



北京邮电大学

Beijing University of Posts and Telecommunications

大数据的处理与存储

石瑞生

网络空间安全学院

- 大数据的计算与存储
 - 分割与聚合
- 大数据存储的安全隐私
 - 数据完整性机制：POR
 - 隐私保护机制（加密数据去重技术）：CE, MLE
 - 安全防护机制：PoW
- 保护隐私的计算
 - 同态加密，安全多方计算

大数据的计算与存储

- 海量数据对计算能力提出了很高的要求
 - 如何（低成本）快速处理多样化的海量数据？
- 最早的商业化大数据服务系统
 - 搜索引擎
 - 电子商务

云计算的定义与特征

- NIST (National Institute of Standards and Technology, 美国国家标准与技术研究院) 对云计算的定义: “云计算是一种模式, 能以泛在的、便利的、按需的方式通过网络访问可配置的计算资源 (例如网络、服务器、存储器、应用和服务), 这些资源可实现快速部署与发布, 并且只需要极少的管理成本或服务提供商的干预。”

云计算一般具有以下五大特征:

- 1) 按需获得的自助服务;
- 2) 广泛的网络接入方式;
- 3) 资源的规模池化;
- 4) 快捷的弹性伸缩;
- 5) 可计量的服务。

- 从技术角度看，云计算是分布式计算、并行计算、网格计算、多核计算、网络存储、虚拟化、负载均衡等传统计算机技术发展到一定阶段，和互联网技术融合发展的产物。
- 云计算的目标在于通过互联网把无数个节点（即计算实体）整合成一个具有强大计算能力的“巨型机”系统，把强大的计算能力提供给终端用户。
- 技术路线
 - MapReduce/GFS/NoSQL：计算，存储，数据管理
 - 虚拟机/容器：细粒度的资源共享机制

计算架构的演变

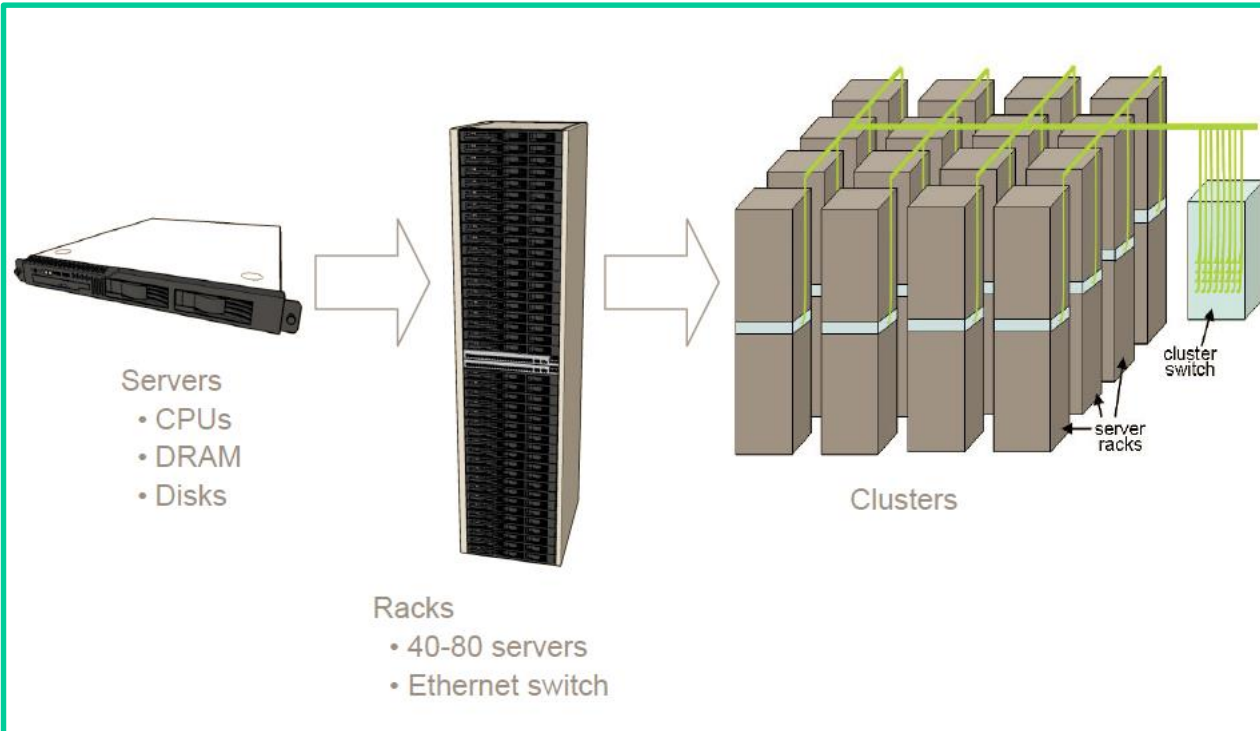


通过不断提高云计算平台的处理能力，减少用户终端的处理负担，使得用户终端可以简化成低配的计算终端，让用户享受到按需使用云计算强大的计算处理能力的服务。

云计算不仅改变了网络应用的模式，也将成为带动IT、物联网、电子商务等诸多产业强劲增长、推动信息产业整体升级的基础。

- 大型应用大多采用C/S架构，C代表Client客户端，S代表Server服务器端。
 - 发展趋势是C越来越小，S越来越大

云在何处? -- 数据中心



云计算三种主要服务模式

- SaaS (Software as a Service, 软件即服务) , 以服务的方式将应用软件提供给互联网最终用户。
 - 开发商将应用软件统一部署在自己的服务器上, 客户可以根据自己实际需求, 通过互联网向开发商定购所需的应用软件服务, 按定购的服务多少和时间长短支付费用, 并通过互联网获得服务。
 - 用户无需购买及部署软件, 也无需对软件进行维护, 所有的数据都存储在开发商的服务器上, 用户所需要的只是在任意一台电脑上打开浏览器, 登录帐号, 即可使用相关服务。
 - 典型SaaS应用如Salesforce的Sales Cloud (在线CRM) , 微软的Office Online (在线办公系统) , 用友在线财务系统等。
- PaaS (Platform as a Service, 平台即服务) , 以服务的方式提供应用程序开发和部署平台。

云计算三种主要服务模式

- PaaS (Platform as a Service, 平台即服务) , 以服务的方式提供应用程序开发和部署平台。
 - 就是指将一个完整的计算机平台, 包括应用设计、应用开发、应用测试和应用托管, 都作为一种服务提供给客户。
 - PaaS服务主要面对应用开发者, 在这种服务模式中, 开发者不需要购买硬件和软件, 只需要利用PaaS平台, 就能够创建、测试和部署应用和服务, 并以SaaS的方式交付给最终用户。
 - 典型的PaaS服务如谷歌的AppEngine (应用程序引擎) , 微软的Azure平台, Salesforce的Force.com等。

云计算三种主要服务模式

- IaaS (Infrastructure as a Service, 基础设施即服务) , 以服务的形式提供服务器、存储和网络硬件以及相关软件。
 - 它是三层架构的最底层, 是指企业或个人可以使用云计算技术来远程访问计算资源, 这包括计算、存储以及应用虚拟化技术所提供的相关功能。
 - 计算: 虚拟机 (VPS, Virtual Private Server)
 - 存储: S3, 云盘
 - 无论是最终用户、SaaS提供商还是PaaS提供商都可以从基础设施服务中获得应用所需的计算能力, 但却无需对支持这一计算能力的基础IT软硬件付出相应的原始投资成本。
- 全世界范围内知名的IaaS服务有亚马逊的AWS、微软的Azure、谷歌的谷歌云等, 国内有阿里巴巴的阿里云、腾讯的腾讯云、电信的天翼云等。

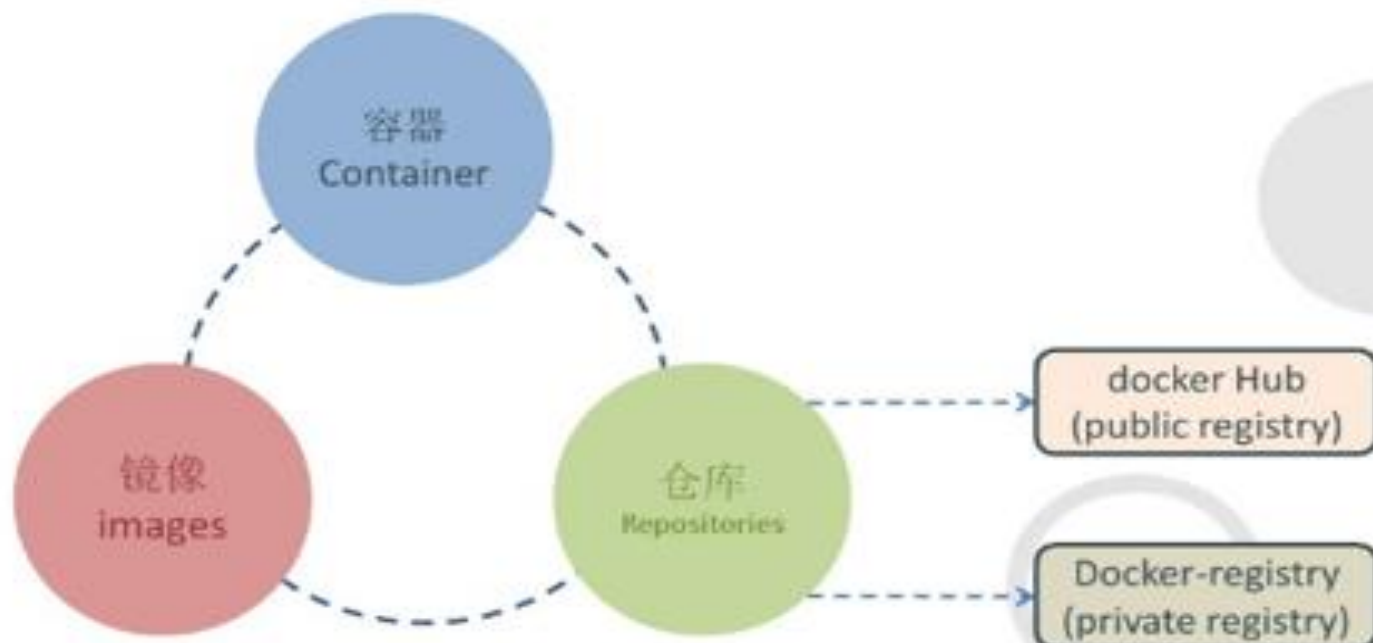
- 虚拟化技术作为云计算的基础技术之一，在云服务系统中发挥了不可替代的作用。
 - 从服务提供方视角来看，虚拟化技术有助于提高资源的利用率。
 - 从用户的视角，提供一个共享的资源池，降低用户的总拥有成本、系统部署和维护的时间成本，给用户提供更方便又便宜的（计算、存储、通信）服务。
 - 成本更低，用户体验更好，是虚拟化技术得以迅猛发展的根本驱动力。
- 重点介绍一下虚拟机技术和容器技术

- 虚拟机技术是云计算系统提高计算资源利用率的重要手段。
 - 绝大多数网站的访问流量都是不均衡的：早晚；季节性（圣诞节）；突发事件。
 - 网站运营者为了应对这些突发流量，不得不按照峰值来配置服务器和网络资源，造成资源的平均利用率只有10%到15%。
 - 云计算系统通过虚拟化技术，可以构建一个超大规模的资源池；对于每个租用者，可以根据需要动态地为其分配资源和释放资源，不需要按照峰值预留资源。

- 计算虚拟化技术的实现形式，是在系统中加入一个虚拟化层，将下层资源抽象成另一种形式的资源，供上层调用。
 - 计算虚拟化技术的通用实现方案，是将软件和硬件相互分离，在操作系统与硬件之间加入一个虚拟化软件层，通过空间上的分隔、时间上的分时，将物理资源抽象成逻辑资源，向上层操作系统提供一个与它原先期待一致的服务器硬件环境，使得上层操作系统可以直接运行在虚拟环境上，并允许具有不同操作系统的多个虚拟机相互隔离，并发运行在同一台物理机上，从而提供更高的IT资源利用率和灵活性。
- 计算虚拟化软件，需要模拟出来的逻辑功能，主要为高效独立的虚拟计算机系统，我们称之为虚拟机。在虚拟机中运行的操作系统软件，我们称之为，Guest-OS。
 - 虚拟化软件层模拟出来的每台虚拟机都是一个完整的系统，它具有处理器、内存、网络设备、存储设备和BIOS。
 - 在虚拟机中运行应用程序及操作系统，和在物理服务器上运行，并没有本质区别。

- 与传统虚拟化等技术相比，容器技术在生产应用中优势明显。
 - 相比虚拟化技术，容器技术具有部署便捷、管理便利、利于微服务架构的实现、弹性伸缩、高可用等特点。
- 容器技术正在快速改变着公司和用户创建，发布，运行分布式应用的方式。

容器技术的基本概念



容器技术有三个核心的概念：镜像（Images），容器（Container），仓库（Repositories）。

镜像：文件的层次结构，以及包含如何运行容器的元数据，**Dockerfile**中的每条命令都会在文件系统中创建一个新的层次结构，文件系统在這些层次上构建起来，镜像就构建于这些联合的文件系统之上。

容器：容器是从镜像创建的运行实例。它可以被启动、开始、停止、删除。每个容器都是相互隔离的、保证安全的平台。**可以把容器看做是一个简易版的Linux环境，Docker利用容器来运行应用。**

仓库：仓库是集中存放镜像文件的场所，仓库注册服务器（**Registry**）上往往存放着多个仓库，每个仓库中又包含了多个镜像，每个镜像有不同的标签（**tag**）。

目前，最大的公开仓库是 **Docker Hub**，存放了数量庞大的镜像供用户下载。**Docker**仓库用来保存我们的 **images**，当我们创建了自己的 **image** 之后我们就可以使用 **push** 命令将它上传到公有或者私有仓库，这样下次要在另外一台机器上使用这个 **image** 时候，只需要从仓库上 **pull** 下来就可以了。

容器技术的架构与实现原理



容器技术的实现依赖于三个核心技术：隔离机制（namespaces），资源配额（Cgroups），虚拟文件系统

Namespaces将容器的进程、网络、消息、文件系统隔离开，给每个容器创建一个独立的命名空间。

Cgroups技术实现了对资源的配额和度量。

LXC，即Linux Container，提供了一种操作系统级的虚拟化方法，借助于namespace的隔离机制和Cgroups限额功能来管理container。

Docker的AUFS：Docker镜像位于bootfs之上；每一层镜像的下面一层称为其父镜像，相邻两层镜像之间为父子关系；第一层镜像为Base Image，容器在最顶层，其下的所有层都为readonly；Docker将readonly的FS层称作Image。

AUFS (Advanced Multi-layered Unification Filesytem)

AUFS 的全称是 Advanced Multi-layered unification filesytem, 它的主要功能是: 把多个目录结合成一个目录, 对外使用。

把多个目录 mount 成一个, 读写操作步骤如下:

1. 默认情况下, 最上层的目录为读写层, 只能有一个
2. 下面可以有一个或者多个只读层读文件
3. 读文件, 从最上面一个开始往下逐层去找, 打开第一个找到的文件, 读取其中的内容
4. 写文件, 如果在最上层找到了该文件, 直接打开
否则, 从上往下开始查找, 找到文件后, 把文件复制到最上层, 然后再打开这个 copy
5. 删除文件: 在最上层创建一个 whiteout 文件, `.wh.<origin_file_name>`, 就是在原来的文件名字前面加上 `.wh.`

容器与虚拟机的对比

一般来讲，虚拟机的操作系统是运行在宿主机操作系统之上的，而容器是与宿主机共享一个操作系统。

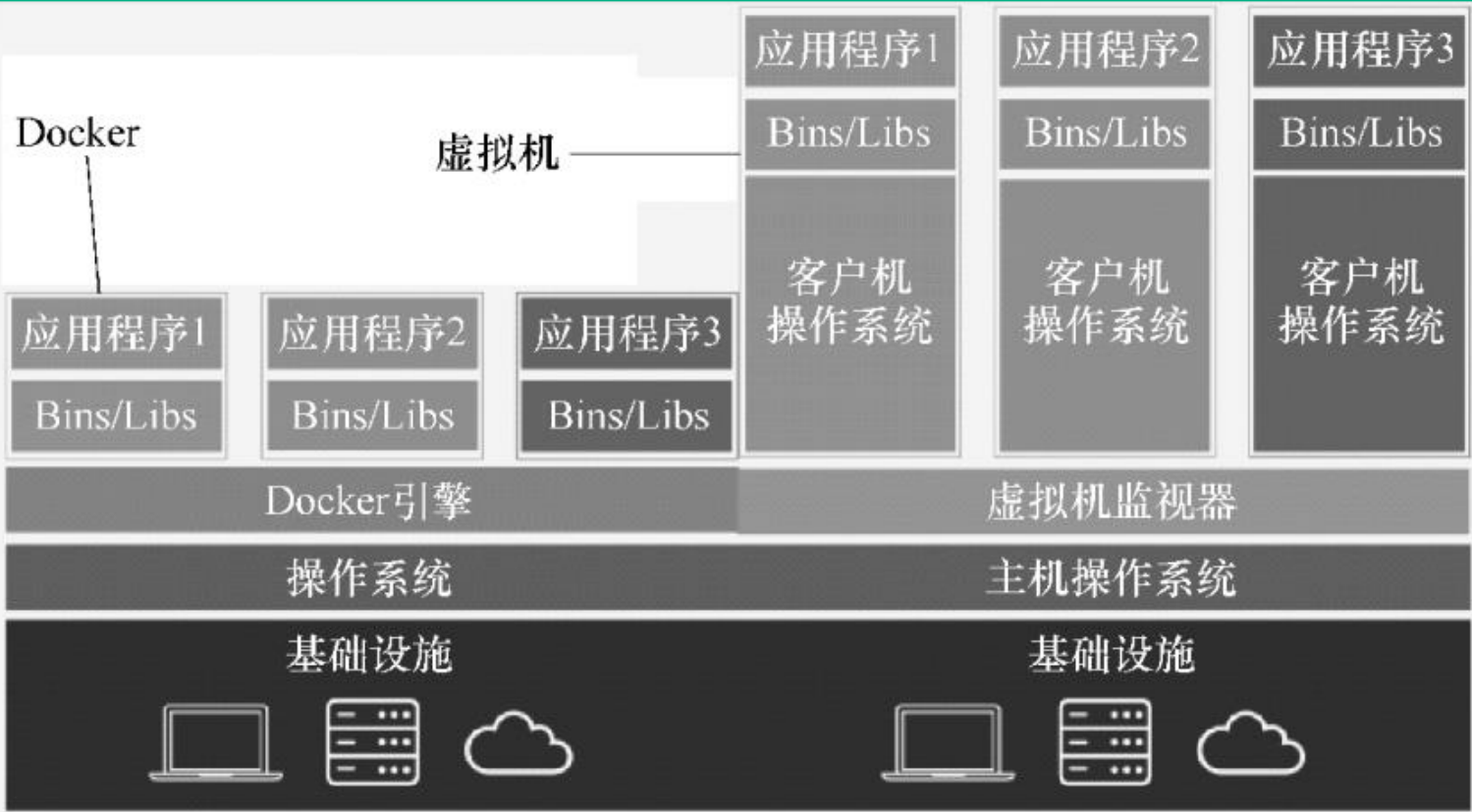


图 5-6 容器技术与虚拟机技术的对比

- 1) 虚拟机镜像庞大；而容器镜像小，便于存储和传输。
- 2) 虚拟机需要消耗更多的CPU和内存，容器几乎没有额外的性能损失。
- 3) 虚拟机部署速度慢，启动需要10秒以上；而容器启动速度快，以Docker为例，一般是秒级的速度。

- 容器的安全
- 虚拟机的安全

- 针对容器的安全攻击

- 2017美国黑帽大会上，Aqu Security公司研究人员Michael Cherny与Sagie Dulce指出，Aqua Security研究人员塞奇·杜尔塞曾提出这种概念验证（PoC）攻击，并首次演示了这种技术。
- Docker承认存在这个问题，并表示这是由于先前所有Docker for Windows版本允许通过TCP/HTTP远程访问Docker。Docker改变了默认配置，关闭了HTTP端口，以此防止访问Docker Daemon。

- 容器的秘密管理

- 容器应用环境中秘密，指的是需要保护的访问令牌、口令和其他特权访问信息。容器需要安全机制来保护这些特权访问信息。

- 沙箱容器

- 这种容器有助于在主机操作系统和容器里面运行的应用程序之间提供了一道安全的隔离边界。

容器的安全 – 沙箱容器

gVisor (<https://github.com/google/gvisor>)，这种新型的沙箱有助于为容器提供安全隔离机制，同时比虚拟机更轻量级。**gVisor**与**Docker**和**Kubernetes**集成起来，因而在生产环境下运行沙箱容器轻而易举。

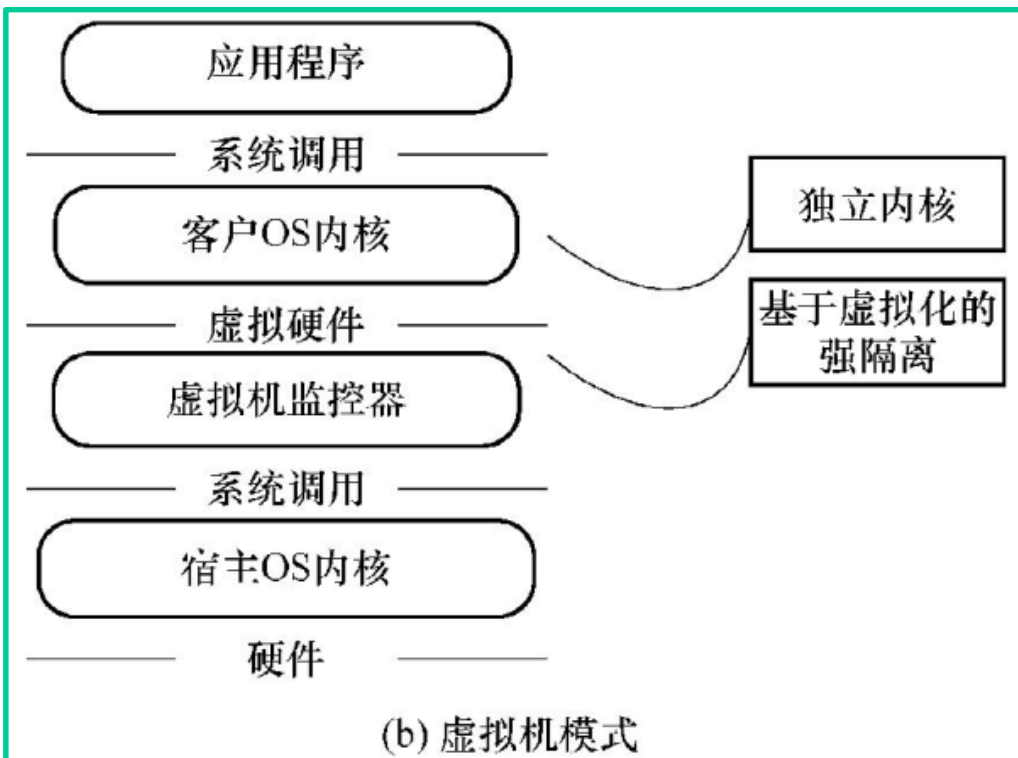
传统Linux容器中运行的应用程序访问系统资源的方式与常规（非容器化）的应用程序一模一样：直接对主机内核进行系统调用。内核在特权模式下运行，因而得以与必要的硬件交互，并将结果返回给应用程序。

如果是传统的容器，内核对应用程序所能访问的资源施加一些限制。这些限制通过使用 **Linux**控制组（**cgroup**）和命名空间来加以实现，然而并非所有的资源都可以通过这种机制来加以控制。此外，即使有这样的限制，内核仍然暴露了很大的攻击面，恶意应用程序可以直接攻击。

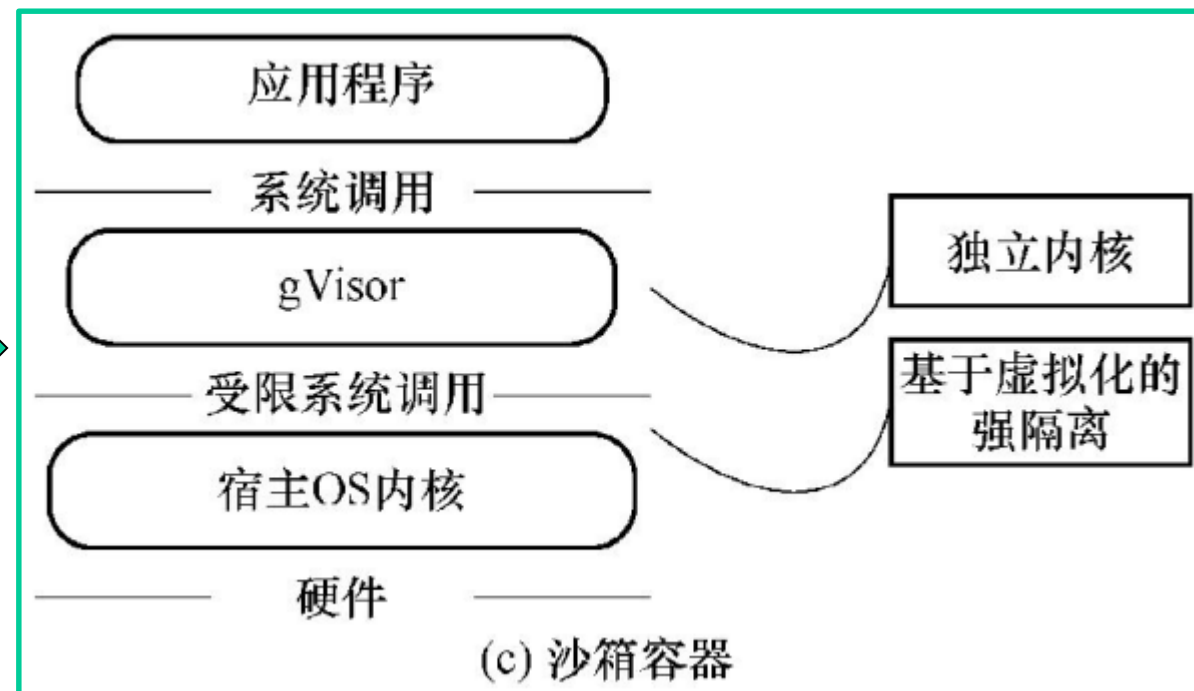
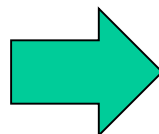
过滤器这样的内核特性，可以在应用程序和主机内核之间提供更好的隔离，但是它们要求用户为系统调用创建预定义的白名单。实际上，常常很难事先知道应用程序需要哪些系统调用。如果发现你应用程序需要的系统调用中存在着漏洞，过滤器提供的帮助也不大。



(a) 传统Linux容器



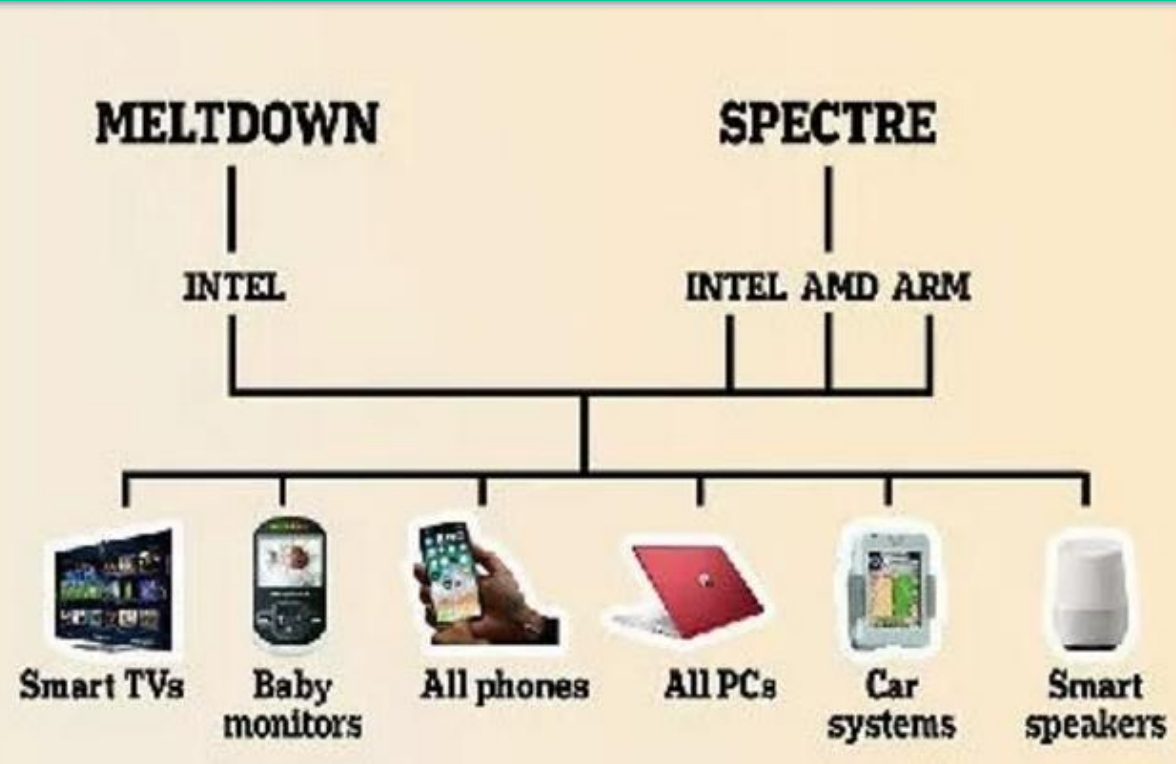
提高容器隔离效果的一种方法是让每个容器在其自己的虚拟机里面运行。在不同的虚拟机中运行容器提供了出色的隔离、兼容性和性能，但可能也需要占用更多的资源。



gVisor比虚拟机更轻量，同时保持相似的隔离级别。gVisor的核心是作为一个普通的非特权进程来运行的内核，它支持大多数Linux系统调用。

虚拟机的安全 – 共用一台物理设备安全吗？

由于多个用户的虚拟机共享一台物理机，那么，是否虚拟用户A有可能获得同一台机器上的虚拟用户B的隐私数据呢？



2017年底，Intel CPU爆出Bug，该缺陷存在于过去十年生产的现代英特尔处理器中。它让普通的用户程序（从数据库应用软件到互联网浏览器中的JavaScript）在一定程度上得以发现受保护内核内存里面的数据。

两种攻击模式：

一种被称为Meltdown，是在用户态攻击内核态，造成内核信息泄露。

另一种被称为Spectre，一个应用可以突破自己的沙盒限制，获取其他应用的信息。

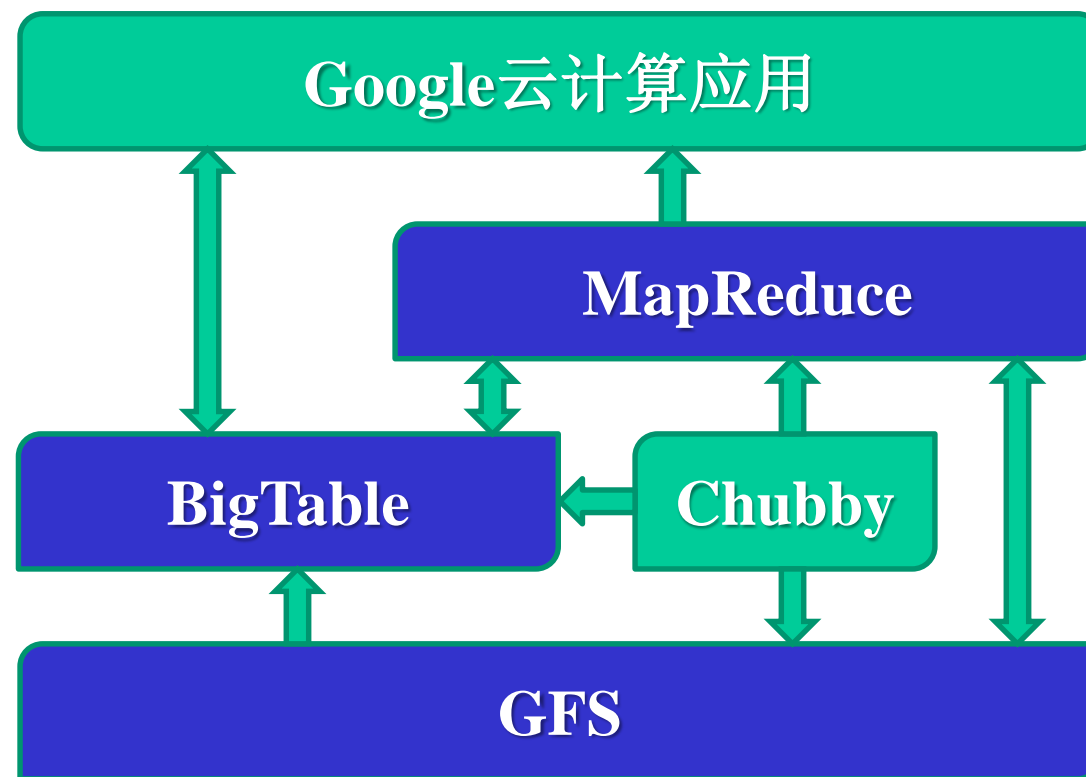
问题与思考

- 云计算带来了很多好处
- 但是，同时带来了新的安全威胁
 - 云计算这种新的计算环境安全可信吗？
 - 威胁：
 - 入侵云服务提供商服务器的黑客；
 - 云服务提供商的内部攻击者；
 - 恶意用户；
- 挑战
 - 如何在不可信的环境下完成计算？
- 解决方案：1) SGX； 2) 同态加密； 安全多方计算

产业发展的视角

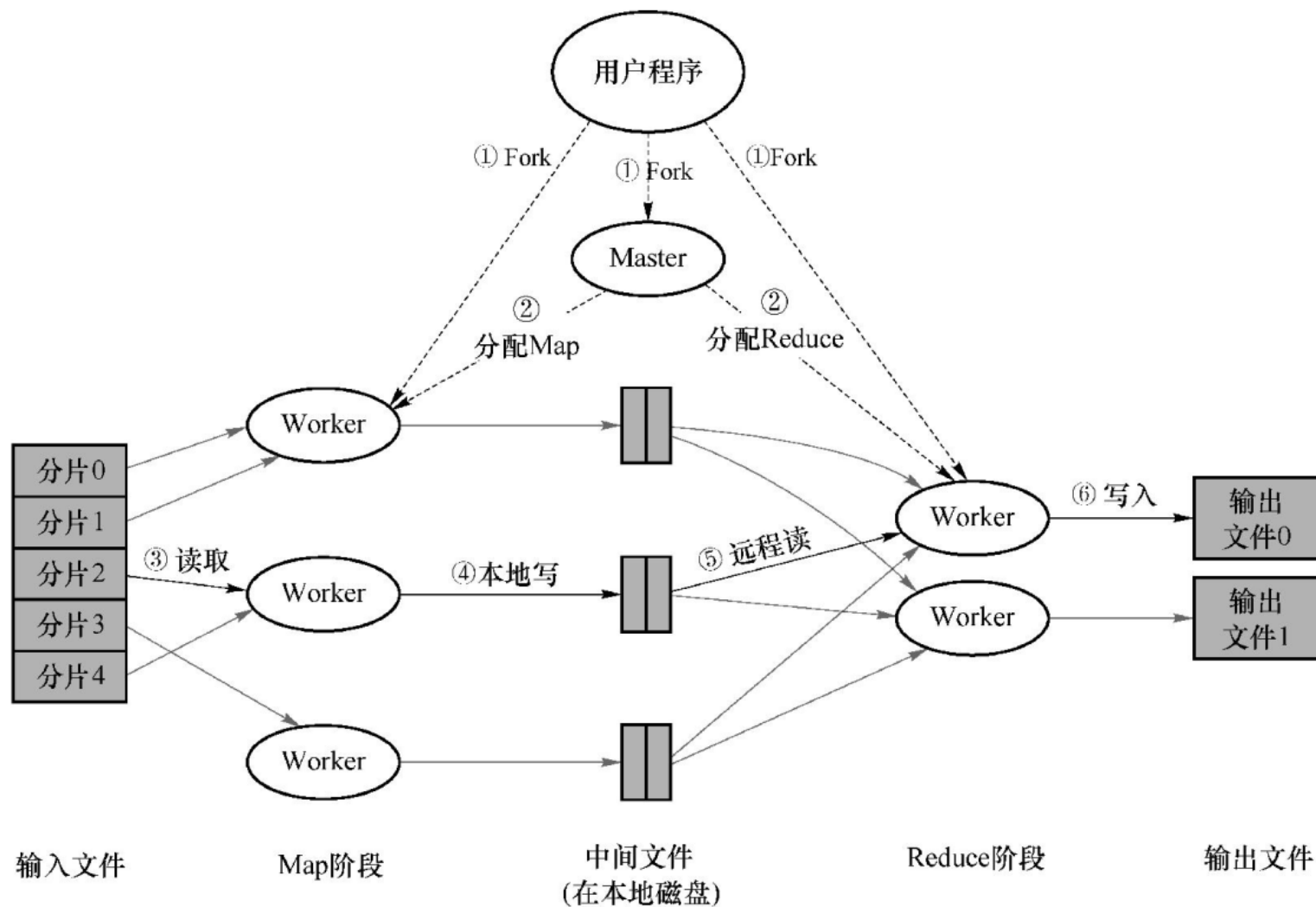
- 从产业发展角度来看，互联网的快速发展使得大众可以参与到信息的制造和编辑，从而导致信息出现无限增长的趋势，这是云计算产生的根源。
- 而摩尔定律的终结，意味着依靠硬件性能的提升无法解决信息无限增长的问题。怎样低成本高效快速地解决无限增长的信息存储和计算问题是一个摆在科学家面前的难题。
 - Scale out（横向扩展） vs. Scale Up（纵向扩展）
- 云计算的出现恰好可以解决这个问题，同时它还使得IT基础设施可以资源化、服务化，使得用户可以按需定制自己的计算资源。
 - 外包计算（成本的规模效应）

- MapReduce
- GFS
- BigTable



Google Chubby是一个分布式锁服务，Chubby底层一致性实现就是以Paxos为基础的。

MapReduce




Google设计GFS的动机

- Google需要一个支持海量存储的文件系统
 - 购置昂贵的分布式文件系统与硬件?



传统的方式如何提高文件系统的可靠性并实现容错?

镜像, RAID5,

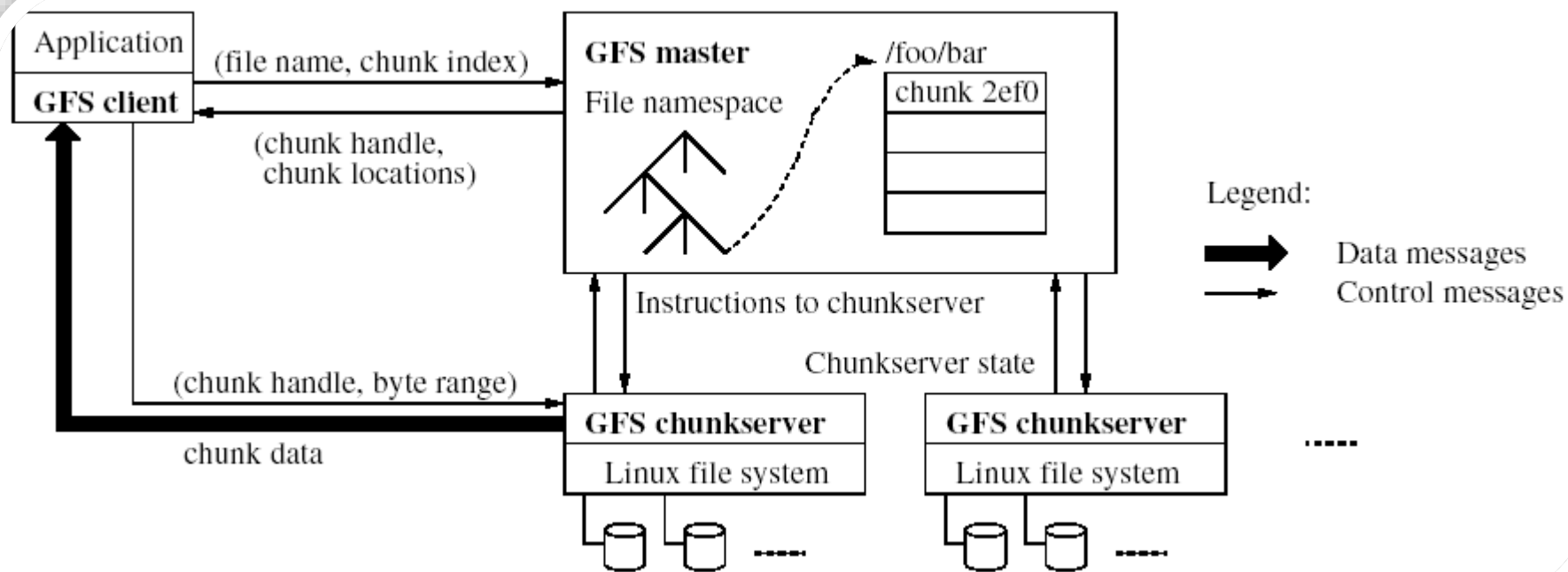
【Scale Up, 增强单节点的可靠性】

A photograph of the Google Garage building, showing the colorful Google logo on the roof and a checkered floor in the foreground.

是否可以在一堆廉价且不可靠的硬件上构建可靠的分布式文件系统?

A close-up photograph of the Google Garage sign, showing the word 'Google' in its multi-colored font and 'Garage' in a gold, 3D font.A photograph of two men sitting in the Google Garage. One man is sitting on a wooden stool, and the other is sitting on a wooden table. They are surrounded by computer equipment and papers.

Google文件系统（Google File System，GFS）是一个大型的分布式文件系统。它为Google云计算提供海量存储，处于所有核心技术的底层。



GFS的新颖之处在于它采用廉价的商用机器构建分布式文件系统，同时将GFS的设计与Google应用的特点紧密结合，简化实现，使之可行，最终达到创意新颖、有用、可行的完美组合。GFS将容错的任务交给文件系统完成，利用软件的方法解决系统可靠性问题，使存储的成本成倍下降。GFS将服务器故障视为正常现象，并采用多种方法，从多个角度使用不同的容错措施，确保存储数据的安全、保证不间断的数据存储服务。

- 超过50个GFS集群
- 每个集群包含数千个存储节点
- 管理着PB(10^{15} Byte)级的数据



巨型、廉价、稳定的数据中心



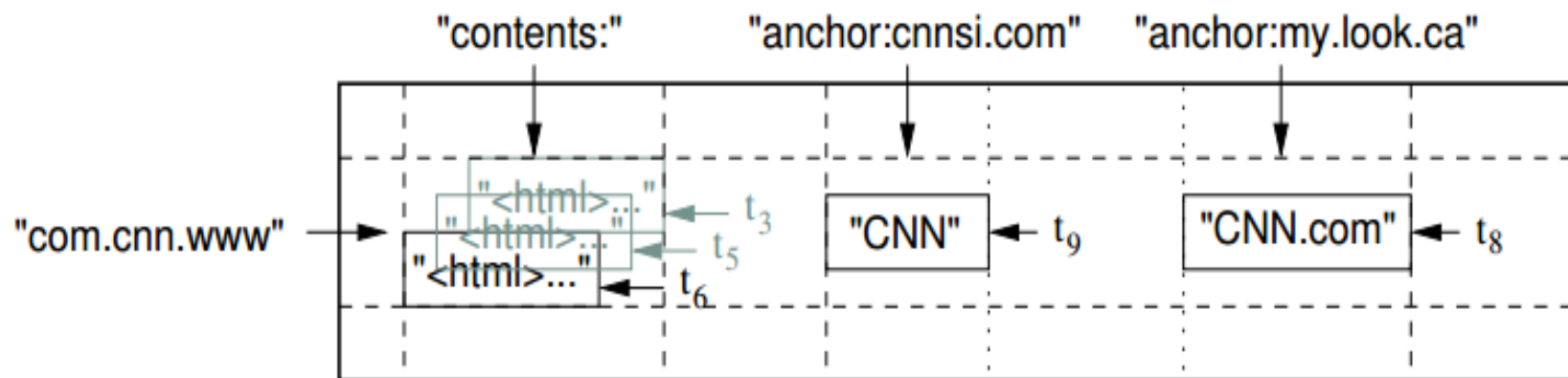
BigTable



Bigtable建立在前面介绍过的GFS之上。相比于GFS中的无结构数据，Bigtable对数据建立了结构化的模型，更适合进行数据分析。

因此，Google的许多服务都建立在Bigtable而并不是直接在GFS上，包括Google的索引、许多MapReduce应用、Google地图、Youtube、Gmail等。

Bigtable的数据模型是一个分布式多维表格，表中的数据通过一个行关键字（Row Key）、一个列关键字（Column Key）以及一个时间戳（Time Stamp）进行索引。





Row Key	Time Stamp	Column Contents	Column Anchor		Column "mime"
			cnnsi.com	my.look.ca	
"com.cnn.w ww"	T9		CNN		
	T8			CNN.COM	
	T6	"<html>.. "			Text/html
	T5	"<html>.. "			
	t3	"<html>.. "			

Row Key	Time Stamp	Column: Contents
Com.cnn.www	T6	"<html>.."
	T5	"<html>.."
	T3	"<html>.."

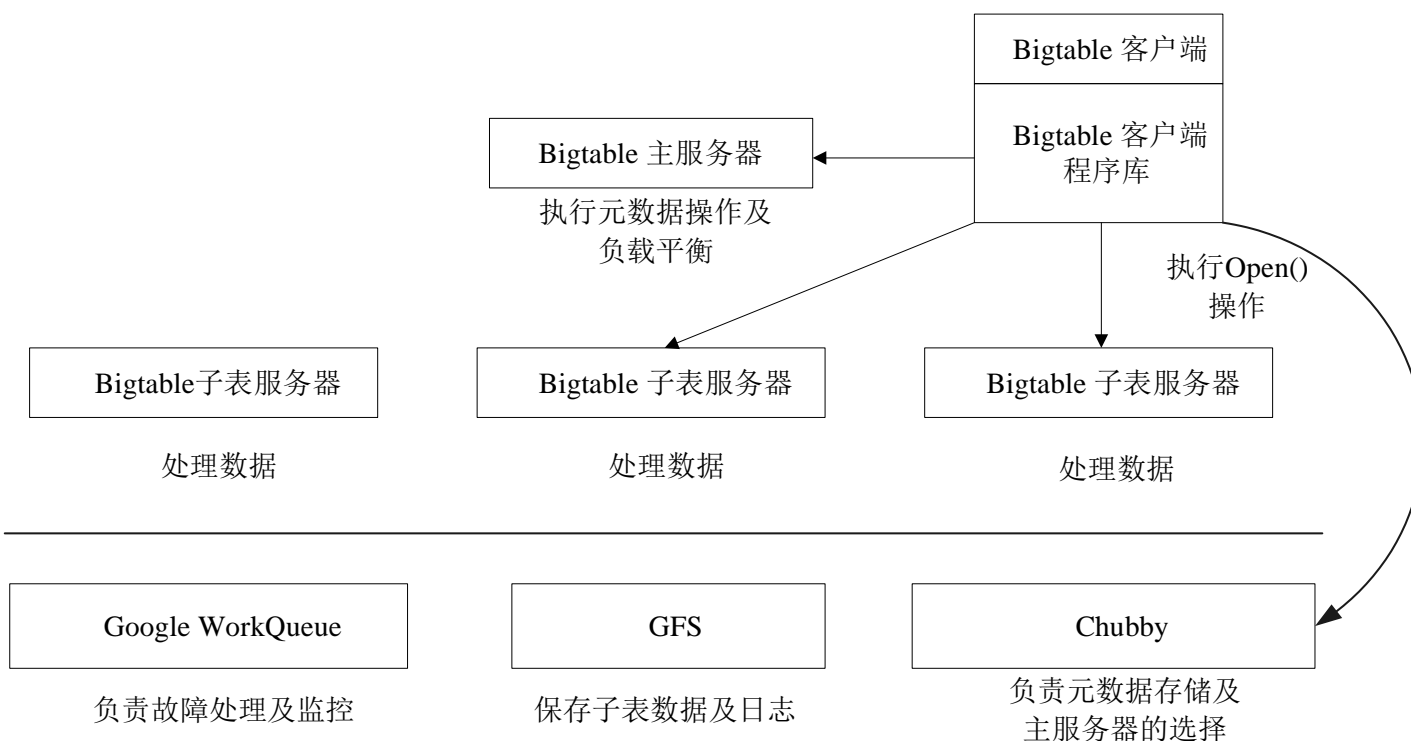
Row Key	Time Stamp	Column: Anchor	
Com.cnn.www	T9	Anchor:cnnsi.com	CNN
	T8	Anchor:my.look.ca	CNN.COM

Row Key	Time Stamp	Column: mime
Com.cnn.www	T6	text/html

BigTable的系统架构



有别于传统的关系数据库，Bigtable允许动态地对表格的列进行添加删除。
与GFS和MapReduce类似，Bigtable也同样在数千台廉价的商业PC上构建，用于支持PB级别的海量数据。



为此，Bigtable根据表格行主键将表格进行分割，形成若干被称为tablet（子表）的存储单元，（每个Tablet由多个文件组成）存储在GFS中。

- 1) Tablet在GFS中的位置被存储在主服务器（Master）中。
- 2) Bigtable利用Chubby（一个分布式锁系统）来管理tablet信息，包括保证可用性、存储tablet位置、保存各个Bigtable的列信息与访问权限、动态监测各个服务器的状态等。

- Google云计算的灵魂人物



Jeff Dean是一个谷歌的系统基础设施部门的研究员。
在加入谷歌之前，他工作于Digital Equipment Corporation的西方研究实验室（Western Research Laboratories），他在2009年当选了美国国家工程院院士。
Jeff Dean是ACM的一个资深会员，他曾以优异成绩取得了美国明尼苏达大学的计算机科学学位，并随后又获得了华盛顿大学的计算机科学博士学位。



Sanjay Ghemawat是谷歌的系统基础设施部门的研究员。
他是Digital Equipment Corporation的系统研究中心的前研究员，于2009年被选为美国国家工程院院士。
他在康奈尔大学得到了一个学士学位，并从麻省理工学院获得了计算机科学专业的硕士和博士学位。

- 随着大数据时代的来临，流行长达20年的传统关系数据库已经失去了信息系统的标准数据存储方式的地位，数据的存储方式呈现出多元化趋势。

关系型数据库的发展

- 1970年, Edgar Codd提出了关系型数据库模型
 - 1963年, Edgar Codd在美国密歇根大学攻读博士学位。科德获得计算机博士学位时已经42岁。大器晚成的他于1967年又回到IBM做研究工作。3年后, 科德博士在《美国计算机学会通讯》(CACM) 上发表了关系数据库的开山之作“大型共享数据库数据的关系模型”(A Relational Model of Data for Large Shared Data Banks)。
 - 科德的关系数据库将数据操作从具体的计算机软件环境和物理存储模式中独立出来。
 - 1981年, Codd博士获得图灵奖。
- 1973年, 加州大学伯克利分校的Michael Stonebraker教授开始基于DEC小型机的UNIX系统的关系数据库INGRES的研发工作。
 - 1974-1985, 从INGRES衍生出多个商业版关系数据库产品 (包括Sybase, Informix)
- 1976年, Oracle公司成立
- 1980年, Stonebraker成立了RTI(Relational Technology, Incorporated)公司, 开始了INGRES数据库的商业推广。
- 90年代中期, 开源软件运动产生了Linux和MySQL
 - 2000年后, 随着云计算技术的推广, 越来越多的公司开始采用Linux和MySQL, 这使得MySQL的市场占有率直逼甲骨文。

- 1988年， IBM的研究人员第一次提出了数据仓库(information warehouse)的概念。
- 1992年， Bill Inmon提出以第三范式为基础的搭建在关系数据库之上的“企业信息工厂”概念， 被业界称为由上向下的数据仓库开发模式。
- Ralph Kimball提出了**抛开关系数据库模式**， 用“事实表”加“维度表”的星型模式搭建企业各个部门需要的**数据超市**， 然后通过合并相同的维度表， 形成维度表总线矩阵， **由下向上形成企业数据仓库**的开发方式。因为金博尔的架构更有利于用户理解和分析数据， 所以更受欢迎。

NoSQL – CAP猜想



- 2000年, PODC年会上UCB的Eric Brewer教授提出CAP猜想。
- 2002年, MIT的Nancy Lynch教授证明该猜想成立。
- 2003年, GFS; 2004年, MR。BigTable, 2006 OSDI
- 2004-2006, Hadoop的出现
- 2007年, Dynamo (亚马逊) 。

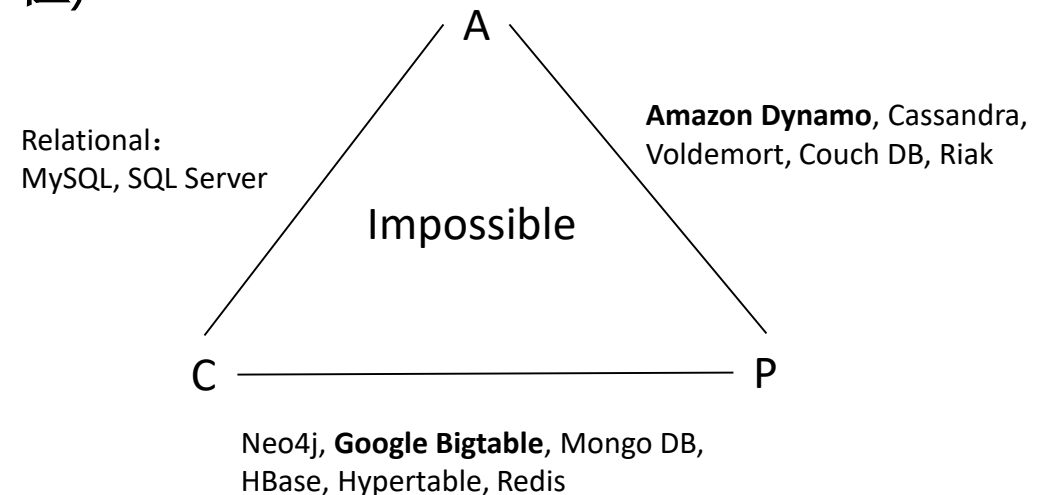
Eric A. Brewer



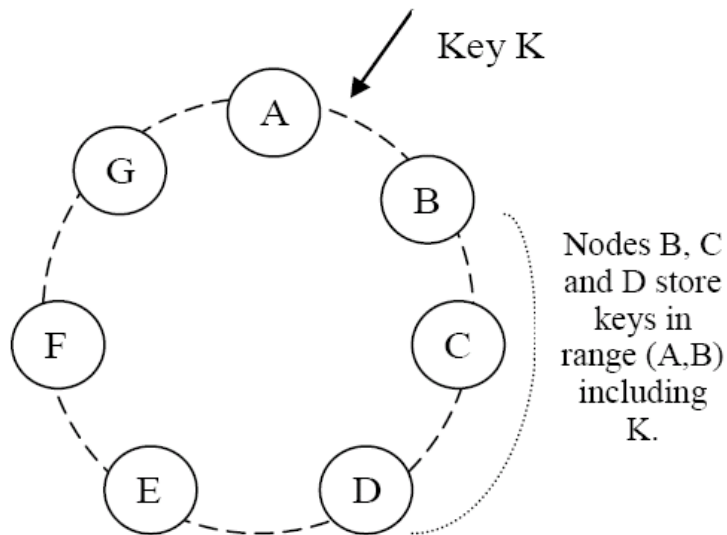
Eric Brewer at TNW Conference 2015

NoSQL介绍及其设计依据 – 帽子定理

- NoSQL (Not Only SQL): 非关系型的数据库。
- CAP ([Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services](#), Seth Gilbert and Nancy Lynch, Jun. 2002 ACM SIGACT News 卷次: 33 刊期: 2)
 - C: Consistency 一致性
 - A: Availability 可用性
 - P: Partition tolerance 分区容错性(可靠性)

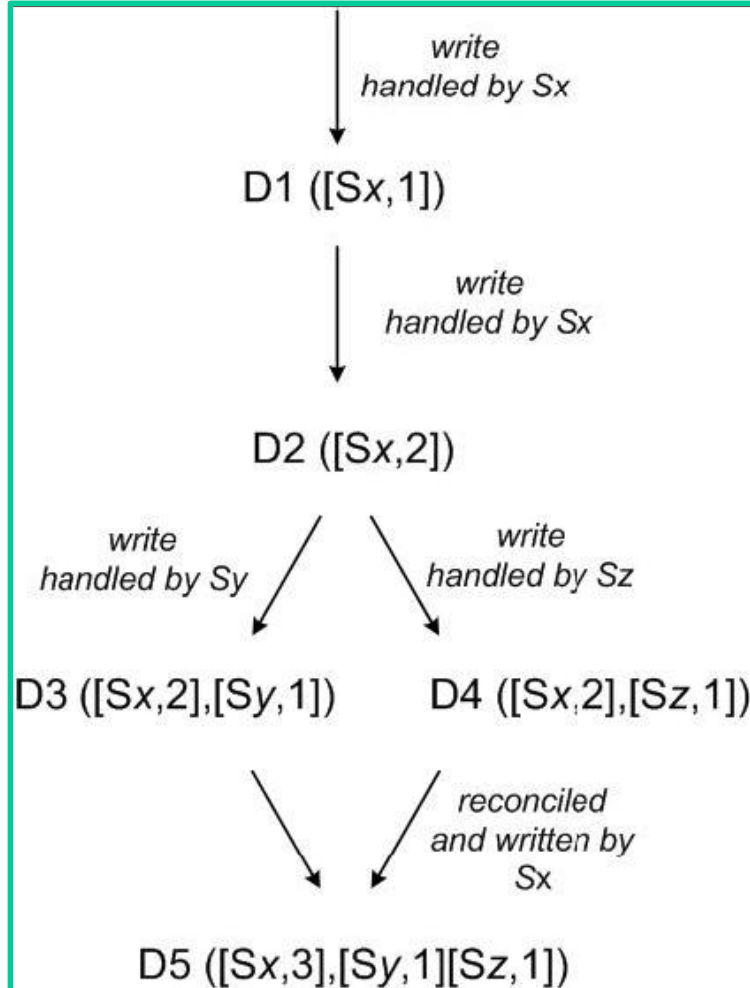


Dynamo及其变体



Partition Algorithm

- 1) Consistent hashing: the output range of a hash function is treated as a fixed circular space or “ring”.
- 2) “Virtual Nodes”: Each node can be responsible for more than one virtual node.



Vector Clock

A put() call may return to its caller before the update has been applied at all the replicas

A get() call may return many versions of the same object.

Challenge: an object having distinct version sub-histories, which the system will need to reconcile in the future.

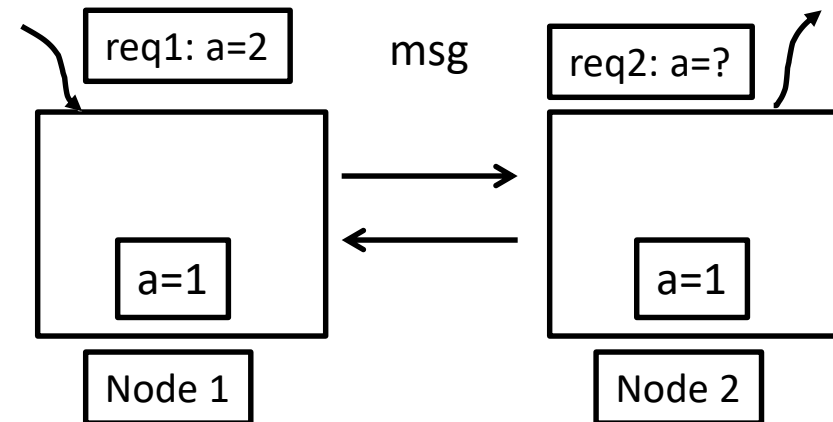
Solution: uses **vector clocks** in order to capture causality between different versions of the same object.

Data Versioning

Another Perspective: CAP Theorem (Eric Brewer, 2000)

- Consistency
 - all nodes see the same data at the same time
- Availability
 - every request to a non-failing node receives a response
- Partition tolerance
 - system allows arbitrary message loss
- All desirable but impossible to achieve all three

- **AC \rightarrow !P:** A (req2 returns), C (a=2) \rightarrow cannot lose msg
- **CP \rightarrow !A:** C(a=2), P (lose msg) \rightarrow req2 need to stall until msg arrives
- **AP \rightarrow !C:** A(req2 returns), P (lose msg) \rightarrow a=1 inconsistent



BASE模型(反ACID模型)

- ACID模型(高一致性+可用性, 很难进行分区)
- Atomicity
- Consistency
- Isolation
- Durability

- BASE模型(牺牲高一致性, 获得可用性+可靠性)
 - Basically Available(支持分区失败)
 - Soft state (状态可以有一段时间不同步, 异步)
 - Eventually consistent (最终数据是一致的就可以了, 而不是时时高一致)
- ‘BASE: An Acid Alternative’, *Dan Pritchett*, May. 2008, Queue 卷次: 6 刊期: 3

云计算技术的发展与云服务商业模式的诞生

- Google的云计算技术：2003-2006
 1. GFS, 2003 SOSP
 2. Map Reduce, 2004 OSDI
 3. Chubby, 2006 OSDI, 基于Paxos算法,
 4. BigTable, 2006 OSDI
- 亚马逊的云计算技术：2006-
 1. Dynamo, 2007 SOSP
 2. AWS (IaaS) : EC2 (虚拟机) , S3 (云存储)

2006年，当亚马逊第一次将其弹性计算能力作为云服务去售卖时，标志着云计算这种新的商业模式诞生。

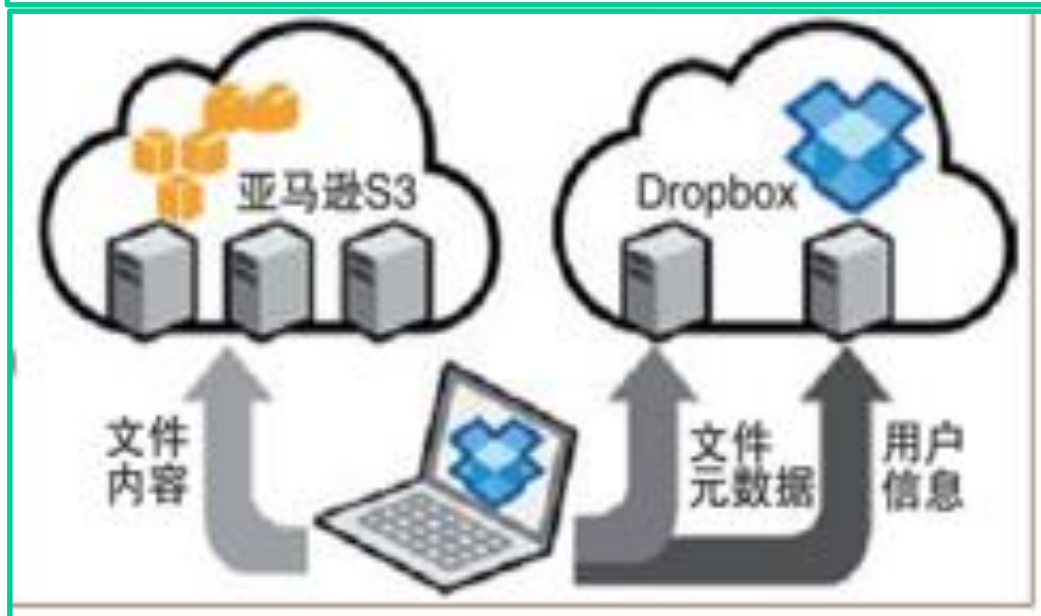
- 云计算的概念：来源，定义，特征
- 技术
 - 分割：虚拟化技术；虚拟化技术在云环境下的安全隐患
 - 聚合：Scale out（横向扩展）vs. Scale Up（纵向扩展）
- 产业发展：数据量增加；摩尔定律的终结；外包计算（成本的规模效应）；
- 聚合：Scale out（横向扩展）vs. Scale Up（纵向扩展）
 - 谷歌的贡献：云计算技术的先驱
 - 亚马逊：技术贡献；推动云计算的产业发展
 - 理论：CAP定理；MR计算模式
 - 开源系统：Hadoop；Spark

发展了大数据计算和存储的新理论，新思想，新的商业模式

大数据存储的安全隐私

云盘的诞生

2006年3月，亚马逊推出S3 (Simple Storage Service)，这是世界上第一个商业云存储基础设施，提供极为简单却极具扩展性的RESTful（自表义、无状态的）访问接口。然而，普通用户要使用S3是极为困难的，因为它连基本的图形界面和“文件（夹）”的概念都没有。



2007年，Dropbox公司一成立，就敏锐地抓住了S3的后端优势和前端不足，在S3和用户之间迅速架起了云存储服务的桥梁。

Dropbox自行搭建了一个规模较小的私有云，以维护那些重要的、敏感的元数据和用户信息。同时，Dropbox充分利用S3的云计算优势，轻松地应对了后来用户量和数据规模的多次“井喷”。

云盘，这种基于云存储技术的SaaS服务，在国内也为广大用户广泛接受。例如，大家常用的百度云盘。

- 云计算与传统的客户服务器架构的区别？
- 例如，云盘和传统的FTP服务有什么不同？
 - 相对于传统文件存储服务，云盘具有以下特点：
 - 1) 海量存储资源，虚拟为一个“云盘”；
 - 2) 云盘可以被视为是一个超大容量的免费网络U盘；可靠性高，数据永不丢失；
 - 3) 海量数据资源池，催生新的技术，例如，“秒传”功能；这些技术让用户获得了很好的用户体验。
- 云盘，改变了我们存放文件的习惯；但是，还有非常大的发展空间，例如，速度，安全性。这些问题解决了，我们也许就真的不需要移动硬盘了。

- 云存储系统可以使用户以较低廉的价格获得海量的存储能力，但高度集中的计算资源使云存储面临着严峻的安全挑战。据Gartner 2009年的调查结果显示因担心云数据隐私被侵犯70%受访企业的CEO拒绝大规模的采用云计算的计算模式。
- 而在最近几年里各大云运营商各自暴漏的安全存储问题引起了人们的广泛关注与担忧。
 - 如2011年3月谷歌Gmail邮箱出现故障，而这一故障造成大约 1 5 万用户的数据丢失。
 - 2012年8月国内云提供商盛大云因机房一台物理服务器磁盘发生故障导致客户的部分数据丢失。
 - 由此可见云中的数据安全存储已经阻碍了云计算在IT领域得到大规模的使用。
- 下面我们介绍云存储中数据完整性机制，隐私保护与安全防护机制。

数据完整性机制

- 随着云存储模式的出现，越来越多的用户选择将应用和数据移植到云中。但他们在本地可能并没有保存任何数据副本，无法确保存储在云中的数据是完整的。
- 那么，如何确保云存储环境下用户数据的完整性呢？

对于数据来说，养兵千日，用兵一时，用户在平时很难知道数据在云端的状态如何。尽管用户可以定时下载自己的数据以确认其完整性，但是当存储量越来越大的时候，这种方法就几乎不可行了。一方面，将数据完全下载下来需要很长的时间，占用很大的带宽；另一方面，这种数据传输业务是需要付费的，必须考虑其中的成本问题。

由于接入云的设备受计算资源的限制，用户不可能将大量的时间和精力花费在对远程节点的数据完整性检测上。通常云用户将完整性验证任务移交给经验丰富的第三方来完成。采用第三方验证时，**验证方只需要掌握少量的公开信息即可完成完整性验证任务。**

POR (Proofs of Retrievability)

安全威胁:

- 1) 外部入侵者有能力攻击云服务器并且损坏其中的数据, 却不被发现;
- 2) 云服务器在多数情况下是不会破坏数据的, 但是为了自身的利益, 也可能删除服务器中长时间不用的数据, 以此减轻负担和开支, 也有可能发现数据被外部入侵者损坏, 却对数据所有者隐瞒实情, 以此来维护自己的名誉。

- 针对以上现实中的需求, 研究者们提出了POR (Proofs of Retrievability) 的概念。
- POR的验证机制需要解决以下两个问题: 1) 更有效地识别外包文件中出现的损坏。2) 能恢复已损坏的数据文件。

基本的思路是通过增加验证信息来提高验证的效率, 并运用纠错编码技术来恢复被损坏的数据。具体而言, 针对第一个问题, 可以在外包的文件中预先植入一些称之为岗哨位 (Sentinel) 的检验数据块, 并在本地存储好这些检验数据块。对于远程服务器而言, 这些岗哨数据块与数据块是无法区分的; 倘若服务器损坏了数据文件中部分内容, 也会相应地损坏到岗哨文件块。对比存储在本地的检验数据, 能判断远程节点上的数据是否是完整的。另外通过岗哨块损坏的数目, 可以评估文件中出错的部分在整个文件中所占的比例。

针对第二个问题, 通常利用RS纠错码 (Reed-Solomon codes) 对文件进行容错预处理, 使用纠错机制可以恢复一部分损坏的数据。

云数据存储服务的架构

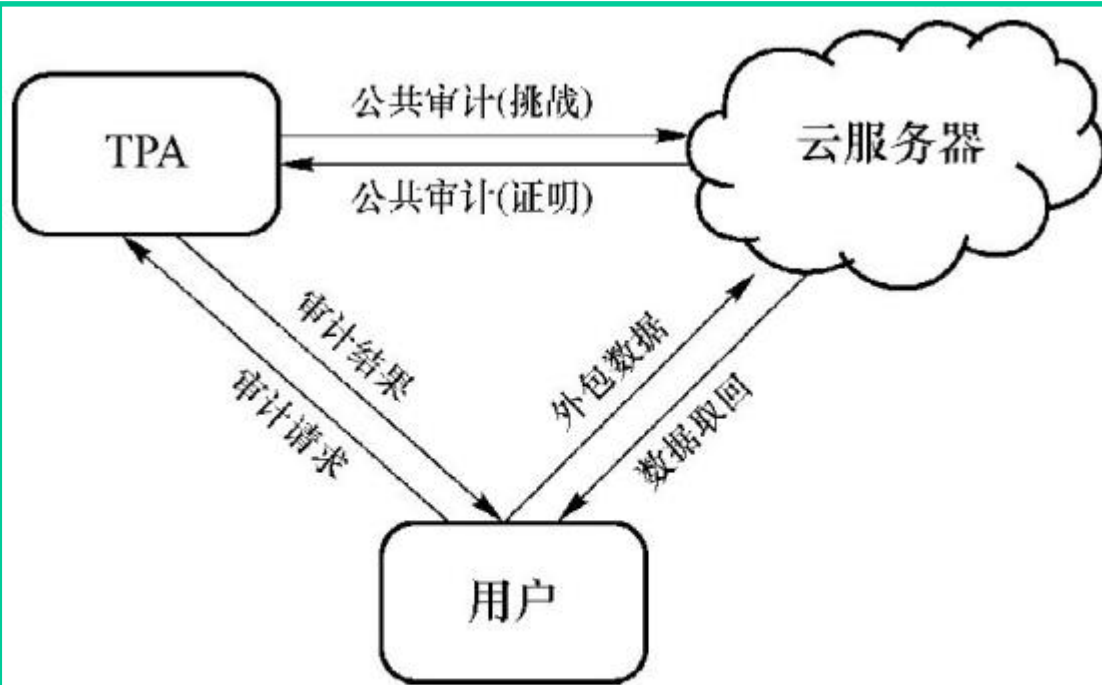


图 5-13 云数据存储服务架构

在这个架构中一共有三个角色：用户，云服务器(cloud server) 和第三方审计者(Third Party Auditor, TPA)

其中，TPA 的作用是代表数据所有者完成数据的完整性认证和审计任务等，这样用户就不需要亲自去做这些事，这对云计算的经济规模化是有价值的。

用户(可以是个人或企业) 就是希望利用云服务器来存储自己的大量数据，从而节省了建立本地存储基础设施的费用。

云服务器除了提供数据存储的服务，还可以提供可用性服务和分享服务。

由于数据所有者失去了对自己数据的直接物理控制，如何进行数据完整性的验证是十分关键的。考虑到自身的资源限制和云计算的经济规模化，数据所有者会把认证这项工作交给TPA来做，但同时又希望不会把数据隐私泄露给TPA；同时，人们希望TPA做的认证工作是有效率的，能够尽量地缩减计算开销和存储开销，尽量减少数据所有者的在线负担，比如密钥或MAC的更新。

隐私保护机制 - 数据去重

- 云盘的“秒传”功能，是如何实现的？
 - 该功能的基本原理是通过客户端软件从文件中获取一个特征值，然后在服务器上保存所有数据的特征值进行比较。如果有重复的，就无需再上传数据。由于很多电影、音乐类的文件，已经有用户在云存储系统中保存，因此，只需要确认一下该文件存在即可。这种设计不仅提高了用户存储文件的效率，而且同一个文件的无须再重复存储，避免了存储空间的浪费。
- 云盘的“秒传”功能给用户带来了很好的用户体验，并且降低了服务提供商的存储成本。
- 目前数据压缩非常有效也是很常用的一个手段是去重（deduplication），即识别数据中冗余的数据块，只存储一份，其余位置存储类似指针的数据结构。研究表明，基于数据分布的不同，**有效的去重能够节省高达50%甚至90%的存储空间和带宽。**去重已经被广泛用于很多商业化的系统如Dropbox。

加密数据去重技术

- 大规模云存储系统往往面临两个矛盾的需求：一方面系统需要压缩数据以节省存储空间的开销；另一方面，用户出于数据安全和隐私的考虑，希望自己的数据加密存储。但是去重却是和数据加密的目标直接相矛盾的。
 - 加密之后的密文需要保留原文的冗余，即原文相同的数据块加密后的密文仍相同（这里的相同不一定是密文的全等，系统只要一种识别包含相同内容的密文的手段即可），这样去重才能够起作用。但是，它与加密算法的安全性定义有不可调和的矛盾。
 - Semantic security **明确禁止原文相等性的检测**，即给定两个密文，不应该能够允许对手断定它们加密的是否是同样的数据，否则对手可以利用这一性质攻破前述IND。可以明确断言的是，满足现代加密算法安全性（如semantic security）的所有加密算法都不支持去重。
- 于是退而求其次，即，我们可以适度放宽对安全性的要求，允许密文泄露原文相等性信息，从而使加密后的去重成为可行。

Convergent Encryption (CE) - 收敛加密

- 最早提出的方案是Convergent Encryption (CE) 。
 - 它的想法非常简单：一个数据块 d 的加密如下： $E(h(d), d)$ ，其中 $E(key, d)$ 是以 key 做密钥加密数据 d 的对称加密算法， $h(x)$ 是一个hash function。也就是说， 当需要加密一个数据块 d 的时候，CE先用数据内容生成 key ，再用一个symmetric encryption算法（如AES等）加密。
 - 严格地讲，对称加密算法本身通常都是randomized或者stateful，即除了密钥之外，算法本身会生成一些随机数（例如Initialization vector或IV），或者维护一个计数器之类的状态，这样即使多次加密同样的信息也会有不同的结果，目的还是为了获得类似semantic security这样的安全性。
 - 这里CE的做法可以理解为 $h(x)$ 输出的一部分作为 key ，另外一部分作为算法所需的随机数（如IV）。这样做的结果是，不管是哪个用户加密，同样的数据块一定会被加密成同样的密文，后续可以做去重了。

加密数据去重的理论

- 但是一个令密码学研究者不安的状况是，虽然CE已经被广泛应用，它的安全性却始终没有严格的分析。它显然没有达到semantic security。那么它到底提供一种什么样的保护呢？这种保护是否足够？是否存在很容易的破解方法使得它完全失去作用？在没有解决这些问题的情况下就广泛使用它显然是令人忐忑的。
- 2013年，Mihir Bellare等人提出了Message-Locked Encryption (MLE) 的框架。他们同时提出了PRV\$-CDA的安全性概念，并证明了PRV\$-CDA比其他相关的安全性都更强。
 - 简单地讲，MLE是这样一种加密算法，它使用的key是从待加密的原文算出来的 (key used for encryption is derived from the message itself) 。
 - CE是MLE的一个特例。MLE允许原文相等信息的判断 (equality checking) ，从而支持去重。

\$通常表示随机数据或因素。在PRV\$-CDA的例子中，CDA代表chosen-distribution attack，PRV\$代表与随机数的不可区分性。简单讲，PRV\$-CDA意味着对手不能够将密文同与密文同样长度的随机数区分开来。

在MLE的框架下，CE被证明满足PRV\$-CDA安全性。

拥有权证明 (PoW, proofs-of-ownership)

在“秒传”设计中，用户首先发送数据的一个标签到云服务器，云服务器根据此标签进行冗余检查，这样用户不必上传每一个数据到云存储服务器，既节约了存储空间，也节约了上传带宽。但是，不幸的是攻击者很容易通过一个文件的哈希值获取整个文件，这类攻击的根本原因是一个很简单的文件哈希值就可以代表整个文件。

为了解决此安全问题，HALEVI等人首先提出了一个拥有权证明 (PoW) 的模型。具体来说，**PoW就是在服务器和客户端之间执行一个挑战/响应的协议**，它能够有效地预防攻击者通过单一哈希值去获取整个文件。在该模型中他们提出了一个基于文件级的去重方案，主要使用纠错码对文件进行编码，同时利用Merkle哈希树方法进行文件拥有权证明。

当数据块很多时，构造的Merkle哈希树高度很大，不利于计算和验证效率。为了进一步提高验证效率，DI等人提出了一个高效的拥有权证明方案，命名为s-PoW，该方案**通过随机选择一些比特位作为文件拥有权证明证据，这只需要一个常量的计算开销。**

- 大数据的处理（计算）与存储
- 大数据存储的安全隐私
 - 数据完整性机制：POR
 - 隐私保护机制（加密数据去重技术）：CE, MLE
 - 安全防护机制：PoW
- 保护隐私的计算
 - 1) SGX;
 - 2) 同态加密；安全多方计算



北京邮电大学

Beijing University of Posts and Telecommunications

感谢聆听！
