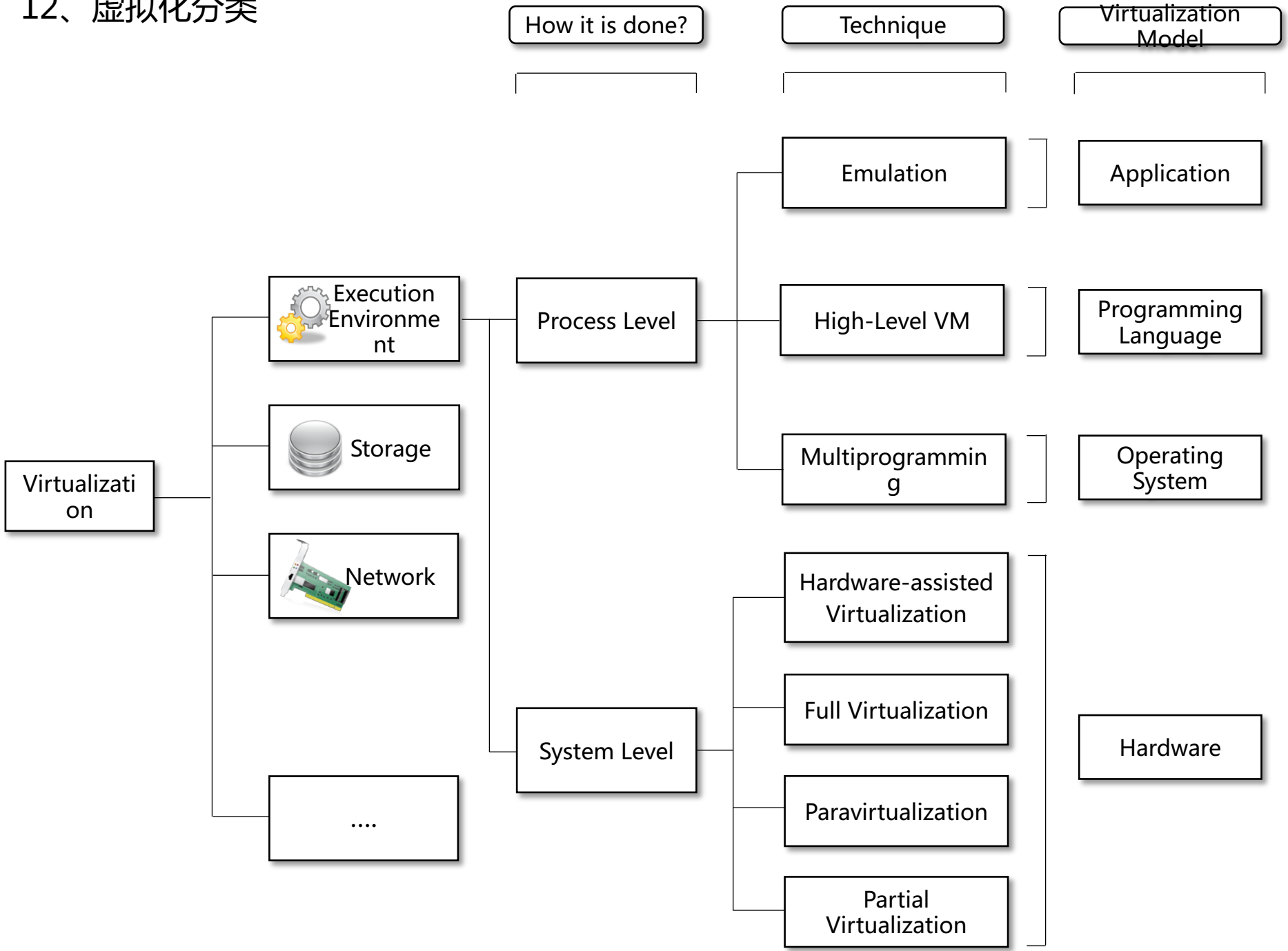
- 实现更广泛的功能特性，包括共享、聚合仿真和隔离。

- 共享：在同一主机创建多个独立的计算环境，开发没有被充分利用的客户机资源。
- 集成：一组独立的主机可以连在一起，作为一个单一的虚拟主机呈现给客户。
- 仿真：客户机程序在由虚拟化层控制的环境中执行，虚拟化层本身是一个程序。通过该程序可以控制和调整提供给客户机的环境。
- 隔离：虚拟化技术为客户机提供了完全独立的可执行环境，无论是操作系统、应用程序还是其他。客户机程序通过与抽象层交互来执行任务，该抽象层支持对底层资源的访问。
- 性能调整：通过调整虚拟环境资源的属性，很容易控制客户机的性能。
- 轻松捕获客户机程序的状态，并维持和重新执行。

（3）可移植

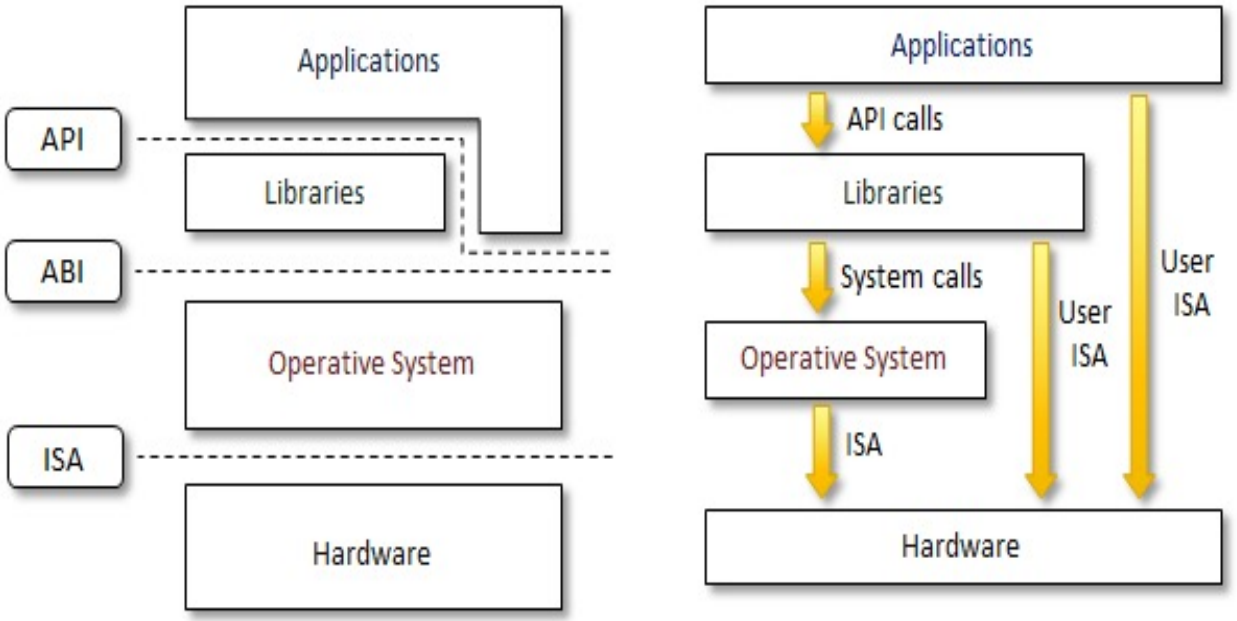
- 根据不同类型的虚拟化，可移植性的概念有不同的应用方式。
- 在硬件虚拟化方式下，客户机被转成虚拟机镜像文件，并在不同虚拟机上安全地迁移和执行。
- 在编程级虚拟化的情况下， 由JVM或.NET运行时实现， 二进制代码表示的应用程序组件(jar包或组件) ，不需要重新编译就可以在任何虚拟机上执行。

12、虚拟化分类


How it is done?
Technique
Virtualization Model
Virtualization
Execution Environment
Storage
Network
....
Process Level
System Level
Emulation
High-Level VM
Multiprogramming
Hardware-assisted Virtualization
Full Virtualization
Paravirtualization
Partial Virtualization
Application
Programming Language
Operating System
Hardware

13、执行虚拟化

（1）机器参考模型



- 底层硬件模型以指令集架构(ISA)表示，ISA定义了处理器、寄存器、存储器、中断管理的指令集。
- 应用程序二进制接口(ABI) 将操作系统层，与由操作系统管理的应用程序和库分隔开来。
- 抽象模型的最高层是应用编程接口(API) ，API将应用程序、库，和底层操作系统连接起来。

13、执行虚拟化

（1）机器参考模型

- 机器参考模型提供了实现管理和访问共享资源的最精简安全模型的方法。
- 硬件指令集被分为不同的安全等级，定义谁可以对其进行操作。
- 非特权指令不能改变共享资源的值或状态，用于不受其他任务干扰的操作。
- 特权指令是特定的限制条件下的执行，用于访问共享资源的状态和值的敏感操作，显示(行为敏感)、修改权限状态(控制教感)。
- 实现权限等级的基于环的安全体系结构

（2）硬件虚拟化

- 硬件级虚拟化是一种计算机提供抽象执行环境，供客户机操作系统运行。
- 客户机指操作系统，主机指物理存在的计算机硬件，虚拟机指模拟的计算机，虚拟机管理程序指对虚拟机的监控管理，通常是一个程序，或者是软件和硬件的集成，可以执行底层物理硬件的抽象。

（3）虚拟机管理程序有两种类型：I型和II型。

- 1型的虚拟机管理程序直接在物理机器上运行。
- II型的虚拟机管理程序依赖主机操作系统提供虚拟化服务。

1、硬件虚拟化技术

（1）硬件辅助虚拟化

- 指硬件提供架构支持、帮助创建虚拟机管理器，允许客户机操作系统独立运行。
- IBM System/370、Intel VT 。

（2）完全虚拟化

- 指运行程序的能力，操作系统无需修改，具有完全的物理机特性。
- VMM必须模拟整个底层硬件资源。
- 增强安全性，易于模拟不同系统结构，同一平台上的不同系统可以共存。
- 带来性能和技术实现问题。

（3）半虚拟化

- 不透明的虚拟化解决方案，实现简化VMM。
- 为虚拟机提供软件接口，操作系统需要进行修改。
- 对性能要求较高的操作直接运行于主机上，从而减少托管运行的性能损失。
- VMWare、TRANGO。

（4）部分虚拟化技术

- 对部分底层硬件环境进行模拟，不支持独立的客户机操作系统。
- 支持多个应用程序透明地运行，不具有完全虚拟化方式的所有特性。
- **分时系统**中实现地址空间虚拟化。

2、操作系统虚拟化

- 为同时被托管的应用程序，创建不同的单独执行环境，没有VMM或虚拟机管理程序。
- 在单一操作系统中，操作系统内核支持多个独立的用户空间实例。内核负责多个用户空间实例间，共享系统资源，防止相互影响。
- 用户空间实例一般是完全独立的文件系统视图，包含独立的IP地址、软件配置、设备访问方式。
- FreeBSD Jails

3、编程语言虚拟化

- 实现应用程序的部署和管理执行，以及跨不同平台和操作系统的移植。
- 包含运行二进制代码程序的虚拟机。编程级虚拟机也称为进程虚拟机，能跨不同的平台提供统一执行环境。
- 被编译的二进制代码程序可在任何操作系统和平台上执行。如Java和.NET。

4、应用级虚拟化

- 原有环境不支持应用所需特性时，应用虚拟化技术为此应用提供虚拟的运行环境。
- 一般情况下，涉及局部文件系统、库和操作系统部件模拟。该模拟是负责执行应用的程序或操作系统组件层。采用策略：
  - 解释：每一条指令都由模拟器解释成能执行的ISA指令，导致性能下降。
  - 翻译：每一条源指令转换为具有同等功能的本地指令。许多指令翻译后被缓存并可以重新使用。

5、存储虚拟化

- 将硬件的物理结构表示为逻辑形式，使用逻辑路径来标识。
- 利用大量的存储设备，在单一的逻辑文件系统下管理和描述这些存储设备。
- 常见的存储虚拟化技术有：存储区域网络(SAN)

6、网络虚拟化

- 将硬件设备和特定的软件结合，以创建和管理虚拟网络。
- 将不同物理网络集成为一个逻辑网络(外部网络虚拟化)，或让操作系统分区具有类似于网络的功能(内部网络虚拟化) 。
- 外部网络虚拟化通常是一个虚拟局域网(VLAN) 。内部网络虚拟化通常与硬件、操作系统级虚拟化一起应用，为客户机提供虚拟的通信网络接口。

7、桌面虚拟化

- 将个人电脑的桌面环境抽象化，以便采用客户端/服务器的方式来访问。
- 桌面环境是被存储在远程服务器或数据中心，通过网络连接来访问。
- Windows Remote Services、Sun Virtual Desktop Infrastructure(VDD)。

8、应用服务虚拟化

- 通过使用负载均衡策略，提供高可用性的基础设施
- 将多台提供相同服务的应用服务器抽象成一台虚拟应用服务器。
- 提供更好的服务质量，而不仅是模拟一个不同的环境。

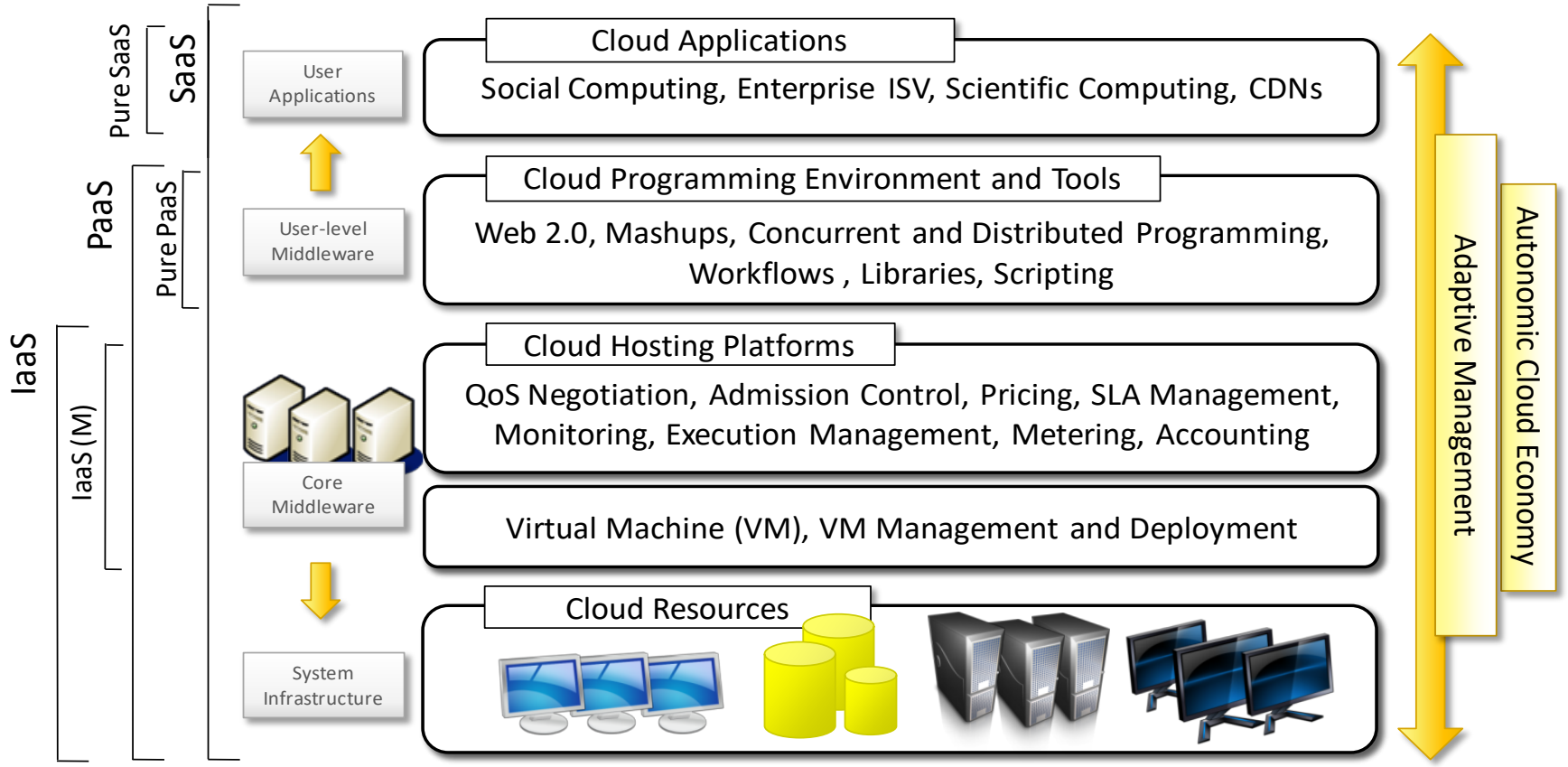
9、虚拟化的优点

- 虚拟化在云计算中扮演着重要的角色，虚拟化能够支持可定制特性、安全性、独立性和可管理性，提供可配置的计算环境和存储。
- 在技术支持创建虚拟执行环境的情况下，利用这两个特点可以构建安全可控的计算环境。虚拟执行环境可以配置为沙箱，从而防止任何非法操作进人虚拟主机。此外资源在不同客户机之间的分配被简化为由程序控制的虚拟主机来完成。
- 可移植性是虚拟化的另一个优点，特别是对于执行虚拟化技术。可移植性和自包含性也有助于减少维护成本，因为预期的主机数量比虚拟机实例的数量少。
- 通过虚拟化技术能够更有效地利用资源。多个系统可以安全地共存、共享底层主机的资源，而且不会相互干扰。

10、虚拟化的缺点

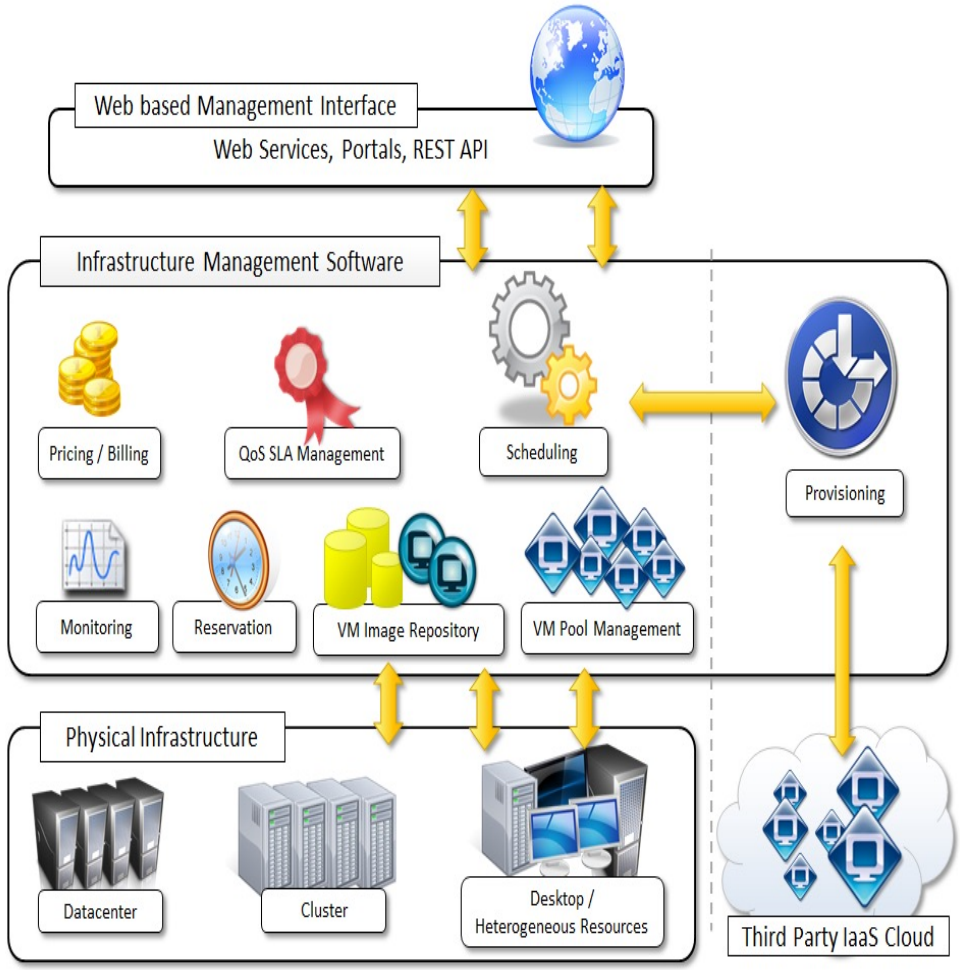
- 性能问题是主要问题。由于虚拟化在客户机和主机之间增加了抽象层，增加客户任务的操作延迟。
- 低效和用户体验下降：虚拟化有时会导致主机的低效使用。
- 安全漏洞和威胁：恶意网络钓鱼(phishing) 。

11、云计算机构



12、IaaS

- 模型划分为三个层次：物理基础设施、基础设施管理软件和用户界面。
- 用户界面可以访问基础设施管理软件提供的服务，一般基于Web 2.0技术：Web服务、RESTful APIs和mash-ups。
- 核心功能在基础设施管理软件层实现的。虚拟机的管理是该层执行的最重要功能。



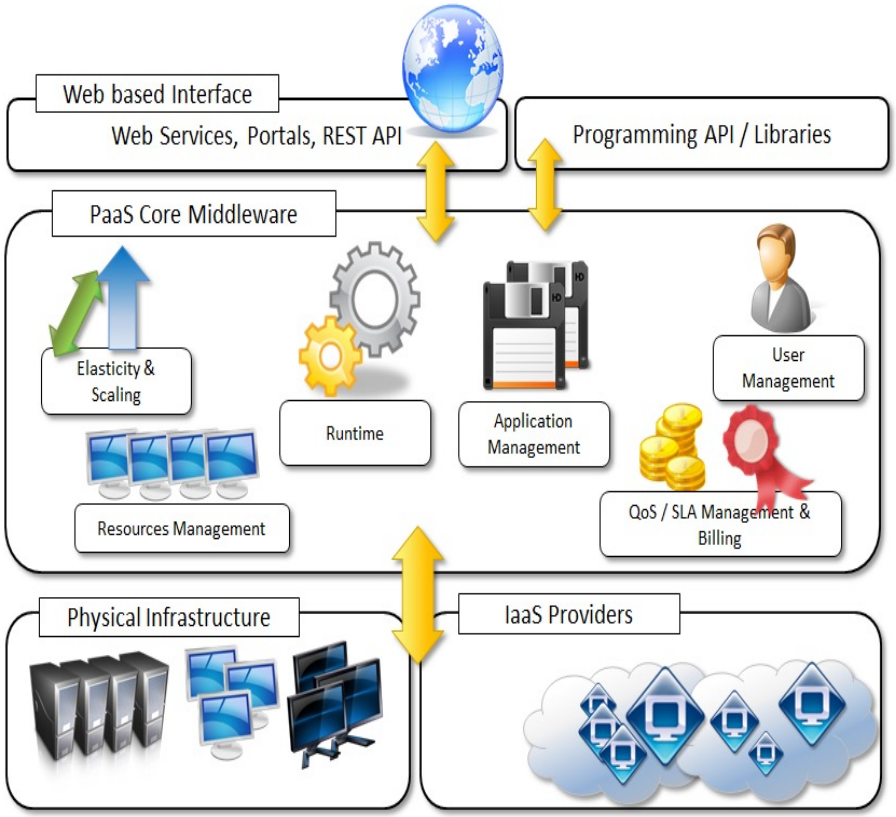
12、IaaS

调度程序负责分配虚拟机实例的执行任务。完成各种任务：

- 定价和计费组件：负责记录每个虚拟机实例的执行成本，维护用户收取费用的数据。
- 监控组件：跟踪每个虚拟机实例的执行情况，维护报告和分析系统性能的数据。
- 预约组件：存储所有已执行或将要执行的虚拟机实例的信息。
- 资源库组件：提供虚拟机镜像的目录，用户使用镜像创建虚拟实例。
- 虚拟机池管理器组件：负责跟踪所有的活动实例。

13、PaaS

- 提供一个开发和部署平台，用于在云计算中运行应用程序；构建应用程序的中间件。
- 应用程序管理是中间件的核心功能，为应用程序提供运行环境。
- 自动将应用程序部署到基础设施， 配置应用程序组件，提供和配置支撑技术。如负载均衡、数据库、根据用户设置策略、调整管理系统。
- 从用户角度，核心中间件提供接口，允许用户在云计算上编程和部署应用。



13、PaaS

PaaS解决方案基本特征：

- 运行时框架：运行时框架代表PaaS模型的“软件栈”。根据用户和供应商设置的策略，执行终端用户代码。
- 抽象：提供更高层次的抽象，关注云计算必须支持的应用程序。
- 自动化：在基础设施上，自动部署应用程序， 并通过按需配置更多资源来扩展应用。
- 云服务：为开发人员和架构师提供服务和API， 简化创建和交付，具有可扩展和高可用的云应用。

14、SaaS

- 一种基于Web的软件交付模式， 提供通过互联网访问应用程序的服务。
- 适合于多用户的应用程序， 不需要进一步定制，就能满足特定需求。 “一对多”的软件交付模式。
- 应用程序或服务通过网络，在一个集中的数据中心被部署、访问并产生使用费用。

应用服务供应商(ASP) 的不同解决方案， 在广泛意义上理解软件应用，ASP具有SaaS的核心特征：

- 销售给用户的产品，是应用程序的访问权。
- 应用程序被集中管理。
- 提供的服务是一对多的。
- 提供的服务是基于承诺契约的综合方案。

1、根据管理域划分，有四种不同类型的云：

- 公共云：开放给广大公众的云。
- 私有云：在一个机构内构建云服务，通常机构成员或子成员可以访问。
- 混合或异构云：私有云资源或服务，经扩展后在公共云中部署。
- 社区云：多管理域，满足特定行业的需求而设计。

2、公有云

- 一个分布式系统，由多个数据中心构成，云服务在数据中心的基础实现。
- 任何客户可以方便地登录云服务供应商，输人详细的账户和计费信息，使用云服务供应商所提供的服务。
- 公共云的基本特征是多租户。任何客户都想拥有一个隔离的虚拟计算环境。

- 公共云提供任何类型的服务：基础设施服务、平台服务或应用服务。如亚马逊EC2提供基础设施即服务(IaaS) 的公共云， 谷歌App Engine提供应用开发平台即服务(PaaS)的公共云、SalesForce.com提供软件即服务(SaaS) 的公共云。

3、私有云

- 依赖于私有基础设施，并向内部用户动态提供计算资源。
- 从系统架构的角度，在更多异构硬件上实现。
- 服务类型：物理层由基础设施管理软件(IaaS) 或者PaaS解决方案实现。

4、混合云

- 利用私有云和公共云的优势，现有IT基础设施、保存敏感信息、按需自动扩展和缩减资源、按需增加配置外部资源，并在不需要时释放资源。
- 一个异构分布式系统，集成多个公共云的附加服务或资源，称为异构云。
- 动态分配是混合云环境的基本组件，利用外部资源，满足超负荷需求，混合云解决了可扩展性问题。

5、社区云

- 整合不同的云服务，满足一个行业、一个社区或一个业务部门的特定需求，创建的分布式系统。
- 国家标准和技术研究院(NIST) 的描述：基础设施由几个组织共享，支持关注共同问题(如任务、安全要求、政策和法规)的特定社区。
- 从架构的角度，在多个管理域上实施。政府机构、私有企业，科研机构，公共的虚拟基础设施供应商，利用他们的资源，构建云基础设施。

6、云计算的优点：

- 减少IT基础设施的投人成本。
- 固定资产相关的折旧成本。
- 以订阅方式替代软件许可
- 降低了IT资源的维护和管理成本。
- 小型企业创业初期，采用云计算模式：IT基础设施、软件开发、客户关系管理(CRM) 和企业资源规划(ERP)

7、云计算在工业界和学术界存在许多挑战：

- 云计算定义：按需自助服务、广泛的网络访问、资源池、快速灵活服务、计量服务； 服务分为SaaS， PaaS和IaaS； 部署模型分为公共云、私有云、社区云和混合云。
- 云计算互操作性和标准：建立不同服务供应商解决方案间的互操作性标准，至关重要。
- 可扩展性和容错性：设计高度可扩展的和容错的系统。
- 安全、可信和隐私
- 组织方面：接受云计算模式，业务流程和组织结构

8、aneka框架

- 开发云应用的软件平台， 利用分布式系统资源，构成独特的运行云应用的虚拟域。
- 按照云计算的服务模型：一个纯平台即服务的云计算解决方案。
- 部署在异构资源平台上。如局域网中的计算机、多核服务器、数据中心、虚拟化的云设备等。
- 提供管理和扩展分布式应用的中间件， 以及开发分布式云应用的编程接口。

Application Development & Management

SDK: APIs & Tools

Management: Tools, Interfaces and APIs

Middleware - Container

Execution Services

Independent Bag of Tasks

Distributed Threads

MapReduce

other models…

Foundation Services

Billing & Reporting

Resource Reservation

Storage

Licensing & Accounting

Fabric Services

Dynamic Resource Provisioning

Hardware Profiling

Persistence and Security

PAL – Platform Abstraction Layer

Infrastruscture

Enterprise Desktop Grid

Data Centers

Clusters

Public Cloud

9、aneka容器结构

- 安装在每个节点上， 构成云中间件的基本单元，管理物理和虚拟的资源池。
- 云平台由一系列互联的容器构成。
- 提供三种不同服务形式：构造服务、基础服务和执行服务。

10、aneka平台基础：平台抽象层

- 检测主机环境的实现与交互， 支持上层容器的运行。
- PAL特征：
  - ·提供统一且与平台无关的执行接口，访问主机平台。
  - ·对主机平台的扩展和附加属性，提供统一访问途径。
  - ·提供统一且与平台无关的途径，访问远端节点。
  - ·提供统一且与平台无关的管理接口。

- 向上层提供的信息包括：
  - ·内核数量、频率和CPU用途。
  - ·内存的大小和使用情况。
  - ·全局层面的硬盘可用空间。
  - ·网络地址和节点上的设备。

11、构造服务

（1）概要分析和监控

- 由心跳、监控和报告服务构成。
- **心跳服务**周期性收集节点动态运行信息，并将信息发布给云中的成员服务。信息由索引节点收集，为资源预留和调度服务提供必要依据，实现异构基础设备应用。
- **报告和监控服务**：负责监控数据的存储，提供给其他服务接口或者分析应用。任何想要提供监控信息的服务，借助本地的监控服务接口，而不必了解整个基础设施的细节。
- **收集信息：**

  ·成员目录记录节点运行信息。

  ·执行服务在任务执行过程中，选取时段进行监控。

  ·调度服务记录任务的状态信息。

  ·存储服务监控和提供数据迁移中的信息，例如上传和下载次数、文件名和大小。

  ·资源分配服务记录虚拟节点的分配和生命周期信息

（2）资源管理

- Aneka提供服务接口，负责资源管理， 包括索引服务(即成员日录)、预留服务和资源分配服务。
- 成员目录是资源管理的基本组件，记录所有节点(无论连接与否) 的基本信息，允许通过节点名等属性字段搜索服务。
- 动态资源分配：允许从IaaS服务商租用的虚拟资源整合到云平台，该服务改变了云平台中的拓扑结构，允许根据不同需求扩展部署。

12、基础服务

- 用于对构建在基础设施上的分布式系统，进行逻辑管理，并提供分布式应用的执行。所有编程模型都可集成到服务，同时提供高级且全面的应用管理。

（1）存储管理

- Aneka提供两组服务模式：集中式文件存储，主要适用于计算密集型应用，分布式文件系统，适合于数据密集型应用。
- 集中式存储通过Aneka的存储管理服务实现， 该服务构成Aneka的数据分级服务。存储服务提供基本的文件传输功能，以及面向终端用户或其他系统组件的抽象接口，接口可由安装的服务动态配置。目前默认的协议配置是FTP。
- 数据密集型的应用采用分布式文件系统进行存储， 其参考模型是谷歌文件系统(GFS) ，其有基于商用硬件的高度可扩展的基础设施。该模型由GFS提出并能够对具有以下特征的应用提供优化支持：
  ·GB级别以上的文件。
  ·修改文件时需要增加文件内容，而不需要对原有文件内容进行修改。
  ·以大量读取访问和少量随机访问为主的负载类型。
  ·持续稳定的带宽比低响应延迟更为重要。

（2）记账、结算和资源定价

- 记账服务记录应用的状态，收集分布式节点的使用情况信息，对资源管理的优化至关重要。所收集的信息与设备使用和应用执行有关，信息构成了用户计费的基础。
- 结算是记账服务的另一个重要功能。为每个用户提供资源使用及其相关费用的详细信息。每项资源可以根据服务接口的类型或者节点上安装的软件来定价。结算模型提供了对各项应用的综合结算，对于具体用户提供每项任务的结算信息。

（3）资源预留

Aneka提供扩展API，支持高级服务，支持三种不同的实现：

·基本预留：代表节点时隙预留的基本方法，执行双备份协议，提供预留资源的备份选项，以避免初始的预留请求不能被满足。

·秤量(计价)预留：根据节点的硬件资源进行节点定价，并提供预留。

·中继预留：实现简单， 使资源代理商预留节点，控制节点资源。

13、应用服务

- 对不同的分布式应用编程模型，应用服务提供相应接口。
- 所支持模型中有两类共性的类型：调度服务和执行服务。

（1）调度

负责规划分布式应用的执行，将应用所包含的任务分配给节点。调度服务同时也是基础服务和构造服务的结合点。调度服务组件如下：

·节点映射任务。

·失败任务的重新调度。

·任务状态监控。

·应用状态监控。

（2）执行

控制单个任务的执行，负责设定执行任务的运行环境，调度服务实例运行时，每个编程模型都有自身需求，但有一些共性操作：

·从调度器获取任务后的分发。
·取回所需的输人文件。
·任务的沙盒运行。
·任务执行完毕后，输出文件的提交。
·执行故障管理(例如抓取有效的上下文信息，进行故障鉴别)。
·性能监控。
·任务打包，并回传给调度器。

（3）应用服务构成对编程模型的运行支持，目前所支持的编程模型

·任务模型：包含独立多任务划分和计算任务。应用被划分为相互独立的任务的集合，任务的执行可以按任意顺序进行
·线程模型：提供典型多线程编程模型进行扩展，使用线程抽象模型装载远端执行程序模块
·MapReduce模型：MapReduce在Aneka上的实现
·参数化模型：任务模型应用的一类实例，将多组参数集合输人任务实例，每组参数代表选定域中一个具体的点。

1、吞吐量计算

- 着眼于以事务的形式传送大量的计算。
- 使用多进程和多线程实现。

2、多核系统架构

- 由一个以共享内存的多处理器核为主的单处理器构成。
- 每个核一般有自己的缓存L1，缓存L2为以共享总线方式连接，所有核共有。
- 双核和四核结构现今十分流行，已成为商用计算机的标准硬件结构。

3、线程

- 标识进程中的单个控制流，是指令的一个逻辑序列。指令的逻辑序列是按照程序员设计的执行指令序列。
- 操作系统将线程识别为运行代码的最小程序块。开发人员使用的显式线程，操作系统执行的任何指令序列，都在一个线程中。
- 每个进程至少包含一个或多个具有不同生存周期的线程组成。所有线程共享内存空间和执行内容。

4、多进程和多线程

- 在一个多任务环境中，操作系统为每个进程分配不同的时隙并交错执行。暂停一个进程的执行， 在寄存器中保存所有信息， 并把它替换为与另一个进程相关的信息，这一操作浪费时间。
- 多线程的使用，使内容切换产生的延时最小，更易于实现多任务的执行。一个线程的执行的状态比描述一个进程要简单。

5、线程API

（1）POSIX

- 创建带属性的线程、线程的终止、等待线程结束(连接操作)。
- 除了线程的逻辑结构之外，还将引进其他抽象概念，如信号量、条件、读写锁等，以支持线程之间的同步。
- POSIX提出的模型已成为其他实现的参考， 为开发者提供具有类似功能接口。
- 从编程角度，理解线程：

  ·一个线程标识指令的一个逻辑序列

  ·一个线程映射为一段执行的指令序列

  一个线程可以创建、终止和连接。

  ·一个线程根据当前情况确定的状态，无论该线程是在执行、中断、终止、等待，还是输人/输出等。

  ·线程正在经历的状态序列，部分由操作系统调度器决定，部分由应用程序开发人员决定。

  ·线程共享进程的内存，由于同时执行，需要同步结构。

  ·提供不同同步抽象，以解决不同的同步问题

（2） Java和C#

- 通过面向对象的方法，为多线程编程提供丰富的函数。
- Java和.NET在虚拟机上执行代码， 所以库提供的应用程序接口，涉及托管或逻辑线程。
- 由这些语言开发的程序运行环境，映射到物理线程上(即由底层操作系统支持的抽象)。从编程的角度，托管的线程和物理线程呈现相同的功能
- 线程操作：开始， 结束，暂停，恢复，强制终止，休眠，连接，中断。
- 两个框架对线程间的同步，提供不同的支持。

6、线程并行计算

- 对于开发并行应用，分解用于分析问题是否可以分成并行执行的任务。如果任务分解为独立单元，通过线程实现并发执行。
- 主要的分解/分割技术包括：域分解和功能分解。

7、域分解

- 用来识别，功能重复、但独立的数据计算模式。
- 当计算操作相同，数据不同，且可以按任何顺序执行时，这种问题称为高度并行。不同线程没有同步和共享数据，线程间的协调和通信很少。
- 常见的组织结构：主从模型
  - ·系统分为两个代码段：一个代码段包含分解逻辑和协调逻辑；另一个代码段包含要执行的重复计算。
  - ·主线程执行第一个代码段，创建足够多的从属线程来执行重复计算。
  - ·每个从属线程的结果收集和最终结果的合成，由主线程执行。

8、高度并行问题包括：

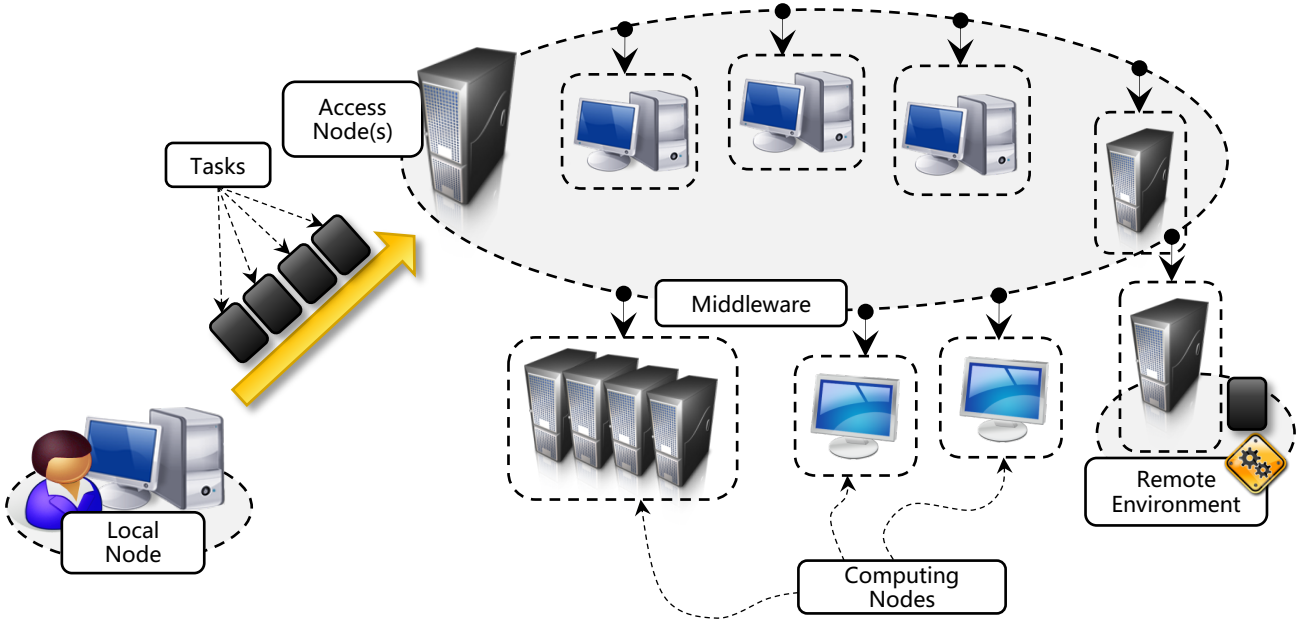
- 二维(或更高维)数据集的几何变换。
- 一个域的独立和重复计算，如曼德勃罗集和蒙特卡罗计算。

基于强有力的假设：每个迭代的分解方法中，隔离作业是一个独立单位。

9、任务

- 一个任务标识一个或多个操作，不同的操作产生不同的输出，并且这些输出可以被分离为单个逻辑单元。
- 在实践中，一个任务分离为不同的代码单元，或一个程序，可以在一个远程的运行环境中执行。

10、任务计算场景



- 中间件是一个软件层，能够协调数据中心，或地理上分散的网络计算机的多种资源。
- 用户提交任务集合到中间件的接人点，中间件负责调度和监控任务的执行。
- 每个计算资源提供了一个适当的运行环境， 不同实现之间可能会有所不同，如简单的shell、沙箱环境或虚拟机 。
- 无论是Web还是编程语言接口， 通常都采用中间件的API提交任务、监控任务状态和收集结果。
- 中间件的一组常用操作，包括基于任务的应用程序创建和执行：
  - ·协调和调度任务在一组远程节点上执行。
  - ·移动程序到远程节点，并管理它们的依赖关系。
  - ·创建在远程节点上执行任务的环境。
  - ·监控每个任务的执行，并通知用户其执行状态。
  - ·访问由任务产生的输出。

11、任务特性

- 一个任务可以由不同的元素表示：
  ·几个应用程序共同执行组成的一个shell脚本。
  ·单个程序。
  ·在一个特定运行环境的上下文中，执行的代码单元(一个Java/C++/NET类)
- 任务通常的特点是输入文件、可执行代码(程序、shell脚本等) 和输出文件。在许多情况下，任务执行的常见运行环境，是操作系统或等效的沙箱环境。除了可以传送到该节点的库依赖之外，任务也需要远程执行节点上的特定软件设施。
- 一些分布式应用程序有额外的限制。例如，在编程级别上，通过类继承或接口，实现描述任务抽象的分布式计算框架，需要额外的限制(即符合继承规则)，以及一组更丰富的，可供开发人员利用的功能。根据应用程序的具体模型，任务之间可能有依赖关系。

12、计算类别

（1）高性能计算（HPC）

- 利用分布式计算设施，来解决大量计算能力的问题。在短时间内，需要处理的一系列计算密集型任务构成。
- 常见是并行和紧密耦合的任务， 需要网络低延迟，尽量减少数据交换时间。
- 评估HPC系统的指标是FLOPS， 表示计算系统每秒可以执行浮点运算的次数，（TFLOPS、PFLOPS）。

（2）高吞吐量计算（HTC）

- 利用分布式计算设施，实现在长时间内要求大量计算能力的应用。
- 必须具有足够的健壮性， 并且能在一段较长的时间上进行可靠操作。
- 由大量任务组成，这些任务的执行可能会持续相当长的时间(几周或数月)。
- 典型例子如科学模拟和统计分析。在分布式资源中，常见的是调度独立任务，任务间不需要通信。HTC系统以每月完成的作业数来衡量性能。

（3）多任务计算（ MTC ）

- 消除HPC和HTC之间的差距。
- MTC类似于HTC， 但着重于在短时间内，使用许多计算资源来完成多个计算任务。
- MTC表示通过文件系统操作，连接多个不同活动的高性能计算。
- MTC的特征是任务的异质性， 可能会有大量不同的性质：任务的大小， 单处理器或多处理器， 计算密集型或数据密集型， 静态或动态， 同构或异构等。
- MTC应用程序是松耦合的， 一般是通信密集型， 但不能通过HPC的消息传递接口，需要注意许多计算是多样的但不能高度并行。
- 考虑到MTC通常由大量的任务构成，所以任何具有大量计算元件的分布式设施，就能支持MTC。这些设施包括超级计算机、大型集群和新兴的云基础设施。

13、任务计算框架

（1）Condor

- 使用最广泛、存在时间最长的中间件，用于管理集群、闲置工作站和集群集合。
- Condor-G是Condor的一个版本，支持由Globus管理的网格计算资源集成。
- 支持批处理队列系统的常见功能， 以及作业的检查点和过载节点的管理功能。
- 提供了一个功能强大的资源匹配机制，仅在具有适当运行时环境的资源上调度作业。处理各种资源上的串行和并行作业。
- 被工业界、政府和学术界的数百个组织，用来管理从几个到上千个工作站的基础设施。

（2）Globus Toolkit

- 提供了一套全面的工具，用于跨企业、机构和地理边界共享计算能力、数据库和其他服务，无需影响本地自治。
- 提供软件服务、库，用于资源监测、发现、管理、安全性和文件管理。
- 解决网格计算的核心问题：管理由跨不同组织的异构资源组成的分布式环境，同时考虑安全性和互操作性方面的隐含条件。为了有效支持网格计算，定义了一组用于互操作的接口和协议，使不同的系统能够相互集成，并在其边界之外公开资源。

（3）Sun Grid Engine(SGE)

- 用于工作负载和分布式资源管理的中间件，支持集群上作业的执行。
- 集成了其他功能，能够管理异构资源，并构成了网格计算的中间件。
- 支持并行、串行、交互式和参数化作业的执行，具有高级调度功能，例如基于预算和组的调度、有期限地调度应用程序、自定义策略和提前预订。

13、任务计算框架

（4）伯克利网络计算的开放式架构(BOINC)

- 志愿计算和网格计算的框架。允许将桌面计算机转变为志愿计算节点，当这些计算机处于非活动状态时，利用这些节点来运行作业。
- 两个组件组成：服务器和客户端。前者是中心节点， 用于跟踪所有可用的资源和作业调度，后者是部署在桌面计算机上，为作业提交创建执行环境。
- 目前，有几个项目正在BOINC基础设施上运行，包括医学、天文学和密码学。

（5）Nimrod/G

- 在全球的计算网格内，自动建模和执行参数化应用程序的工具。
- 提供一种简单的声明式参数化建模语言，用于表示参数化实验。领域专家可以很容易地创建参数化的实验计划， 并使用Nimrod/G系统在分布式资源上部署作业，并执行。
- 已广泛用于各种应用程序，从量子化学到政策和环境。
- 此外，使用基于经济原则的新型资源管理和调度算法。支持在分布式网格资源上，对应用程序进行期限和预算约束的调度，以最小化执行成本，同时及时交付结果。

14、基于任务的应用模型

（1）高度并行应用

- 分布式应用中最简单，最直观的一类。由一系列相互独立，且可以按任何次序执行的任务构成。各任务的类型可能相同，也可能不同，它们之间不需要通信。调度相对简单，主要关注任务与可用资源间的优化映射。
- 支持高度并行应用的框架包括：Globus工其包、BOINC和Aneka。
- 应用包括图像、视頻渲染、演进优化、模型预测。

（2）参数化应用

- 高度并行应用的一个分支，所划分的任务本质相同只是执行参数不同。
- 参数化任务由模板任务和参数集所定义。模板任务定义了远程节点上所需要执行的操作。参数集是由将模板任务配置为计算实例的变量构成的集合。参数以及它们各自的可行域构成了应用多维域，多维域中的每个点代表了一个任务实例。
- 支持高度并行应用的分布式计算框架，也支持参数化应用的执行。将要执行的任务通过迭代所有参数的可行域。框架本身或者分布式计算的中间件的工具完成任务迭代。
- 参数化应用类型众多，大部分来自科学计算领域，如演进优化算法、天气预报模型、流体力学应用、蒙特卡罗方法等
- 参数化应用提供原生支持的框架，通常提供了一系列文件操作命令：
  - ·执行：在远程节点上执行一段代码。
  - ·复制：从远程节点导人或导出文件。
  - ·替换：用参数在文件中的占位符替换参数值。
  - ·删除：删除文件。

14、基于任务的应用模型

（3）消息传递接口(MPI)应用

- 通过交换消息进行通信的并行程序的规范。联合了几个支持分布式计算(支持分布式存储器共享和消息传递)的平台，致力于创造一种共同规范。
- 目前， MPI事实上已经成为了开发可移植的、高效消息传递的高性能计算应用。接口规范已经被C、C++和Fortran语言定义和实现。
- MPI规范：
  - ·管理MPI程序所运行的分布式环境。
  - ·向点对点通信提供服务。
  - ·向群组通信提供服务。
  - ·支持数据结构定义和内存分配。
  - ·为块调用的同步提供基本支持。

14、基于任务的应用模型

（4）工作流应用

- 由一系列具有依赖性的任务构成。这些依赖(主要是数据依赖，如一项任务的输出构成另一项任务的前提)决定了应用调度的方式和分布。
- 在商用开发领域，工作流定义为商业过程的自动化，在整体或局部上，按照一系列流程规范，将文件、信息或任务从一个参与者(资源、人或机器)传递给另一个。

- 科学工作流:
  - 流程由一个执行应用所决定，参与者间传递的元素主要是任务和数据，参与者是计算和存储节点。
  - 流程规范由工作流定义框架所定义，以便指导任务调度。涉及数据管理、分析、仿真和支持工作流执行的中间件。
  - 通常由有向无环图(DAG) 表示， 描述任务或操作间的依赖关系。节点表示工作流应用中需要执行的任务，连接节点的箭头定义了任务间的依赖关系以及连接任务的数据路径。

（5）工作流技术

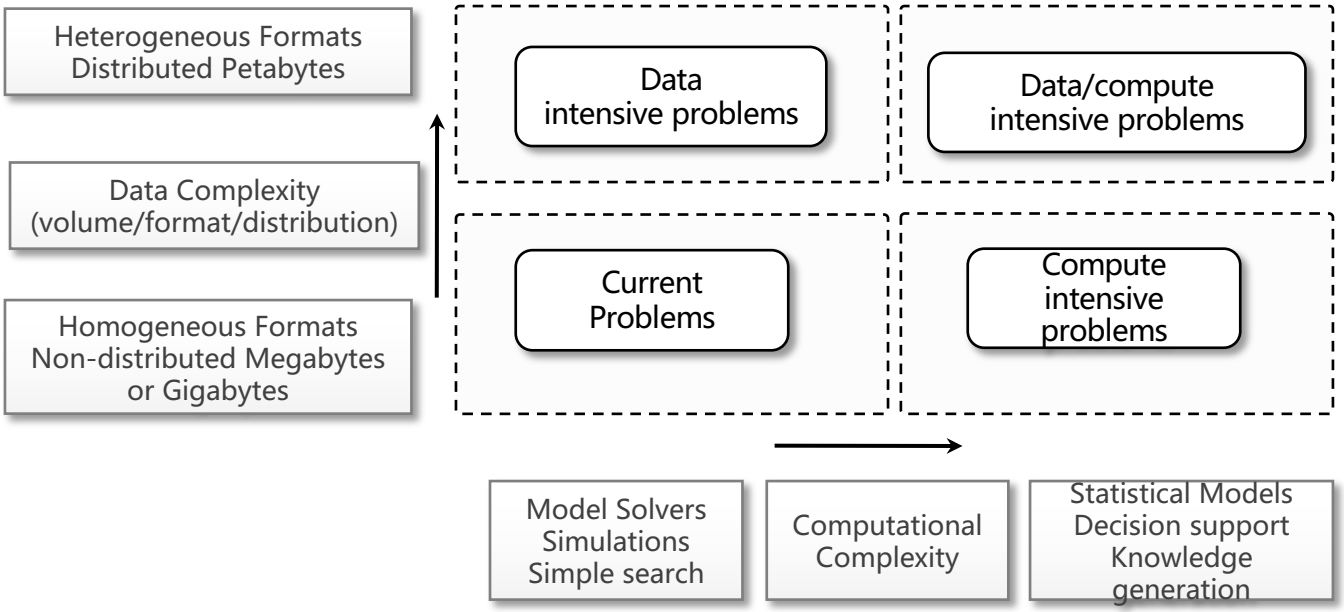
- 工作流引擎是一个客户端组件，可与资源或者执行工作流的中间件组件进行交互。
- 框架能够处理工作流实例的调度器，实现对工作流应用的支持。
- Kepler:
  - 开源科学工作流引擎，系统在Ptolemy II系统上构造。
  - 提供一种基于actor概念的构造环境， actor是独立且可重用的计算模块， 可加载网络服务、数据库命令等。actor通过端口通信从输人端口获取数据，并将处理结果和数据写人输出端口。
  - 利用执行工作流的协作逻辑来分离数据流。对同一个工作流，提供不同的模式（同步和异步），工作流规范由XML语言所描述。
- DAG Man(有向无环图管理)
  - 扩展Condor调度器，处理任务内部的关联性。DAG Man作为Condor系统的元调度器可以将任务以合适的次序提交给调度器。
  - 输人文件是简单文本文件， 包含任务提交文件指针和任务的依赖性描述。
- WfMS(Cloud工作流管理系统)
  - 部署在分布式的云或网格系统上，用于处理大型应用工作流的中间件平台。
  - 通过Web的门户提供相应软件工具，帮助终端用户组织、调度、执行、监控工作流。门户提供上传工作流，使用图形编辑器定义新工作流的方法。
  - 使用Grid bus Broker执行工作流， Grid bus Broker是网格或云环境下的资源代理器， 在异构分布式计算环境下按照服务等级属性执行应用。
  - 使用一种私有XML语言作为其工作流规范。
- OffSpring
  - 能够与任何支持简单任务划分应用的分布式中间件集成。允许以插件的形式定义工作流， 不需要使用任何XML语言规范。

1、数据密集型计算

数据密集型计算涉及数百MB到PB，甚至更大规模数据的产生、处理和分析。术语数据集通常用于表示信息元素的集合，这些元素与一个或多个应用相关。数据集通常保存在库里，需要支持大量信息存储、检索以及索引的基础设施。为了便于分类和搜索，需要将元数据附加到数据集。

数据密集型计算出现在许多应用领域，如计算科学、一些IT产业。

2、数据密集型计算特性



数据密集型应用程序不仅处理大量数据，还表现出计算密集型的特性。数据密集型应用处理几个TB和PB级规模的数据集。数据集以多种格式保存，分布在不同的位置。应用程序以多步分析的管道处理数据，包括转换和融合。处理需求几乎与数据大小成线性关系，并且可轻松地并行处理。处理过程需要高效的数据管理、过滤和融合机制，以及高效的查询和分发机制。

3、历史背景

（1）早期的高速宽带网络

数据传输和流媒体的技术、协议和算法的发展，使数据密集型计算成为可能。

PSC的大型复杂科学数据集（人脑高分辨率MRI扫描）的远程可视化。

（2）计算网络

- 随着网格计算的出现，通过跨管理域的异构资源，获得大量计算能力和存储容量，数据网格作为支持数据密集型计算的基础设施。数据网格提供客户发现、转换和操作数据集服务，创建和管理数据备份。
- 大部分数据网格提供存储和数据集管理能力，作为产生大量数据的科学实验的支持。科学家利用特定的发现和信息服务，确定他们最感兴趣的数据集的位置。
- 由于网格资源异构性和不同管理域，因此需要适当安全措施和虚拟组织(VO)的使用。
- 引人了新的挑战：
  - 大量数据集：在批量传输、复制内容、管理存储资源时，需要将延迟降到最低。
  - 共享数据集：资源共享包括数据分布式收集，数据库可以用来存储和读取数据
  - 统一的命名空间：数据网格强制使用统一的逻辑名称空间来定位数据集合和资源。每个数据元素都有一个单一的逻辑名称，最终映射到不同的物理文件名，以实现复制和可访问。
  - 访问限制：数据网格中的身份验证和授权，涉及对共享数据集合的粗粒度和细粒度访问控制。

（2）计算网络

- 数据网格：数据网格是一种更加结构化和综合化的进行数据密集型计算的方法。
  - LHC网格：一个全球的网格计算环境，为与LHC实验合作提供服务。支持从几百TB到PB的大规模数据集的存储和分析。
  - 生物信息学研究网络(BIRN) ：通过数据共享和在线协作，提供数据共享基础设施、软件工具、战略和咨询服务。
  - 国际虚拟天文台联置(IVOA) ：利用网络形成天文学的科学研究环境，允许科学家发现、访问、分析和组合来自异构数据集合的实验数据。

（3）数据云和大数据

- 数据集越来越大，以至于使用现有数据库管理工具变得复杂。关系型数据库和桌面统计/可视化软件包，对于海量信息变得无效，取而代之的，是“运行在数十甚至上千的服务器上的大量并行软件”。
- 数据云计算几种方式支持数据密集型计算；
  - 提供大量的实时计算实例，用于并行处理和分析大量数据集。
  - 提供优化的存储系统，用于保存大文件数据，以及分布式数据存储架构。
  - 提供构架和可编程API， 以便处理和管理海量数据。这些API大多与特定的存储基础架构相结合，以优化系统的总体性能。

（4）数据库和数据密集型计算

- 分布式数据库是存储在计算机网络不同站点的数据集合。每个站点可能会提供一定程度的自主权，为本地应用程序的执行提供服务，也参与全球应用程序的执行。
- 分布式数据库可以通过将现有数据库的数据拆分并分散到不同的站点上，或者将多个现有数据库联合在一起来创建。这些系统非常健壮，提供分布式事务处理、分布式查询优化和高效的资源管理。
- 最关心的是可以使用关系模型表示数据集，对数据强制执行ACID属性，限制它们的扩展能力， 就如同数据云和网格一样。

4、高性能分布式文件系统和存储云

（1）Lustre

- 一个大规模并行分布式文件系统，涵盖从一个小型集群工作组到一个大型计算集群的需求。该文件系统被多个500强超级计算系统使用。
- 提供对PB（PB）存储的访问，以每秒数百GB（GB/s）的IO吞吐量服务于数千个客户端。
- 系统由包含文件系统元数据信息的元数据服务器和负责提供存储的对象存储服务器集合组成。
- 用户通过兼容POSIX的客户机访问文件系统，该客户机作为模块安装在内核中，也可以通过库安装。
- 文件系统实施了一种强健的故障切换策略和恢复机制，使服务器故障和恢复对客户端透明。

（2）GPFS

- IBM开发的高性能分布式文件系统，为RS/6000超级计算机和Linux计算集群提供支持。
- 一个多平台分布式文件系统，提供高级恢复机制。
- 基于共享磁盘概念构建，其中磁盘集合通过某种交换结构连接到文件系统节点。文件系统使此基础结构对用户透明，通过复制文件的一部分在磁盘阵列上对大型文件进行分块，以确保高可用性。通过这种基础设施，系统能够支持PB级的存储，以高吞吐量访问，并且不会丢失数据的一致性。
- 与其他实现相比，GPFS还分发整个文件系统的元数据，提供对它的透明访问，从而消除了单点故障。

（3）GFS

- 一个支持谷歌计算云中分布式应用程序执行的存储设施。建立在商用硬件和标准Linux操作系统上，具有容错的、高可用性的、分布式的文件系统。
- 不是一个分布式文件系统的通用实现， 专门针对谷歌在应用程序的分布式存储方面的需求，按照以下假设进行设计：
  - 该系统建立在经常出错的商用硬件的顶层。
  - 该系统存储适量的大文件。多GB文件是普遍的而且应该得到有效的处理，小的文件必须被支持，但没必要优化它。
  - 工作负载基本由两种读取构成：大型流读取和小型随机读取。
  - 工作负载也具有将数据追加到文件的许多大量和连续的写人
  - 带宽的高持续比低延迟更重要。

（3）GFS

- 该文件系统的架构组织成一个单一的主设备，包含整个文件系统的元数据，以及一个提供存储空间的块服务器的集合。
- 从逻辑上看，该系统由软件后台程序的集合组成，实现主服务器和块服务器。
- 文件是块的集合，块的大小可以由文件系统级别确认。块被复制在多个节点以实现容错功能。
- 客户查询主服务器，获得想要访问的文件的具体块的位置信息，客户和块服务器之间的联系就发生了。
- 应用程序通过文件系统与一个特定的接口交互，支持对文件的建立、删除、读取和写人的常规操作。也支持快照和记录追加的操作。
- 通过给予在其他任意属于该设施的节点上复制主节点信息，单主架构故障的潜在单一节点已得到解决。守护进程和广泛的日志记录有助于故障系统的恢复。

（4）S3

- 在线存储服务支持高可用性、可靠性、可扩展性、无限存储和低延迟商业价值。
- 提供组织成桶的平面存储空间， 并连接到亚马逊网络服务(AWS) 账户。每一个桶能够存储多个对象， 每个对象以一个唯一的关键词来识别。
- 对象以唯一的URL被识别并通过HTTP访问， 允许简单的读/取语义。对象通过BitTorrent协议检索。开发了一个类似于POSIX的客户机库来装载S3存储桶。
- 对象的可见性和可访问性链接到AWS帐户，桶的所有者可以决定对其他帐户或公众可见。还可定义经过身份验证的URL，在有限的时间段内提供公共访问。
- BitTorrent用于分发大量数据的P2P文件共享协议。允许用户从多个主机并行下载文件。

5、NoSQL

- 一类存储和数据库管理系统的总称，克服由关系模型带来的限制，不使用固定模式的标签，以适应更大范围的数据类型，或避免横向扩大性能和规模。
- No SQL的实现：
  - 文档存储(Apache Jackrabbit， Apache Couch DB， Simple DB，Terra store)。
  - 图表(Allegro Graph， Neo4j.Flock DB， Cerebrum)。
  - 键一值存储：磁盘上的键-值存储、RAM中的键-值缓存、键-值存储等级、最终一致性键-值存储和排序键-值存储。
  - 多值数据库(OpenQM， Rocket U2， OpenInsight)。
  - 对象数据库(ObjectStore， JADE， ZODB)。
  - 表格存储(BigTable， Hadoop HBase， Hypertable)。
  - 元组存储(Apache River)。

（1）Apache CouchDB和MongoDB

- 文档存储，提供无模式存储，主要的对象是组成一个键一值域集合的文件。
- 每个域的值可以是字符串类型、整数、浮点数、日期或数值数组。
- 数据库公开RESTful接口， 用JSON格式表示数据。
- 允许通过MapReduce编程模型查询和搜索数据， 使用Javascript，不是SQL作为数据查询和操作的基本语言， 支持大文件文档。
- 从基础设施的角度，支持数据复制和高可用性。
- CouchDB支持数据ACID属性。MongoDB支持分片， 分割不同节点的内容。

（2） Dynamo

- 分布式键-值存储，提供可扩展规模的高可用性的存储系统，实现大规模的可靠性。
- 提供读/取语义的简单化接口， 对象通过接口以唯一的标志(键)被存储和取回。
- 系统强加了一些限制。例如牺牲了ACID的数据性能。
- 系统由存储节点的集合组成， 节点组成一个为应用程序共享**键间距**的环。键间距在存储节点间分开，而且键通过环进行复制。每一个节点具有本地存储设施的访问权，设施中存储着原始对象和副本。即每个节点提供分割环间的更新、探测错误和未达到节点的能力。

（3）Bigtable

- 分布式存储系统， 用于在数千服务器上处理高达PB级规模的数据。
- 为不同工作负载类型的Google应用程序，提供存储支持：从面向吞吐量的批处理作业到对延迟敏感的据服务。
- 设计目标是广泛的适用性、扩展型、高性能和高可用性。用表格组织数据存储表格的行在支持中间件的GFS中被分割。
- 从逻辑角度，一个表格是一个以关键字索引的多维有序映射，关键字以任意长度的字符串表示。表格被组织成行和列，列可以被组织成列族，允许特定的优化，以实现更好的访问控制、存储和索引数据。
- 一个简单的数据访问模型，构成了客户应用程序的接口，应用可以通过一行中单一列的粒度水平找到数据。每一个列值以多版本存储， 这些版本可以自动标记时间。
- 除了基本数据访问之外，Bigtable API还允许更复杂的操作，例如通过Sazwall脚本语言或MapReduce API进行单行事务和高级数据操作。
- 一个子表服务器负责服务一个子表请求，子表是一个表格中一部分连续行。每一个服务器可以管理多个子表(通常是10~1000个)。
- 主服务器负责跟踪子表服务器状态和子表服务器配置的子表的状态。服务器不断地监视子表服务器以检查它们是否正常，如果发现子表服务器不可连接，配置的子表将被重新安排并最终分配给其他服务器。

（4）Apache Cassandra

- 分布式对象存储，以管理分散在许多商用服务器中的海量结构化数据。
- 最初由Facebook开发，现在是Apache Incubator项目的一部分，为Facebook、Digg和Twitter等超大型的Web应用提供存储支持。
- Cassandra被定义为第二代分布式数据库，建立在遵循完全分布式设计的Dynamo，继承了Bigtable的“列族”概念。以关键字索引的分布式多维表，对应于一个键的值是一个高度结构化的对象，构成了表中的行。将表格的行组织为列，一组列被组织成列族。
- 用于访问和操作数据的API非常简单：插人、取回和删除，插人以行级别执行，取回和删除可以在列级别操作。

（5）Hadoop HBase

- 支持Hadoop分布式编程平台的存储需求的分布式数据库。
- 通过利用商用硬件的集群，提供数十亿行和数百万列的表格的实时读/写操作。
- 整个系统由Hadoop分布式文件系统(HDFS) 支持，模仿了GFS的结构和服务。

1、数据密集型应用的编程平台

提供抽象概念，有助于在大量信息和运行时系统上描述计算，从而能够高效地管理大量数据。

2、MapReduce编程模型

通过两个简单的计算逻辑：map和reduce。

map (k1,v1) → list(k2,v2)

reduce(k2,list(v2)) → list(v2)

map函数读取一个键-值对，并产生一列不同类型的键-值对。reduce函数读取由键和列值组成的一个对，并产生一个相同类型的值的列表。map任务的输出，根据相应的键将值分组，以这种方式合并在一起， 并构成reduce任务的输入，对于每个找到的键， 此任务就是将附加值的列表减少为单个值。

3、Map Reduce应用实例

（1）grep

MapReduce在文本流中执行模式识别的grep操作（查找内容包含指定的范本样式的文件），通过广泛的文件集来运行。MapReduce提供并行和更快的执行。输人文件是一个简单的文本文件，每一次识别出给定的模式时，map函数发送一行到输出。reduce任务将map任务送出的所有行汇聚到一个单一文件。

（2）URL访问频率计数

MapReduce用于分配Web服务器日志分析的执行。map函数需要输人一个Web服务器的日志，并对于日志中记录的每一页访问发送一个键-值对<URL，1>到输出文件。reduce函数通过相应的URL汇聚所有行，由此得到输出<URL，totalcount>对。

（3）反向网络链接图

反向网络链接图跟踪所有可能导致一个给定链接的网页。输人文件是简单的由map任务扫描的HTML页面，它为网页源发现的每个链接发送<target，source>对。reduce任务将校对所有的对，如果这些对有相同目标就成为<target，list(source) >对。最后的结果给出一个或更多包含这些映射的文件。

（4）每个主机的词向量
词向量重述最重要的文字， 它们出现在一组<word， frequency>列表形式的文档中， 其中出现的单词数量是其重要性的度量。MapReduce用于提供一组文档的源和其相应的词向量之间的映射， 这个源是作为文档URL的主机部分被获得的。在这种情况下， map任务为每一个检索的文本文档创建一个<host， term-vector>对， 并且reduce任务汇聚了与从相同主机检索的文档相对应的词向量。

（5）反向索引
反向索引包含文档中文字表示的信息。相比于直接扫描文件，这些信息在允许快速全文搜索时有用的。在这一情况下， map任务需要输入一个文件， 为每一个文档发射一个<word， documentid>集合。reduce函数汇集出现的相同文字，产生一个<word， list(document-id) >对。

（6）分布式序
在这种情况下， MapReduce用于对大量记录进行排序操作的并行执行。这一应用多依赖于MapReduce的运行时特性， 即排序和创建部分中间文件， 而不是由map和reduce任务执行操作。事实上， 这些非常简单：map任务从一个记录中抽取键，并为每一个记录分配一个<key.record>对， reduce任务将简单地复制所有对。排序处理由MapReduce运行时完成， 它会通过键对记录排序以分配键-值对。

（7）其他应用

任何能以两个主要阶段形式表示的计算，都可用MapReduce计算形式表达。

- 分析：这一阶段直接对数据输人文件进行操作， 并与map任务执行的操作保持一致。此外这一阶段的计算期望是高度并行的。
- 聚集：这一阶段以中间结果为基础，其特点是对前一阶段获得的数据进行汇总、求和或细化，以将其呈现为最终形式。这是由reduce函数执行的任务。

4、MapReduce运行机制

（1）MapReduce函数首先把输入文件分成M块，每块大概16-64MB（可以通过参数决定），接着在群集的机器上执行分派处理程序。

（2）集群中由M个任务和R个Reduce任务需要分派，Master选择空闲Worker来分配这些Map或Reduce任务。

（3）Map Worker读取并处理相关输入块，Map函数产生中间结果<key,value>对，暂时缓冲到内存。

（4）中间结果定时写到本地硬盘，将其分成R个区，中间结果在本地硬盘的位置信息将被发送到Master，然后Master负责把这些位置信息传给Reduce Worker。

（5）当Reduce的Worker收到map的中间结果<key,value>对的位置时，它调用远程过程，直接从Map Worker的本地硬盘上读取缓冲的中间数据。当Reduce Worker读到所有的中间数据，他就使用中间的Key进行排序，这样可使相同key的值都在一起。
（6）Reduce Worker根据每一个唯一中间key，来遍历所有的排序后的中间数据，并且把key和相关的中间结果值集合，传递给用户定义的Reduce函数，Reduce函数的结果写到一个最终的输出文件。
（7）当所有的Map任务和Reduce任务都完成的时候，Master激活用户程序。此时MapReduce返回用户程序的调用点。

5、MapReduce变形和扩展
（1）Hadoop
包括：Hadoop分布式文件系统(HDFS) 和Hadoop MapReduce。

（2）Pig
一个分析大量数据集的平台。Pig由表示数据分析程序的高级语言，以及用于评估这些程序的基础设施组成。Pig的基础设施层包括一个用于产生MapReduce工作序列的高级语言的编译器， 运行在Hadoop的分布式基础设施的上层。开发者可以用一种叫作PigLatin的文本语言编写数据分析程序， 一个类似SQL的接口，其特点是以表现性为主、较少的编程工作， 以及一个熟悉的MapReduce接口。

（3）Hive
在Hadoop MapReduce的上层，提供一个数据仓库基础设施。用于简单数据汇总的工具、adhoc即席查询，以及存储在Hadoop MapReduce文件的大量数据集的分析。存在查询延迟，不是一个解决网上交易处理的有效方法。Hive的主要优势在于向外扩展的能力。

（4）Map-Reduce-Merge
将第三阶段(Merge阶段) 引人标准MapReduce管道， 允许对已分割的数据进行有效地融合，并用map和reduce模块分类(或哈希排序) 。Map-Reduce-Merge框架简化了异构的相关数据集的管理，提供一个能够表示普通关系代数运算符和一些连接算法的抽象。

（5）Twiter
允许创建MapReduce工作的迭代执行。Twister还提供附加功能，比如查询静态内存数据的map和reduce任务； 引人combine的附加阶段， 在MapReduce工作的结尾运行，将输出聚集在一起；以及用于数据管理的其他工具。

6、MapReduce替代品

（1）Sphere

- 利用Sector分布式文件系统(SDFS) 的分布式处理引擎。实现流处理模型(单一程序， 多个数据) ，允许开发者以用户定义函数(UDF) 的形式表示计算，UDF针对分布式基础设施运行。UDF的一个特定组合允许Sphere表示MapReducc计算。
- 利用Sector分布式文件系统，建立在Sector用于数据访问的API的上层。UDF根据读写流的程序来表示。流是一种数据结构，提供在SDFS中映射到一或多个文件数据段的集合的访问。UDF的集体执行通过Sphere处理引擎(SPE) 的分布式执行来实现。执行模型是客户机控制的主从模型：Sphere客户机向主节点发送处理请求，主节点返回可用从节点的列表，客户机将选择从节点，从节点执行Sphere进程并协调整个分布式执行。

（2）Al-Pairs

- 一个抽象和运行时环境， 以优化数据密集型工作负载的执行。
- 提供一个简单的抽象， 即All-Pairs函数， 这在很多科学计算领域普通存在：
  All-pairs(A：set， B：set， F：function) 一M：matrix
- 例子出现在生物统计学领域，相似性矩阵由几个包含主题图片的比较结果构成。

6、MapReduce替代品

（3）Dryad LINQ

- 一个微软研究项目， 用于并行写入和分布式程序的编程模型。
- 提供一个基础设施，自动并行执行应用程序，无需开发人员了解分布式和并行编程。开发人员可以将分布式应用程序表示为一组管道连接在一起的顺序程序。即一个Dryad计算可以用一个有向无环图的形式表示， 节点是顺序执行的程序， 顶点表示连接这类程序的通道
- 支持动态修改图形和分区的能力，将图形分为几个阶段执行。
- Dryad LINQ是一个编程环境，从语言综合查询 (LINQ) 扩充到C#生成Dryad计算。框架提供了一个完全集成到.NET的解决方案， 并能够表示多个分布式计算模型， 包括MapReduce。

7、EC2

- 一个IaaS解决方案
- 允许以虚拟机的形式部署服务器，虚拟机创建一个特定映像的实例。映像配有预装的操作系统和软件栈，实例可以配置内存、多个处理器和存储器。
- 用户提供凭证远程访问实例

（1） 亚马逊机器映像：创建虚拟机的模板

（2）EC2实例：代表虚拟机。使用AMI作为模板创建实例， 模板是专门用来选择内核数量、计算能力以及所安装的存储器的； EC2计算单元是一个虚拟内核的运算能力的度量方式， 表示分配给一个实例的真实CPU的预测数量。

（3）EC2环境：负责分配地址、附加存储卷、配置访问控制和网络连接方面的安全。

（4）高级计算服务：

- AWS Cloud Formation，具有EC2特征的实例的简单部署模型的扩展。模板是JSON格式的文本文件， 描述在EC2运行一个应用程序或服务及它们之间的关系所需要的资源。模板提供一个简单的声明来构建复杂的系统， 整合EC2实例与其他AWS服务。
- Beanstalk：一个在AWS云上打包和部署应用程序的简单方法。可以简化供应实例和部署应用程序代码的过程，并提供适当的访问权限。
- MapReduce：采用Hadoop作为MapReduce的引擎， 部署在EC2实例组成的虚拟基础架构上， 使用S3实现存储需求。为AWS用户的MapReduce应用提供云计算平台。

8、S3

- 一个分布式对象存储，允许用户将信息以不同的格式存储。
- 核心部件：桶和对象。桶代表存储对象的虚拟容器，对象代表实际存储的内容。对象通过元数据不断充实，元数据用来为所存储的内容标记附加信息。
- 通过一个表征状态转移(Representational State Transfer， REST) 的接口进行访问。对存储器执行的所有操作以HTTP(GET， PUT DELETE、HEAD和POST) 请求的形式进行， 根据它们的地址元素而执行不同的操作。

（１）资源命名：桶、对象和附加元数据通过REST接口进行访问，由s3.amazonaws.com域中统一资源标识符(URI) 的表示，所有操作的执行表示为指向URI请求的实体。

（２）桶：对象的容器。

- 一个托管在S3分布式存储的虚拟驱动器，提供添加对象的扁平存储方式。
- 桶是S3存储架构的顶层元素，不支持嵌套。
- 桶位于一个特定的地理位置，桶的复制是为了更好的容错能力和内容分布。
- 用户可以选择创建桶的位置，默认情况下在亚马逊的美国数据中心创建。一旦创建了桶，所有属于该桶中的对象将被存储在桶的相同可用区域。
- 桶一旦创建， 不能重命名或迁移。如果有必要的话， 只能删除后再重新创建桶。通过DELETE请求执行删除桶， 当且仅当桶为空时，才可以成功删除。

(3)对象和元数据

- 用户存储文件或向S3发送文本流都表示对象内容。
- 对象通过存储内容的桶内的唯一名称进行标识。该名称用UTF-8编码， 不能超过1024个字节，允许使用几乎任何字符。
- 用户通过指定对象名称、桶名称、内容及其他属性的PUT请求创建一个对象。
- 对象用元数据标记， 通过PUT请求的属性传递。该属性通过GET请求或HEAD请求来检索，检索结果只返回该对象的元数据而无内容。
- 元数据由系统和用户定义、S3使用系统定义的元数据控制与对象的交互；用户定义的元数据对用户更有意义，每个元数据属性可以存储2KB数据，属性表示为键-值对的字符串。

(4)访问控制和安全性

- S3通过访问控制策略(ACP) 访问桶和对象。一个策略允许定义多达100个访问规则， 每个规则可以授权给允许授权者。
- 被授权者可以是单个用户或组。通过S3注册的规范的ID或电子邮件地址来识别用户。对于组而言，只有三个可能的选项：所有用户，身份验证的用户和日志传送用户。
- 资源一旦创建， S3只能将默认ACP授予具有完全控制权限的主人。通过使用资源的URI请求时加上“?acl”可以改变ACP。GET方法可以检索ACP， PUT方法上传一个新的ACP来取代现有的ACP。
- ACP提供一套强大的规则来控制S3用户对资源的访问。所有非验证用户被视为一组，为了提供更细粒度的区分， S3允许定义签名的URI， 提供一个临时的访同令牌，授权在有限的时间内的所有资源访问请求。

(5)高级功能

- S3提供附加功能：服务器访问日志和集成比特流(BitTorrent) 文件共享网络。
- 服务器访问日志让桶所有者获得有关桶及其所有包含对象所提出的请求的详细信息。
- 集成比特流(BitTorrent) 文件共享网络把S3对象公开到比特流网， 允许使用BitTorrent协议下载存储在S3中的文件。要实际下载对象， 它的ACP必须授予每个人读权限。

9、亚马逊弹性块存储(EBS)

- 允许AWS用户提供EC2实例， 以卷的形式持久存储， 在实例启动时安装卷。
- EBS卷可以被克隆， 用作引导分区，并构成持久存储。
- EBS卷通常存储在EC2实例的同一可用性区域内， 这将利用它们的最大化I/O性能。也可以连接位于不同可用区的卷。一旦安装为卷，其内容根据操作系统的要求在后台延迟加载，降低进人网络的/O请求的数量。卷的镜像不能在实例间共享，但可以从镜像创建多个(独立的)活动卷。此外，可以将多个卷连接到单个实例或从一个给定的快照建立卷，如果格式化文件系统允许，还可以修改卷的大小。
- 卷的费用包括在S3中所占的存储量和对卷执行I/O请求的数量所产生的成本。

10、亚马逊弹性缓存

- 实现一种基于EC2实例的集群弹性内存缓存。通过Memcached兼容协议，从其他EC2实例实现快速数据访问，因此现有应用程序无需修改就可透明地迁移到弹性缓存。
- 基于运行缓存软件的EC2实例的集群， 通过Web服务提供。一个弹性缓存集群可以根据客户端应用程序的需求进行动态调整。
- 此外自动补丁管理，故障检测和缓存节点的恢复等功能支持缓存集群持续运行无需AWS用户的管理。仅当需要的时候， 用户才弹性地改变群集大小。
- 弹性缓存节点根据EC2的成本核算模式定价，使用安装此实例的缓存服务不同，因此会有一点小的价格差异。

11、结构化存储解决方案

(1)预先配置的EC2AM1

- 预定义的模板，用于安装一个指定的数据库管理系统。
- 从AMI创建的EC2实例，通过持久存储的EBS卷完成。AMI包括：IBM DB2、Microsoft SQL Server、MySQL、Oracle， PostgreSQL、Sybase和Vertica。
- 实例根据EC2的成本模式按小时定价。

(2) 亚马逊RDS

- RDS是关系数据库服务， 依赖于EC2基础设施， 由Amazon.Developers管理。
- 通过AWS控制台或一个特定的Web服务，为用户提供了自动备份、快照、时间点恢复以及执行复制的能力。关系数据库引擎：MySQL和Oracle。
- muli-AZ部署功能：为用户提供RDBMS解决方案故障转移的基础设施。
- muli-AZ读副本功能：针对严重依赖于数据库读取的应用程序，提供更高的性能。亚马逊部署主服务副本仅用于数据库中读取，从面减少了服务的响应时间。
- 对基于Oracle和MySQL引擎的应用程序迁移到AWS的基础设施， 且需要一个可扩展的数据库解决方案的情况，该方案是最佳解决方案。

(3)亚马逊简单数据库

- 一种轻量级的、高度可扩展的、灵活的数据存储解决方案。
- 支持半结构化数据，基于域、项和属性的概念，该模型对输人数据的结构限制较少，在查询大量数据方面具有较高性能。
- 使用域作为顶层元素来组织数据存储，域相当于关系模型中的表，允许项具有不同的列结构，每个项表示为一个键-值对属性的集合。
- 实现一个宽松的约束模型，从而产生最终一致的数据。最终一致是指非常短时间内对同一数据的多个访问可能读取的值不相同，但经过一段时间最终会相同。
- 允许客户端有条件的插人或删除，当有多个写人时，可以防止丢失更新。当且仅当条件验证有效时才会执行该操作，用来检查项的属性预先存在值。
- S3用于存储大对象更便宜；SimpleDB适合小对象半结构化数据快速访问，而不是用于大对象的长期存储。

12.亚马逊Cloud Front

- 实现内容交付网络。按照一定策略利用位于全球范围的边界服务器集合，满足静态和流媒体网页内容请求，尽可能减少传输时间。
- AWS为用户提供简单的Web服务API来管理Cloud Front。
- Cloud Front交付的内容是静态的(HTTP和HTTPS) 或流(实时消息协议或RMTP) 。托管分发内容的原始副本的源服务器可以是一个S3的桶、一个EC2实例或一个亚马逊网络外部的服务器。用户可以限制只有一个或几个可用的协议访问分发，设置访问规则进行更精细的控制。

13、虚拟网络(VPC)

- 由一组服务组成， 允许AWS用户控制计算和存储服务，以及它们间的连通性。在基础设施方面， **虚拟网络**和**直接连接**提供连接解决方案， Route53在命名方面使连接更容易。
- VPC在基础设施内部和外部创建虚拟专用网络。服务供应商设计常用网络场景的模板，或者配置一个完全可定制的网络服务。模板包括公共子网、独立网络、通过网络地址转换(NAT) 接入因特网的专用网络， 以及包含AWS资源和私有资源的混合网络。还可通过使用身份访问管理(IAM) 服务控制不同的服务(EC2实例和S3桶) 间的连通性。
- 直接连接允许AWS用户创建用户私网和端口（亚马逊直接连接位置）之间的专用网络。可以进一步分割多个逻辑连接， 提供访问Amazon基础设施的公共资源。
- Route53实现动态域名解析服务，允许通过不同于amazon.com域的域名来获取AWS资源。利用亚马逊DNS服务器的大型全球分布式网络， AWS用户可以公开EC2实例或S3存储桶作为其属性域内的资源。
- 通过与Route 53Web服务进行交互， 用户可以管理一组托管区， 用户域由服务控制，通过它编辑资源。

14、**消息**

- 包括简单队列服务(SQS) ， 简单通知服务(SNS) 和简单电子邮件服务(SES) 。
- SQS是非连接模型， 通过消息队列的方式在应用程序之间交换消息， 在AWS基础设施中提供的功能。
- SNS提供用于连接异构应用程序的发布-订阅方法。SQS必须不断轮询指定的队列以处理一个新消息， 当有感兴趣的新内容时， SNS会通知应用程序此功能可以通过Web服务获取， 这样AWS用户可以创建一个其他应用程序可以订阅的主题。
- SES为AWS用户提供充分利用AWS基础设施的可扩展的电子邮件服务。

15、附加服务

- 允许用户利用聚合的服务：亚马逊Cloud Watch和亚马进灵活支付服务(FPS)。
- Cloud Watch提供一种全面的统计数据服务， 帮助开发人员理解和优化托管在AWS上的应用执行。
- FPS的基础设施允许AWS用户利用亚马逊的计费设施向其他AWS用户出售商品和服务。