# 第一章作业

## 2.新摩尔定律的含义是什么？

芯片性能的增长不再只依赖于传统的晶体管数量翻倍（摩尔定律），而更多依赖于架构创新、能效改进、异构计算等新技术手段推动计算能力的提升。

## 3.云计算有哪些特点？

超大规模、虚拟化、高可靠性、通用性、高可伸缩性、按需服务、廉价

## 4.云计算按照服务类型可以分为哪几类？

云计算按服务类型大致分为三类：

* 基础设施即服务（IaaS）
* 平台即服务（PaaS）
* 软件即服务（SaaS）

## 5.云计算技术体系结构可以分为哪几层？

云计算技术体系结构分为以下几层：

* 物理资源层（计算机、存储器、网络设施、数据库、软件等）
* 资源池层（将大量同构资源构成资源池）
* 管理中间件层（资源管理、任务管理、安全管理、用户管理等）
* SOA构建层（面向服务架构的构建层，功能依赖外部设施提供）

# 第二章作业

## 1．Google云计算技术包括哪些内容？

Google文件系统（GFS）

分布式数据处理框架MapReduce

分布式锁服务Chubby

分布式结构化数据表Bigtable

分布式存储系统Megastore

大规模分布式系统的监控基础架构Dapper

海量数据的交互式分析工具Dremel

内存大数据分析系统PowerDrill

Google应用程序引擎

## 2．当前主流分布式文件系统有哪些？各有什么优缺点？

当前主流的分布式文件系统主要包括：

1. **HDFS (Hadoop Distributed File System)**
2. **Ceph**
3. **GlusterFS**
4. **Lustre**
5. **MooseFS (MFS)**

### 1. HDFS (Hadoop Distributed File System)

**特点：** Apache Hadoop生态系统的核心组件，为Hadoop生态圈中的其他项目（如MapReduce、Spark）提供底层存储。

**优点：**

* **高吞吐量：** 专为大文件读写优化，支持高吞吐量的数据访问，非常适合批处理应用。
* **高容错性：** 通过多副本机制（默认3副本）实现数据冗余，节点故障时数据不会丢失，并能自动恢复。
* **可扩展性：** 可以通过增加DataNode来水平扩展存储容量和处理能力。
* **成本效益：** 可以部署在廉价的商用硬件上。
* **生态系统完善：** 拥有庞大的社区支持和丰富的周边工具。

**缺点：**

* **不适合低延迟访问：** HDFS为高吞吐量设计，不适合毫秒级的低延迟数据访问，例如随机读写小文件效率低。
* **不适合存储大量小文件：** 每个文件和目录的元数据都由NameNode管理，大量小文件会导致NameNode内存压力过大。
* **不支持随机写入：** 文件一旦创建就不能修改，只支持追加写入。
* **单点故障（NameNode）：** 早期版本NameNode存在单点故障问题，虽然HDFS High Availability (HA) 解决了这个问题，但配置和管理更复杂。

### 2. Ceph

**特点：** 一个统一的分布式存储系统，提供对象存储（RADOS Gateway）、块存储（RBD）和文件系统（CephFS）三种接口。

**优点：**

* **统一存储：** 能够同时提供对象、块和文件存储服务，功能全面。
* **高可扩展性：** 无单点故障，可以轻松扩展到PB级甚至EB级存储。
* **高可靠性：** 采用CRUSH算法管理数据分布和副本，数据自动修复和均衡，具有极高的容错能力。
* **高性能：** 通过智能数据分布和客户端缓存实现高性能。
* **自管理、自修复：** 集群具有自我修复和自我管理能力，降低运维成本。

**缺点：**

* **复杂性高：** 部署和管理相对复杂，学习曲线较陡峭。
* **资源消耗：** 对硬件资源（尤其是内存和网络）要求较高。
* **性能调优：** 性能调优需要深入理解其内部机制。

### 3. GlusterFS

**特点：** 开源的，横向扩展的分布式文件系统，通过软件的方式将多台服务器的存储空间聚合起来。

**优点：**

* **简单易用：** 部署和管理相对简单，不需要额外的元数据服务器。
* **高扩展性：** 通过添加服务器即可扩展存储容量。
* **高可用性：** 支持多种复制模式（如NFS、Erasure Code）确保数据高可用。
* **灵活性：** 支持多种卷类型（如分布式卷、复制卷、条带卷、分布式复制卷等），可以根据需求选择。
* **无单点故障：** 没有中央元数据服务器，消除了单点故障。

**缺点：**

* **元数据管理：** 元数据分散存储，对于大量小文件的元数据操作性能可能受限。
* **性能：** 在某些特定场景下，其性能可能不如HDFS或Ceph，尤其是在大量随机读写操作时。
* **不适合特定场景：** 对于需要高一致性或复杂事务的场景可能不是最佳选择。

### 4. Lustre

**特点：** 高性能并行文件系统，主要应用于高性能计算（HPC）领域。

**优点：**

* **极致高性能：** 专为大规模并行读写优化，能够提供数TB/s的带宽，非常适合科学计算、数据分析等对I/O性能要求极高的场景。
* **高可扩展性：** 可以支持数十万个客户端和EB级存储。
* **POSIX兼容：** 提供标准的POSIX文件系统接口。

**缺点：**

* **复杂性高：** 部署、配置和管理非常复杂，需要专业的知识和经验。
* **成本高：** 通常需要高性能硬件支持，部署成本较高。
* **通用性差：** 主要面向HPC领域，不适合作为通用文件系统。
* **元数据瓶颈：** 元数据服务器（MDS）可能成为性能瓶颈，尽管可以通过多MDS来缓解。

### 5. MooseFS (MFS)

**特点：** 轻量级、高可用、高性能的分布式文件系统。

**优点：**

* **简单易用：** 部署和管理相对简单，轻量级。
* **高可用性：** 通过Master/Chunk Server/Client架构和多副本机制实现高可用。
* **高可扩展性：** 可以轻松扩展存储容量。
* **数据可靠性：** 支持多副本、垃圾回收、数据修复等功能。
* **POSIX兼容：** 客户端通过FUSE（Filesystem in Userspace）挂载，支持POSIX接口。
* **文件操作灵活：** 支持随机读写、追加写入等。

**缺点：**

* **Master单点故障（早期版本）：** 早期版本Master存在单点故障，虽然可以通过Master高可用方案解决，但增加了复杂性。
* **社区规模：** 相对于HDFS和Ceph，社区规模较小。
* **性能瓶颈：** 在超大规模集群或某些极端I/O场景下，性能可能不如HPC领域的专业文件系统。

## 3．GFS采用了哪些容错措施来确保整个系统的可靠性？

* **Master容错**：
  + Master会周期性地设置检查点（checkpoint），并导出Master的数据，包括命名空间（文件系统的目录结构）、Chunk与文件名的映射表、Chunk副本的位置信息。
  + 当Master发生故障时，在磁盘数据保存完好的情况下，可以迅速恢复以上元数据。
  + GFS还提供了Master远程的实时备份，以防止Master彻底死机的情况。
* **Chunk Server容错**：
  + GFS采用副本的方式实现Chunk Server的容错，每个Chunk默认有三个存储副本。
  + 对于每一个Chunk，必须将所有的副本全部写入成功，才视为成功写入。
  + 当相关的副本出现丢失或不可恢复等情况时，Master会自动将该副本复制到其他Chunk Server。
  + 每个Chunk以Block为单位进行划分（默认大小64MB，Block大小64KB），每个Block对应一个32bit的校验和，用于数据完整性检查。

## 4．MapReduce与传统的分布式程序设计相比有何优点？

MapReduce与传统的分布式程序设计相比，主要优点在于：

* **封装了并行处理、容错处理、本地化计算、负载均衡等细节**：这极大地简化了分布式程序的开发，开发者无需关注这些复杂的底层细节。
* **提供了一个简单而强大的接口**：使得大规模数据处理变得更加易于实现。
* **实现任务的可靠执行与容错机制**：通过主节点管理下的各分节点共同完成任务，并在此过程中实现容错。

## 6．阐述Bigtable的数据模型和系统架构。

### Bigtable的数据模型

Bigtable被设计为一个**分布式多维映射表**，其数据通过以下三个维度进行索引：

* **行关键字（RowKey）**：
  + 可以是任意的字符串，但大小不能超过64KB。
  + 表中所有数据都根据行关键字进行排序，使用词典序。
  + 这种排序方式使得同一地址域的网页等相关数据可以存储在表中的连续位置，有利于数据压缩和读取效率。
* **列关键字（ColumnKey）**：
  + 由\*\*族名（Column Family）**和**限定词（Qualifier）\*\*组成。
  + **列族**：用于组织相关的数据，族名必须有意义。列族同时也是Bigtable中访问控制（Access Control）的基本单元。
  + **限定词**：可以任意选定，用于区分列族内的具体数据项。
  + 这种组织方式使得数据结构清晰明了，含义清楚。
* **时间戳（TimeStamp）**：
  + 是一个64位整型数，具体的赋值方式可以由用户自行定义。
  + 用于区分同一行、同一列下不同版本的数据。例如，Google的网页检索和用户个性化设置等服务需要保存不同时间的数据版本，时间戳就是用来区分这些版本的。

**存储逻辑可以表示为： (row:string, column:string, time:int64) → string**

这意味着，通过一个行关键字、一个列关键字和一个时间戳，可以唯一确定一个存储在Bigtable中的字符串值。

### Bigtable的系统架构

Bigtable的整体架构主要由以下组件构成：

* **Bigtable客户端程序库**：
  + 位于客户端，应用程序通过调用该库中的相关函数与Bigtable进行交互。
* **主服务器（Master Server）**：
  + 负责整个Bigtable集群的管理和协调工作。
  + **故障处理及监控**：监控子表服务器的状态，及时检测服务器的加入或撤销。
  + **执行元数据操作及负载均衡**：当新的子表产生时，主服务器会将其分配给空间足够的子表服务器，确保负载均衡。
  + **与Chubby的交互**：从Chubby中获取一个独占锁，确保同一时间只有一个主服务器。扫描Chubby中的服务器目录，发现活跃的子表服务器。与活跃子表服务器联系以了解子表分配情况。扫描元数据表，发现未分配的子表并进行分配。
* **子表服务器（Tablet Server）**：
  + 负责处理实际的数据读写操作。
  + **保存子表数据及日志**：每个子表由多个SSTable（Google为Bigtable设计的内部数据存储格式）及其日志文件构成，不同子表的SSTable可以共享。每个子表服务器上仅保存一个日志文件。
  + **数据存储**：较新的数据存储在内存中的\*\*内存表（Memtable）\*\*里，较早的数据则以SSTable格式保存在GFS中。
  + **子表分裂**：较大子表的分裂由子表服务器发起完成，完成后会通知主服务器。
* **Chubby**：
  + Google的分布式锁服务，在Bigtable架构中扮演关键角色。
  + **作用一**：选取并保证同一时间内只有一个主服务器。
  + **作用二**：获取子表的位置信息。
  + **作用三**：保存Bigtable的模式信息及访问控制列表。
* **GFS (Google File System)**：
  + 作为Bigtable的底层存储系统，所有的SSTable文件都存储在GFS上。
* **Google WorkQueue**：
  + 未在讲义中详细说明其具体作用，但作为架构的一部分，可能用于任务调度或队列管理。

**Bigtable系统内部采用了一种类似B+树的三层查询体系**，用于子表地址的组成和查询。

**数据存储和读/写操作流程**：

* **写操作**：数据首先写入内存表（Memtable），同时写入子表日志。
* **读操作**：优先从内存表读取，如果内存表中没有，则从SSTable文件（GFS上）读取。

这种架构设计使得Bigtable能够满足海量数据的存储和高并发的服务请求，并且具有高可用性、可扩展性和简单性等特点。

## 7．分布式存储系统Megastore的核心技术是什么？

分布式存储系统Megastore的核心技术是：

**通过结合传统数据库的同步复制技术和NoSQL数据库的异步复制技术，来满足海量数据的存储和高并发的服务请求。**

具体来说，Megastore通过以下方式实现这一结合：

* **同步复制**：用于确保数据的一致性和可靠性，通常在主副本和少数备份副本之间进行，以保证事务的原子性、一致性、隔离性和持久性（ACID特性）。
* **异步复制**：用于在更广泛的地理分布范围内进行数据复制，以提高读取性能和可用性，同时避免同步复制可能带来的延迟问题。

通过这种混合复制策略，Megastore旨在提供兼具传统关系型数据库的数据一致性和事务支持，以及NoSQL数据库在大规模分布式环境下的可伸缩性和高可用性的存储解决方案。

## 9．相比于行存储，列存储有哪些优点？

相比于行存储，列存储具有以下优点：

* **节省I/O**：
  + 在行存储中，如果要查询某个特定的属性，需要读取整行数据，即使只需要其中一列。
  + 在列存储中，如果查询只涉及少量列，则只需要读取这几列的数据，从而大大减少了I/O开销。
* **压缩率更高**：
  + 在行存储中，每一行的数据类型可能不同，难以进行有效的压缩。
  + 在列存储中，同一列的数据类型相同，且通常具有较高的数据相似度（例如，一列存储的都是年龄，数值范围有限），这使得数据压缩更加高效，从而节省存储空间。
* **更好的索引**：
  + 列存储可以为每一列单独建立索引，这对于只查询部分列的场景非常有利，可以显著提高查询速度。

## 10．为什么MapReduce不适合实时数据处理？

MapReduce不适合实时数据处理的原因是：

* **无法满足交互式查询的延时要求**：MapReduce的设计目标是批处理大规模数据，其处理模式通常是离线和非实时的。它会经历任务调度、数据分发、计算和结果收集等多个阶段，这些阶段会引入显著的延迟。对于需要亚秒级或秒级响应的实时查询场景，MapReduce的批处理特性无法满足其低延迟的要求。

## 13．Google App Engine有哪些不同的开发环境？各自有什么特点？

Google App Engine提供了以下两种不同的开发环境：

* **本地开发环境**：
  + **特点**：提供了一个模拟App Engine生产环境的开发服务器。开发者可以在本地进行应用程序的开发、测试和调试，而无需部署到真实的App Engine服务器上。这大大加快了开发周期，降低了开发成本。
* **在线部署环境（实际的App Engine生产环境）**：
  + **特点**：开发者可以将应用程序部署到Google App Engine的服务器上。这意味着应用程序将运行在Google的基础设施上，享受其提供的可扩展性、可靠性、负载均衡等服务。同时，开发者也需要遵循App Engine的配额限制和计费模式。

# 第三章作业

## 1．在Dynamo中添加一个新的节点时，原先各节点保存的数据是否需要改变？如果改变，应该如何变化？

添加新节点时，由于Dynamo采用改进的一致性哈希算法，只会影响到新节点的前驱节点，不会对其他节点产生大范围影响。具体变化包括：

* 通过引入虚拟节点的概念，可以减小数据分布不均衡的可能性
* 添加节点时，只会引起较小的数据传输
* 新节点会被分配到哈希环的特定位置，并可能接管部分数据分区

## 2．Merkle哈希树的创建需要较大的时间开销。频繁地重建Merkle树会对系统造成很大的负担。假设Merkle树的叶子节点表示的是数据分区的Hash值，请设计一个Merkle树重建方案，尽量减少Merkle树的重建工作。

重建方案：

1. 增量更新：不是每次都全量重建Merkle树，而是只对发生变化的数据分区进行局部更新
2. 缓存部分计算结果：对于未发生变化的子树，直接复用之前的哈希值
3. 设置更新阈值：只有当数据变化超过一定比例时才触发Merkle树重建
4. 并行计算：对不同数据分区的哈希值计算可以并行进行，减少总体重建时间

## 3．私有IP、公有IP和弹性IP的区别在哪里？

* 私有IP地址：由动态主机配置协议（DHCP）分配产生，用于内部网络通信
* 公共IP地址：动态分配，用于外部网络通信
* 弹性IP地址：
  1. 与用户账号绑定，而不是与特定实例绑定
  2. 可以在实例故障时快速重新映射
  3. 需要手动关联到指定实例

## 5．简单存储服务S3与传统的文件系统有哪些区别？

1. 存储结构不同

* 传统文件系统：层级目录结构，使用文件夹嵌套
* S3：桶（Bucket）结构，桶不可以被嵌套，类似于"容器"

1. 命名规则

* 传统文件系统：文件名通常无严格限制
* S3：桶名要求全局唯一，需符合DNS命名规则

1. 数据一致性模型

* 传统文件系统：通常提供强一致性
* S3：采用最终一致性模型，存在数据同步延迟
  + 写入新对象后立即读取可能返回"键不存在"
  + 替换对象后立即读取可能返回原有数据
  + 删除对象后可能仍然可以读取或列出已删除对象

1. 操作类型

* 传统文件系统：读取、写入、修改、删除
* S3：提供Get、Put、List、Delete、Head五种基本操作

1. 安全机制

* 传统文件系统：通常依赖操作系统权限
* S3：提供更复杂的安全措施
  + 身份认证（Authentication）
  + 访问控制列表（ACL）
  + 多级权限控制（READ、WRITE、FULL\_CONTROL等）

1. 元数据管理

* 传统文件系统：元数据相对简单
* S3：提供更丰富的元数据，包括：
  + last-modified（最后修改时间）
  + ETag（MD5哈希值）
  + Content-Type（MIME类型）
  + Content-Length（数据长度）

1. 可扩展性

* 传统文件系统：扩展性有限
* S3：具有高度可扩展性，理论上可以存储海量数据

## 6．简单阐述SQS在Amazon云计算中的作用。

SQS（Simple Queue Service）的主要作用：

* 消息队列服务，实现分布式系统中的异步通信
* 解耦系统组件，提高系统的可扩展性和容错性
* 提供消息传递的缓冲机制
* 支持多生产者和多消费者模式
* 保证消息至少被传递一次
* 提供消息可见性超时机制，防止多个消费者同时处理同一消息

关键特点：

* 保证消息按照FIFO（先进先出）或无序方式传递
* 支持标准队列和FIFO队列
* 可设置消息保留时间（最长14天）
* 自动扩展，无需管理底层基础设施

## 7．如何理解传统数据库在可扩展性方面的能力较弱？

传统数据库可扩展性弱的主要原因：

* 垂直扩展（Scale Up）局限性
  1. 单一服务器硬件资源有限
  2. 增加硬件成本高昂
  3. 存在物理和技术上的性能瓶颈
* 数据一致性要求严格
  1. ACID事务模型导致扩展复杂
  2. 强一致性降低了系统的并发性能
  3. 复杂的关联查询和联表操作难以分布式处理
* 数据分片和水平扩展困难
  1. 跨节点数据一致性维护复杂
  2. 关联查询和事务处理变得困难
  3. 数据重平衡和迁移成本高
* 性能瓶颈
  1. 单点写入（写锁）限制了并发能力
  2. 复杂的索引和约束增加系统负担
  3. 大数据量时查询性能急剧下降

## 8．非关系型数据库是如何解决可扩展性问题的？

非关系型数据库解决可扩展性的策略：

1. 弱一致性模型

* 采用最终一致性
* 牺牲部分实时性换取高可用性和可扩展性
* 异步同步数据，减少同步开销

1. 水平扩展（Scale Out）

* 通过增加服务器数量提高系统性能
* 使用分片（Sharding）技术
* 数据自动分布在多个节点

1. 去中心化设计

* 没有单点故障
* 节点可动态添加或移除
* 数据自动复制和均衡

1. 简化数据模型

* 放弃复杂的关联查询
* 去除外键和复杂约束
* 使用键-值（Key-Value）或文档模型

1. 分区容错性

* CAP理论中优先保证分区容错性
* 允许节点部分失效
* 快速故障恢复

1. 无共享架构

* 每个节点相对独立
* 节点间通过消息传递协作
* 减少资源竞争

具体实现技术：

* 一致性哈希
* 虚拟节点
* 数据复制
* 分布式共识算法

# 第四章作业

## 1．微软云计算平台包含几部分？每部分的作用是什么？

微软云计算平台（Microsoft Azure）主要包含三大部分：

1. 计算服务（Compute Services）  
   作用：

* 提供虚拟机（Virtual Machines）计算资源
* 支持不同类型的应用程序和工作负载
* 包括云服务、容器服务和无服务器计算
* 灵活支持Windows和Linux操作系统
* 可根据需求弹性扩展计算资源

1. 存储服务（Storage Services）  
   作用：

* 提供多种数据存储解决方案
* Blob存储：存储非结构化数据
* 表存储：存储NoSQL结构化数据
* 文件存储：提供文件共享服务
* 队列存储：支持消息传递
* 磁盘存储：为虚拟机提供持久化存储

1. 网络服务（Network Services）  
   作用：

* 提供网络连接和安全性管理
* 虚拟网络：创建私有网络环境
* 负载均衡：分发网络流量
* 网络安全组：控制网络访问
* VPN网关：建立安全远程连接
* 流量管理：优化网络性能和可用性

## 2．Microsoft Azure存储服务提供了几种类型的存储方式？阐述每种存储方式主要的存储对象。

Microsoft Azure存储服务提供四种主要的存储方式：

1. Blob存储（Blob Storage）  
   主要存储对象：

* 非结构化数据
* 文档、图像、视频、音频文件
* 日志文件
* 备份和归档数据
* 支持三种Blob类型：  
  a) 块Blob：适用于文本和二进制文件  
  b) 页Blob：适用于随机读写操作  
  c) 追加Blob：适用于日志记录

1. 表存储（Table Storage）  
   主要存储对象：

* NoSQL结构化数据
* 键-值对数据
* 半结构化数据
* 快速查询和扩展的数据集
* 适合存储元数据、用户配置信息

1. 文件存储（File Storage）  
   主要存储对象：

* 文件共享服务
* 支持标准文件系统协议
* 企业级文件共享
* 支持SMB（服务器消息块）协议
* 可作为网络文件系统使用

1. 队列存储（Queue Storage）  
   主要存储对象：

* 消息队列
* 异步通信机制
* 分布式系统组件解耦
* 支持消息传递和通信
* 最大消息大小64KB

## 3．阐述Web Role实例和Worker Role实例之间的通信机制。

Web Role实例和Worker Role实例通过Fabric控制器进行通信。具体来说：

* Fabric控制器是一个分布式应用，控制数据中心内的机器集合和运行在这些机器上的软件
* 每个Role实例都有Fabric代理，通过Fabric代理与Fabric控制器进行交互
* Fabric控制器负责：
  1. 控制所有运行的应用
  2. 根据配置信息决定应用运行位置
  3. 选择物理服务器以优化硬件使用
  4. 决定并创建虚拟机（VMs）数量
  5. 监控虚拟机运行状态

## 4．SQL Azure数据同步技术主要有几种？分别如何实现？

SQL Azure数据同步技术主要有两种实现方式：

1. 本地SQL Server与SQL Azure数据库同步
2. 不同微软数据中心之间的SQL Azure数据库同步

关键特点：

* 采用"轮辐式（hub-and-spoke）"模型
* 所有变化首先复制到SQL Azure数据库"hub"
* 然后传送到其他"spoke"数据库
* 可以同步整个数据库，也可以只同步有更新的数据库表格

目的：

* 提高数据访问性能
* 确保网络故障时应用仍能访问数据库
* 防止数据丢失

## 5．阐述SQL Azure和SQL Server的相同点和不同点。

相同点：

* 都提供关系型数据库服务
* 支持Transact-SQL（T-SQL）
* 都使用标准SQL Server客户端和工具

不同点：

1. 管理层面

* SQL Azure：微软全面管理物理资源，用户无法直接管理底层系统
* SQL Server：用户可以完全管理物理和逻辑资源

1. 服务特点

* SQL Azure：
  + 自动数据复制，提供高可用性
  + 自动负载均衡和故障转移
  + 云端部署，无需准备硬件
* SQL Server：需要自行管理这些功能

1. 功能限制

* SQL Azure不支持SQL Server的所有特征和数据类型
* SQL Azure锁定物理资源操作命令
* 不能使用SQL Server的备份机制

## 6．Service Fabric的体系架构及各个子系统功能？

Service Fabric体系架构由多个分层子系统组成，包括：

1. 传输子系统

* 功能：实现点对点数据报信道
* 支持Service Fabric群集内部以及群集与客户端的通信

1. 联合子系统

* 基于传输子系统的通信基元
* 将各个节点拼结为统一群集
* 为其他子系统提供系统基元：
  + 失败检测
  + 群首推举
  + 一致性路由

1. 可靠性子系统

* 确保Service Fabric服务状态高度可用
* 包含三个关键组件：
  + 复制器：确保主副本和辅助副本状态一致性
  + 故障转移器：确保故障情况下的可用性
  + Resource Manager：确保故障转移单元可用性，平衡节点基础共享池中的服务资源

1. 管理子系统

* 提供端到端服务和应用程序生命周期管理
* 实现：
  + 预配
  + 部署
  + 修补
  + 升级
  + 取消预配应用程序

1. 宿主子系统

* 为特定节点服务
* 确保副本正确放置且正常运行

1. 通信子系统

* 通过命名服务提供群集内部的可靠消息传送
* 提供服务发现功能

1. 可测试性子系统

* 为测试服务而设计的工具
* 包括错误注入、产品测试等功能

# 第五章作业

## 1．简述Hadoop 1.0、Hadoop 2.0、Hadoop3.0的优缺点，思考为什么要升级。

**Hadoop 1.0**

* **优点**
  + 实现了分布式文件系统HDFS和分布式计算模型MapReduce。
  + 能够在廉价硬件上存储和处理PB级大数据，实现高容错和可靠性。
* **缺点**
  + 资源管理只支持MapReduce，对其它计算框架支持差。
  + 单一JobTracker架构易成瓶颈，不易扩展，单点故障明显。
  + 不支持多租户或细粒度资源管理。
* **为何升级**
  + 为突破计算调度瓶颈，实现多计算类型、多用户高效资源分配。

**Hadoop 2.0**

* **优点**
  + 引入YARN（Yet Another Resource Negotiator）资源管理框架，实现资源与计算分离。
  + 可支持非MapReduce应用（如Spark、Tez），极大扩展应用场景。
  + 解决JobTracker瓶颈，提升扩展性与容错性。
* **缺点**
  + 虽资源调度改善，但NameNode仍为单点故障，可靠性有限制。
  + 应用开发复杂度增大，需适应YARN接口。
* **为何升级**
  + 满足多样化分布式数据处理需求，进一步提升大规模集群资源利用率和高可用性。

**Hadoop 3.0**

* **优点**
  + 支持多NameNode（活跃/备用不限于双节点），大幅提升容错性。
  + HDFS支持纠删码，降低冷数据存储开销（空间缩减至1.4倍，副本方式为3倍）。
  + 提升MapReduce本地优化，洗牌性能最高提升30%。
  + 支持对接云对象存储如Azure、阿里云OSS。
  + DataNode内部支持磁盘均衡，解决单节点多盘数据不均问题。
* **缺点**
  + 新特性需要更复杂的配置和运维，升级带来旧系统兼容性风险。
  + 跨版本迁移、调优、监控运维要求更高。
* **为何升级**
  + 适应大规模数据、“冷数据”与“热数据”混合存储趋势，支持混合云架构，强化高可用与极致扩展性。

## 2．简述Hadoop生态圈产品区别与联系，思考为什么出现这些产品。

**联系：**

* 都以Hadoop（HDFS、YARN、MapReduce）为核心，解决分布式存储与计算。
* 通过“分而治之”，实现数据存储、资源管理、任务调度、应用开发全链条覆盖。
* 各产品之间通过统一的访问协议（Java API、REST、命令行等）打通，实现互操作。

**区别：**

* **分布式存储**：HDFS、HBase（NoSQL）、Ozone（对象存储）。
* **资源/作业管理**：YARN，ZooKeeper（协调服务）、Oozie（工作流调度）、Ambari（集群管理）。
* **数据交换/传输**：Flume（日志采集）、Sqoop（DB与HDFS间数据导入导出）、Kafka（流式消息）。
* **分析与挖掘**：Hive（SQL on Hadoop）、Pig（脚本式分析）、Spark（内存计算）、Mahout（机器学习）。
* **特定应用/增强**：Tez（高性能DAG执行）、Ranger/Sentry（安全）、Kylin（多维分析）、Phoenix（HBase SQL）、Flink（流处理）。
* **开发者友好工具**：Beeline CLI、WebHDFS、Java API等多终端接口。

**为何出现众多产品？**

* **满足实际需求多样性**：不同行业、场景（批量、流式、实时、图计算、SQL分析）需要不同架构和工具。
* **分层解耦**：技术本源为解放单点架构，带来可扩展、可插拔（如YARN资源层、HDFS存储层、各种计算与应用层）。
* **促进开源生态繁荣**：工业界通过补足Google闭源技术缺环，不断丰富大数据能力，共同推动发展。

## 3．简述Hadoop即其生态圈产品安全机制，试着找出其风险漏洞。

**安全机制：**

* **认证**：
  + Kerberos集成，统一Hadoop集群身份认证（用户/服务票据机制）。
  + HTTP认证，限制Web接口访问。
* **权限与访问控制**：
  + ACL（访问控制列表），细致控制用户/组操作权限。
  + HDFS本身对目录/文件实施权限（类似UNIX模式）。
  + 服务间调用支持Token认证和ACL限制。
* **数据安全**：
  + HDFS支持传输和存储加密（如本地库中的加解密）。
  + NameNode元数据、数据块校验和机制确保存储完整性。

**主要风险与漏洞：**

* Kerberos配置复杂，错误配置易致认证绕过或失效。
* 部分交互（如某些Web接口、历史日志）可能被绕过安全限制。
* 集群通信未用加密时，网络窃听风险较大。
* 服务间认证被禁用时、ACL管控失误易被滥用（如未限制超级用户操作）。
* 尽管支持多备份/快照，但人为误操作或恶意删改依然可能造成数据丢失，需定期演练恢复。

## 4．简述Yarn编程过程，简述MR、Spark编程过程，思考三者有何关系。

**YARN编程过程：**

1. Client提交作业给ResourceManager。
2. ResourceManager为每个作业分配ApplicationMaster，AM协调具体任务（如MR/Spark）。
3. ApplicationMaster向Scheduler申请资源（Container）。
4. NodeManager在指定机器启动任务（如MR的Map/Reduce/Spark Executor）。
5. 任务进度、资源状态通过心跳上报，实现自动重调度、容错。
6. 全部完成后，AM和作业注销、资源释放。

**MapReduce编程过程：**

* Map函数将输入数据切片，各片并发处理。
* Map输出（key,value）按照key分组、洗牌至Reduce节点。
* Reduce阶段合并、加工同key数据，生成最终结果。
* MR流程严格分Map&Reduce两阶段，不支持复杂DAG。

**Spark编程过程：**

* 用户用Scala/Java/Python编写RDD/Dataset等并行操作。
* Spark内部构建DAG（有向无环图）任务依赖，自动切成Task并并发执行。
* 通过Driver调度，Executor执行，支持数据缓存、宽依赖、管道优化和复杂工作流。

**三者关系：**

* YARN是统一资源管理平台，能调度MR/Spark等多种计算框架资源。
* MR、Spark都可运行在YARN之上，获取资源并进行分布式任务处理。
* MR是早期批处理框架，Spark以DAG/内存优化成为“快批批结合”新一代代表，YARN为其“操作系统”基础。

## 5．简述数据“生产、收集、传输、存储、分析、应用”每阶段典型开源软件。

* **生产**：Web服务器、业务应用（日志/行为数据产生地）
* **收集**：Flume（日志采集）、Logstash、Chukwa
* **传输**：Kafka（消息中间件）、Flume、Sqoop（DB数据迁移）
* **存储**：
  + HDFS（分布式存储，文件级）
  + HBase、Cassandra（NoSQL，结构化/半结构化数据）
  + Ozone（对象存储），Redis（缓存）、Hive（数据仓库元数据）
* **分析**：
  + MapReduce、Spark（批处理、内存计算）
  + Hive、Pig（SQL/脚本分析）
  + Mahout（分布式机器学习）、Flink/Storm（流式计算）、Tez（DAG计算）
* **应用**：
  + 数据可视化/报表（如Hue，Superset），
  + 机器学习（Mahout，Mllib），
  + BI系统/推荐系统，Web/Mobile服务端，工业云等各种大数据应用

# 第六章作业

## 1. 虚拟化技术在云计算中的哪些地方发挥了关键作用？

* **计算资源虚拟化（服务器虚拟化）**：  
  是实现云主机（VM）、弹性伸缩、资源池化底座，支持IaaS（如OpenStack Nova、AWS EC2）。
* **存储虚拟化**：  
  把物理存储整合成统一的逻辑存储池，支撑云盘、对象存储（如OpenStack Cinder、Swift）。
* **网络虚拟化**：  
  实现灵活、安全的云网络（子网、私有云、虚拟交换机），支持多租户隔离（如OpenStack Neutron）。
* **桌面虚拟化**：  
  提供桌面云、云桌面、远程办公服务（Application/VDI）。
* **管理与调度**：  
  通过API集中管控，自动按需分配和动态迁移计算/存储/网络资源，实现高可用与自动故障恢复。

## 2. 服务器虚拟化、存储虚拟化和网络虚拟化都有哪些实现方式？

#### 服务器虚拟化实现方式

* **全虚拟化（Full Virtualization）**：  
  典型如VMware ESXi、KVM。使用VMM/Hypervisor完全模拟硬件，客户机无需修改。
* **半虚拟化（Para-Virtualization）**：  
  如Xen早期模式，客操作系统需改造以配合Hypervisor，性能好于全虚拟化。
* **操作系统级虚拟化（Container）**：  
  如Docker、LXC，基于共享内核实现多个进程隔离，启动快、资源开销小。

#### 存储虚拟化实现方式

* **块级虚拟化**：  
  统一管理SAN等物理块设备如LVM、Ceph-RBD。
* **文件级虚拟化**：  
  跨物理NAS文件系统聚合（DFS，GlusterFS）。
* **对象存储虚拟化**：  
  面向对象存储接口（如Swift、Ceph-Object），支持弹性扩展，适合海量非结构化数据。

#### 网络虚拟化实现方式

* **VLAN/VXLAN/GENEVE等网络分段**：  
  虚拟交换、路由支持多租户隔离（如Linux Bridge、OVS）。
* **SDN（软件定义网络）**：  
  控制/数据分离，通过控制器统一编排、自动配置虚拟网络。
* **NFV（网络功能虚拟化）**：  
  软化各类防火墙、负载均衡器等网络功能。

## 3. 讨论桌面虚拟化的实现和作用。

#### 实现方式

* **远程桌面协议（RDP、VNC、PCoIP）**：  
  用户操作界面和应用运行在远端虚机/服务器，通过网络将画面交付本地。
* **VDI（虚拟桌面基础设施）**：  
  集群服务器集中运行不同用户桌面，终端仅负责输入输出（如Citrix XenDesktop、VMware Horizon）。
* **应用虚拟化**：  
  只将应用界面/进程托管在中央服务器，而非整个桌面环境。

#### 作用

* 简化桌面运维和运算资源分配，提高桌面管理安全性和弹性，适合集中办公、远程办公、弹性扩缩需求。
* 保障数据不落地终端，降低数据泄露风险，支持远程接入。
* 硬件利用率更高，降低终端故障与维护负担。

## 4. 总结OpenStack的主要组件及其功能。

* **Nova**：计算资源管理（VM生命周期、调度、迁移等）。
* **Glance**：镜像管理（注册、存储、分发虚机镜像）。
* **Cinder**：块存储服务（云盘的申请、挂载、快照等）。
* **Neutron**：网络服务（虚拟网络、子网、安全组、负载均衡、VPN等）。
* **Swift**：对象存储服务（海量非结构化数据的弹性存储与访问）。
* **Keystone**：身份认证和授权（统一管理服务、用户、权限）。
* **Horizon**：基于Web的管理界面。
* **Heat**：编排服务，支持一键部署复杂云环境（自动化模板）。
* **Ceilometer/Gnocchi**：监控与计量服务。
* **其他**：如Ironic（金属裸机管理）、Magnum（容器集群管理）等。

## 5. 请根据学过的知识总结一下各服务模块之间如何协同工作。

* Keystone提供统一身份认证，各模块（Nova、Cinder、Swift、Neutron等）通过Keystone校验用户权限。
* 用户通过Horizon、API或命令行操作云服务，调度由Nova协调，需求云盘时调用Cinder，分配网络通过Neutron，镜像分发由Glance处理。
* Swift为对象存储服务，可接入Nova、Glance等作为镜像源或备份存储。
* Heat可调用各基础服务编排云资源，实现自动化部署。
* 监控计量（Ceilometer）为资源弹性伸缩、计费、安全审计等提供数据。
* Message Queue（如RabbitMQ）支撑各核心服务异步通信和高效协作。

## 6. 请通过学过的知识概括一下OpenStack与AWS的异同。

#### 相同点

* 都是完整的公有/私有云操作系统，实现“计算+存储+网络”等基础资源池化。
* 都支持按需自助服务、弹性伸缩、多租户、自动计费等云服务关键特性。
* 核心资源服务模型类似，且具备丰富API接口、生态插件体系。

#### 不同点

* **开源/商用**：OpenStack为开源社区推动，由厂商和用户贡献代码；AWS为Amazon自研完全商用化产品。
* **部署和控制权**：OpenStack可本地或私有化部署，用户主控；AWS仅作为云服务订购，无需自建物理资源。
* **功能集成/完善度**：AWS服务种类更丰富（AI、IoT、Serverless等领域领先），OpenStack偏向基础IaaS/PaaS层。
* **生态互通与标准**：AWS有自有数据格式、专属API；OpenStack强调跨厂商和开放接口兼容性。

## 7. 请对比超融合架构和传统架构的优缺点。

| **超融合架构（HCI）** | **传统架构** |
| --- | --- |
| 架构形态 | 计算、存储、网络三合一，软件定义，共用X86服务器 | 计算、存储、网络物理分离 |
| 优点 | 部署灵活、弹性扩展、统一管理、资源利用率高、TCO低 | 专项硬件高性能、分工明确、技术成熟 |
| 缺点 | 对超大规模场景挑战较大，某些高性能需求需定制优化 | 扩展慢、建设和运维成本高，资源池弹性差 |
| 适用性 | 中小企业、私有云/边缘云，快速上线场景 | 金融、运营商等高性能/高安全行业 |

## 8. OpenStack Swift中默认配置设置，R是否可为1？简单说明理由。

R（副本数，replica）**默认不可为1**。

**理由**：  
Swift作为对象存储，其核心目标为高可用与容错。R=1，任何节点或硬盘损坏都将导致数据不可恢复（无副本、无冗余），违背分布式存储“数据可靠”设计原则。默认配置要求R≥3（生产环境推荐），以实现至少跨节点/机柜3份分布、1份损坏不影响数据访问。

# 第七章作业

## 1.容器的概念及与虚拟机的区别。

**容器的概念：**

* 容器是一种轻量级的、可移植的封装技术，通过操作系统内核层的“逻辑隔离”机制，让应用和其依赖环境一同打包，并作为独立单元在主机上运行。
* 容器拥有独立的进程空间、文件系统、网络和资源限制，但直接共享宿主机操作系统内核。

**与虚拟机的区别：**

* 虚拟机（VM）通过Hypervisor虚拟出完整硬件并运行各自独立的操作系统，隔离在操作系统和驱动层。
* 容器是在宿主操作系统内核之上隔离应用，多个容器共享同一个操作系统内核，省去了来宾操作系统层，启动更快、资源占用更低。
* 容器体积小、启动速度快、易于快速弹性扩缩容；虚拟机隔离性强、兼容性好，但资源开销大，启动慢。

## 2.容器化的关键技术及其优势是什么？

**关键技术：**

* **Namespace（命名空间）**：为进程分配独立的资源视图（如PID、网络、挂载点、用户等），实现进程间隔离。
* **Cgroups（控制组）**：对CPU、内存、IO等物理资源进行定量划分与限制，实现公平分配。
* **联合文件系统（UnionFS）**：将不同层次的文件系统按只读/读写方式叠加，支持镜像快速分发和层次化存储。
* **镜像技术**：应用和环境打包为只读镜像，支持可移植、可复用和快速部署。
* **容器引擎**：如Docker、rkt、LXC等，负责容器生命周期管理和底层调用。

**优势：**

* 启动快、迁移方便、资源占用更低、更高密度部署、弹性伸缩能力强。
* 应用、配置、依赖环境一体化封装，实现“即开即用”与一致性的运行环境。
* 支持微服务架构，以及无状态服务的快速弹性扩展。
* 易于自动化交付和持续集成/持续部署（CI/CD）。

## 3.简述Docker的系统架构及基本使用流程。

**系统架构主要组件：**

* **Docker 客户端**：CLI工具，向Docker守护进程发送命令（如build、run）；
* **Docker Daemon（守护进程）**：负责真正的镜像构建、容器管理等实际工作；
* **Docker 镜像仓库（Registry）**：集中存储镜像，如Docker Hub或私有仓库；
* **Docker 镜像（Image）**：只读模板，用于创建容器；
* **Docker 容器（Container）**：镜像实例化并运行，具备隔离环境。

**基本流程：**

1. 开发者编写Dockerfile定义镜像。
2. 用docker build命令生成镜像，推送至Registry。
3. 用户通过docker pull拉取镜像，docker run启动容器实例。
4. 容器运行后可随时停止、删除，支持日志查看和资源监控。

## 4.Rkt容器与Docker有哪些差别？

* Rkt由CoreOS开发，强调安全、开放和可与系统集成。
* Rkt采用“无守护进程”设计，每个容器进程由systemd等管理，提升安全隔离。
* 支持多种镜像格式（App Container Image、Docker Image），兼容性强。
* 更注重安全（沙箱机制、多用户namespace）、可组合（stage1阶段灵活定义容器运行方式）和集群友好（易与Kubernetes等集成）。
* Docker则以守护进程主导，采用单一镜像/运行机制，生态广，但默认隔离粒度有限。

## 5.Garden容器是如何实现资源控制的？

* Garden采用Linux Cgroups技术对CPU、内存、IO等物理资源进行隔离和限制。
* 通过namespace实现进程、网络、文件系统等隔离。
* 支持每个容器定制资源上限，基于BOSH与Cloud Foundry集群编排，控制不同租户或应用的资源分配，实现有效的多租户资源隔离管理。

## 6.简述Kubernetes的系统架构组成。

**主要组件包括：**

* **Master节点（控制平面）**：
  + **kube-apiserver**：RESTful API入口，统一接受与处理请求；
  + **etcd**：分布式键值存储，保存集群所有配置信息与状态；
  + **kube-scheduler**：负责Pod的调度到具体Node；
  + **kube-controller-manager**：各种控制器（如副本数管理、节点监控、故障自动恢复等）。
* **Node节点（工作节点）**：
  + **kubelet**：负责Pod生命周期管理；
  + **kube-proxy**：服务发现与负载均衡；
  + **容器运行时**：如Docker、containerd，负责拉取和运行容器。

## 7.主流容器编排系统有哪些？它们的特点是什么？

* **Kubernetes**：谷歌发起、业界主流、支持自动调度、伸缩、服务发现、自愈、声明式API等，社区活跃度高，生态完善。
* **Docker Swarm**：Docker原生编排工具，易用性好、较为轻量，适合小型场景，深度集成Docker生态。
* **Apache Mesos + Marathon**：通用资源调度，支持容器和非容器负载，灵活但部署维护复杂，常用于大公司数据中心或混合计算场景。
* **OpenStack Magnum**：为OpenStack平台集成K8s、Swarm等容器支持。

## 8.Kubernetes采用了哪些理念来满足基于Docker的大规模、大数据计算的场景？

* **声明式API（Declarative API）**：用户只需声明“期望状态”，K8s自动调达成目标。
* **自动扩缩容**：根据负载自动添加/释放Pod，提高资源利用和弹性。
* **滚动升级和自动恢复（Self-healing）**：服务不中断地实现升级、失败自愈、容器重建。
* **多租户与资源隔离**：资源配额、命名空间、调度策略满足多用户安全和公平分配。
* **横向、纵向扩展**：支持业务“批+流”混合、多Pod调度、高并发任务。
* **服务发现与负载均衡**：自动生成服务DNS、IP，支持流量自动分配。
* **灵活扩展与插件架构**：网络、存储、调度等均可集成多种开源方案。
* **对大数据友好**：有原生支持Hadoop/Spark等大数据平台的Operator与CRD管理。

## 9.如何基于Docker和Kubernetes来构建一个Hadoop大数据系统？

**基本思路和步骤：**

1. **镜像准备**：制作Hadoop相关角色（NameNode、DataNode、ResourceManager等）Docker镜像，内含配置和依赖。
2. **YAML模板编排**：定义Pod、Service、StatefulSet、ConfigMap等描述Hadoop各节点应用部署方式。
3. **持久存储**：将HDFS数据挂载到K8s的PersistentVolume，保证容器重启/迁移中数据不丢失。
4. **网络配置**：配置K8s网络以支持Hadoop各角色节点互通。
5. **启动与扩缩容**：用kubectl部署各节点，支持任意Hadoop节点的横向扩展（DataNode/NodeManager可弹性伸缩）。
6. **服务暴露**：通过K8s Service方式暴露Web UI、RPC端口，实现外部访问与管理。
7. **高可用**：利用K8s自动修复、滚动升级机制提升Hadoop集群容错能力。
8. **调度优化**：可根据大数据作业特性定制K8s调度策略（节点选择、亲和/反亲和、资源预留等）。

**优势：**

* 充分利用Docker快速部署、易移植的优势，配合K8s自动编排和弹性能力，实现运维自动化、弹性伸缩和更好的资源利用率
* 适合快速搭建、测试及弹性生产大数据平台。

# 第八章作业

## 1．云原生的概念及核心内容是什么？

云原生（Cloud Native）是由Pivotal公司的Matt Stine在2013年提出的概念，它是一种专注于构建和运行云应用的方法论，旨在充分利用云计算的优势，实现应用的快速迭代和高效运维。云原生的核心内容包括以下几个方面：

* **微服务（MicroServices）**：将复杂的应用程序拆分成多个小型、独立的微服务，每个微服务专注于一个特定的业务功能，可以独立开发、部署和扩展。
* **敏捷基础设施（Agile Infrastructure）**：通过容器化、自动化等技术，实现基础设施的快速配置和灵活调整，以适应应用的动态变化。
* **DevOps**：强调开发和运维团队之间的紧密协作，通过自动化工具和流程，实现应用的快速交付和持续改进。
* **持续交付（Continuous Delivery）**：确保软件在任何时候都可以可靠地发布，通过自动化构建、测试和部署流程，实现软件的快速迭代。

## 2．什么是微服务？微服务架构作用及其分类。

**微服务**是指将大型复杂软件应用拆分成多个简单应用，每个简单应用描述一个小业务，系统中的各个简单应用可被独立部署。相比传统的单体架构，微服务架构具有降低系统复杂度、独立部署、独立扩展、跨语言编程等特点。

**微服务架构的作用**：

* **降低系统复杂度**：将复杂的应用拆分成多个小型服务，每个服务专注于一个特定功能，易于理解和维护。
* **独立部署和扩展**：每个微服务可以独立部署和扩展，根据业务需求灵活调整资源分配。
* **技术栈灵活**：可以根据项目业务及团队特点，选择最适合的技术栈。
* **快速迭代**：局部修改容易部署，支持快速迭代和创新。

**微服务架构的分类**：

* **侵入式架构**：服务框架嵌入程序代码，实现类的继承，以Spring Cloud最为常见。
* **非侵入式架构**：以代理的形式与应用程序部署在一起，接管应用程序的网络且对其透明，以服务网格（Service Mesh）为代表。

## 3. 对比分析Spring Cloud和Dubbo架构特点。

**Spring Cloud**：

* **基于Spring Boot**：利用Spring Boot的开发便利性，简化了分布式系统基础设施的开发。
* **功能丰富**：提供了服务注册发现、配置中心、消息总线、负载均衡、断路器、数据监控等功能。
* **侵入式架构**：需要在代码中集成Spring Cloud的相关组件，对业务代码有一定侵入性。
* **社区活跃**：与Spring生态系统紧密结合，社区支持丰富，文档和案例众多。

**Dubbo**：

* **高性能RPC框架**：提供高性能的RPC调用能力，支持多种协议，如Dubbo协议、HTTP协议等。
* **服务治理功能强大**：支持服务自动注册和发现、负载均衡、容错、监控等功能。
* **非侵入式架构**：通过配置文件和服务治理中心实现服务的管理和调用，对业务代码侵入性较小。
* **社区活跃度较低**：虽然有阿里巴巴等大厂的支持，但社区活跃度相对Spring Cloud较低。

## 4. 如何基于容器技术构建容器云平台？

构建容器云平台通常需要以下步骤：

* **选择容器运行时**：如Docker，它是目前最流行的容器运行时，提供了容器的创建、管理和运行功能。
* **构建容器镜像**：将应用程序及其依赖打包成容器镜像，确保在不同环境中的一致性。
* **选择容器编排工具**：如Kubernetes，它提供了容器的编排、调度、管理等功能，支持大规模容器集群的管理。
* **配置存储和网络**：为容器提供持久化存储和网络连接，确保容器之间的通信和数据存储。
* **部署和管理应用**：通过容器编排工具将应用部署到容器云平台，并进行监控、扩缩容等管理操作。

## 5. 消息中间件传输模型有哪两种？

消息中间件的传输模型主要有以下两种：

* **点对点模型（Point-to-Point, PTP）**：生产者发送消息到队列，消费者从队列中消费消息。每个消息只能被一个消费者消费，适用于一对一的通信场景。
* **发布/订阅模型（Publish/Subscribe, Pub/Sub）**：生产者发布消息到主题，消费者订阅主题接收消息。一个消息可以被多个消费者消费，适用于一对多或多对多的通信场景。

## 6. 常见的分布式消息中间件服务有哪些？区别是什么？

常见的分布式消息中间件服务包括：

* **Kafka**：分布式发布-订阅消息系统，具有高吞吐量、可持久化、易扩展等特点，适合处理大规模消息数据。
* **ActiveMQ**：支持多种消息传输模型，如PTP和Pub/Sub，功能丰富，社区支持良好。
* **RabbitMQ**：基于AMQP协议的消息中间件，支持多种消息分发策略，如Direct、Fanout、Topic等，适合复杂的路由场景。
* **RocketMQ**：阿里巴巴开源的消息中间件，支持高并发、高可用，适合大规模分布式系统。

它们的区别主要体现在性能、功能、适用场景等方面：

* **性能**：Kafka在处理大规模消息数据时性能较高，RocketMQ在高并发场景下表现较好。
* **功能**：ActiveMQ功能较为全面，支持多种协议和传输模型；RabbitMQ在消息路由和分发策略方面更为灵活。
* **适用场景**：Kafka适合日志收集、流处理等场景；ActiveMQ适合企业级应用的消息通信；RabbitMQ适合复杂的业务逻辑和消息分发场景；RocketMQ适合电商等高并发业务场景。

## 7. 什么是DevOps？常用工具有哪些？

**DevOps**是一组过程、方法与系统的统称，强调开发和运维团队之间的紧密协作，通过自动化工具和流程，实现软件的快速交付和持续改进。DevOps的目标是缩短开发周期，增加部署频率，更可靠地发布软件。

常用的DevOps工具包括：

* **Ansible**：开源配置管理工具，支持多种语言，无客户端架构，使用SSH进行通信，适合自动化部署和配置管理。
* **SaltStack**：基于C/S架构的服务器管理平台，支持配置管理、远程执行、监控等功能，适合大规模集群管理。
* **Jenkins**：开源持续集成工具，支持自动化构建、测试和部署，具有丰富的插件生态系统。
* **Terraform**：基础设施管理工具，用于构建、更改和管理基础设施，支持多种云平台。
* **Packer**：用于创建虚拟镜像的工具，支持多种镜像格式。

## 8. 持续交付与持续集成、持续部署的关系。

* **持续集成（Continuous Integration, CI）**：是一种开发实践，要求开发者频繁地将代码提交到共享代码库，并通过自动化构建和测试确保代码质量。它是持续交付的基础，确保代码的集成和质量。
* **持续交付（Continuous Delivery, CD）**：是一种软件工程手段，确保软件在任何时候都可以被可靠地发布。它包括自动化构建、测试、部署等流程，确保软件可以快速、可靠地交付到生产环境。
* **持续部署（Continuous Deployment）**：是持续交付的下一步，指代码通过评审和自动化测试后，自动部署到生产环境。它要求更高的自动化程度和质量保障。

**三者的关系**：

* 持续集成是持续交付的前提，通过频繁的代码集成和自动化测试，确保代码质量。
* 持续交付是持续集成的延伸，通过自动化流程和工具，确保软件可以快速、可靠地交付到生产环境。
* 持续部署是持续交付的更高阶段，实现了代码的自动部署，进一步提高了软件交付的速度和效率。

# 第九章作业

## 1．集装箱数据中心有哪些优点？常见的节能措施有哪些？

**优点**：

* **高密度**：集装箱数据中心模块可容纳高密度计算设备，相同空间内可容纳六倍于传统数据中心的机柜数量。
* **模块化**：有利于数据中心的模块化，可以建立一个最优的数据中心生态系统，具有所需的供电、冷却和计算能力。
* **按需快速部署**：不需要企业再经过空间租用、土地申请、机房建设、硬件部署等周期，可大大缩短部署周期。
* **移动便携**：安装非常容易，只需要提供电源连接、水源连接（用于冷却）和数据连接即可，可移动性强。

**节能措施**：

* **缩短送风距离**：将空调盘管安装到服务器顶部，减少送风过程中的冷量损失。
* **提高冷通道温度**：提高冷通道温度后，可升高盘管供/回水温度，减少压缩机工作时间，提高冷水机组的工作效率。
* **冷/热通道完全隔离**：防止冷、热空气混合，增大进、出风温差，提高盘管制冷能力。
* **隔热保温材料**：在集装箱内外涂隔热保温涂料，防止冷量外泄。
* **Free Cooling功能**：使用Free Cooling减少压缩机工作时间，提高能源利用率。

## 2．云计算数据中心配电系统节能的原理是什么？

* **高压直流配电技术**：将传统的集中供电分散到每个机架，去除了服务器节点的交流电源系统，减少了转换环节，提高了配电系统的效率。
* **市电直供配电技术**：最大化减少配电系统的转换环节，整个配电系统的综合效率能够达到92%左右。
* **优化UPS系统**：采用更高效的UPS系统，减少电力转换过程中的损耗。
* **动态功率管理**：根据负载动态调整功率，降低空闲能耗。
* **虚拟化技术**：通过虚拟化减少硬件资源的使用数量，提高资源利用率，从而降低能耗。

## 3．能源利用效率（PUE）的计算方式是什么？

能源利用效率（PUE）是衡量数据中心能源效率的一个重要指标，其计算方式如下： PUE=数据中心总能耗/IT设备能耗​

* **数据中心总能耗**：包括IT设备能耗、空调系统能耗、配电系统能耗以及其他辅助设备能耗。
* **IT设备能耗**：主要是服务器、存储设备等IT设备的能耗。

PUE的基准值为2，比值越接近1，表示数据中心的能源利用率越高。

## 4．海底数据中心与传统数据中心对比。

| **特征** | **传统数据中心** | **海底数据中心** |
| --- | --- | --- |
| 建设周期 | 1-2年 | 90天左右 |
| 制冷设施 | 需要 | 不需要 |
| 营运成本 | 高（用电量及电价相对较高；电费0.8元/度） | 低（用电量及电价相对较低；电费≤0.4元/度） |
| 数据传输 | 高延迟 | 低延迟 |
| 安全性 | 需要额外的安全措施 | 高安全性，利用海水冷却，减少火灾风险 |
| 环境影响 | 高能耗，对环境影响较大 | 低能耗，利用海水冷却，减少碳排放 |
| 部署灵活性 | 需要固定场地，部署复杂 | 可灵活部署在沿海地区，便于快速扩展 |

海底数据中心通过与海水进行热交换进行冷却，无需额外部署制冷系统，营运成本具有明显优势。同时，其模块化建设方式使得施工周期短，部署灵活，适合在沿海发达地区使用，可以有效降低数据传输的延迟。

# 第十章作业

## 1．传统云计算有哪些缺陷和挑战？

传统云计算存在以下缺陷和挑战：

* **平台锁定问题**：消费者过分依赖单一云服务提供者的产品和服务，无法在不产生大量切换成本的情况下进行灵活迁移，造成平台锁定问题。
* **数据安全性问题**：消费者将数据全部交给一个云服务提供者保管，若云服务提供者内部出现问题，很容易获取用户的所有信息，数据隐私泄露风险较高。
* **服务可靠性问题**：若云服务提供者停止服务或遭遇故障，各类云服务消费者将面临数据全部丢失的风险，造成不可挽回的经济损失。
* **效费比问题**：许多使用云服务的互联网创新业务在全球化的过程中越来越受制于单一云服务提供者应对业务爆发性的能力，例如“双十一”等场景需要超过10倍于日常需求的IT资源，给云服务提供者带来了巨大的IT资源效费比问题。

## 2．云际计算的概念是什么？有哪些代表性工作？

**云际计算的概念**： 云际计算是以云服务提供者之间开放协作为基础，通过多方云资源深度融合，方便开发者通过“软件定义”方式定制云服务，创造云价值，实现“服务无边界、云间有协作、资源易共享、价值可转换”的新一代云计算模式。

**代表性工作**：

* **Inter-Cloud**：由思科公司提出，试图通过使用其他云服务提供者的资源，解决单一云服务提供者物理资源有限或地理上难以实现无处不在的覆盖等问题。
* **SuperCloud**：强调以用户为中心，试图跨多个云服务提供者构建全套透明的计算、存储和数据通信服务。
* **Multi-Cloud**：通过更多云服务提供者共存的软硬件基础设施冗余优化容错能力，降低云用户数据丢失或云计算环境中局部构件失效所引起的宕机等情况下产生的服务中断风险。
* **Federated Cloud**：在多个未建立信任关系的云服务提供者之间，通过部署和管理多个内外部云计算服务来匹配业务需求，支持若干个独立云平台合作完成一个目标任务。
* **Cloud Service Broker (CSB)**：为云服务提供者与云服务消费者提供咨询、集成和协商谈判服务，通过创新渠道和市场机会，为云服务提供者增长销量。

## 3．云际计算的优势有哪些？

云际计算的优势包括：

* **业务连续性**：通过在多个云服务提供者之间分散部署云资源，即使某个特定的云服务提供者遭受攻击或故障，托管于其他云服务提供者的资源不会受影响，从而确保业务连续不中断。
* **数据可靠性**：通过复制或纠删码的方式将数据存储在多个云中，显著提高数据可靠性，降低因云失效引起的数据丢失风险。
* **访问性能**：充分利用每个云服务提供者的优势，根据用户的需求将不同用户的数据发送给不同的云服务提供者，获得最佳的云服务。
* **安全隐私**：使用纠删码来跨多个云服务提供者分发数据，确保任何单个云服务提供者都不会拥有数据的完整内容，从而减轻隐私担忧。
* **管理灵活性**：根据每个云的价格以及擅长的特定服务来最佳地选取多个云的使用策略，避免平台锁定。
* **存储成本**：使用多个云的客户可以为不同的服务选择资费最合理的云服务提供者，实现最大的成本效益。

## 4．云计算的架构有哪些？分别阐述每种架构的机制和优缺点。

云际计算的架构主要包括以下几种：

* **集中式代理架构**：
  + **机制**：将云际代理部署在本地或私有云等可信任的位置，连接用户和不可信的各大云服务提供者。系统大致分为数据处理层、选取策略层及数据库层三个部分。
  + **优点**：方便管理文件元数据信息。
  + **缺点**：不适合地域跨度大的云际部署，扩展性差。
* **半集中式代理架构**：
  + **机制**：根据地理位置部署多个云代理系统，每个代理系统帮助分发其附近用户的数据到各大云服务提供者，各代理系统间通过中心协调服务统一协调传递信息。
  + **优点**：相比集中式代理架构具有更好的扩展性，且对地理分散的用户依然能保持良好的性能。
  + **缺点**：随着数据量的进一步增长，依然会面临中心协调服务的性能瓶颈。
* **分散式代理架构**：
  + **机制**：按地理位置部署多个云代理系统，同时为每个代理系统附加协调器组件，协调器定期交换状态，无须中央控制。
  + **优点**：有更好的发展前景，避免了中心协调服务的性能瓶颈。
  + **缺点**：管理难度较大。

## 5. 云际协作环境的三类核心服务是什么？分别阐述其作用。

云际协作环境的三类核心服务包括：

* **分布云交易**：
  + **作用**：针对云服务消费者与云服务提供者，以及不同云服务提供者之间如何确定服务等级协议、如何处理违约、如何计量计费等问题，提供拍卖竞价、协商谈判和计量计价等服务支持。
* **分布云社区**：
  + **作用**：提供云际协作的社区服务。异源异质异构的不同云服务提供者，为了提高互操作性，可依据分布云社区提供的软件定义标准，封装服务接口，在逻辑层面实现同质化，达成可互操作的基本条件。
* **分布云监管**：
  + **作用**：为云交易提供关于云服务提供者的信用度量和评价、云服务消费者的消费行为审计、服务等级协议及契约的违约举证等方面的机制支持。