相对于集中式计算，云计算架构优点是什么？

云计算可以提供可伸缩的按需服务、弹性定价；有着灵活性和可扩展性，可以随时随地的访问，有着高可用性和容错性，可以自动化和管理简化，有安全性和合规性，还有着技术更新和创新性

云计算架构按照服务分层，可以划分哪三种类型？

基础层（硬件/物理层，虚拟化层）、管理中间件层、用户交互层

云计算服务采用的电厂模式，含义是什么？

云计算服务所采用的电厂模式，通常指的是“资源池化”或“按需分配”的理念。有这几种含义：1.资源池化：就像电厂将电力资源集中在一起，云计算将计算、存储和网络资源池化，用户可以根据需要按需访问和使用这些资源。这样可以提高资源利用率，减少闲置；2.按需服务：用户可以根据实际需求随时获取资源，避免了一次性采购大量硬件和软件的成本和浪费。用户只需为使用的资源支付费用，这种按需计费的方式更加灵活；3.弹性扩展：类似于电厂能够根据需求波动调整电力供应，云计算服务也能快速扩展和缩减资源，满足不同用户在不同时间的计算需求；4.集中管理：所有的计算和存储资源通过集中管理，可以实现更高效的维护和监控，确保资源的安全性和可靠性

什么是吉姆格雷提出的科学研究第四范式？

数据驱动的科学。这一范式的核心在于海量数据的采集和分析。科学研究不仅依赖于理论和实验，更加重视通过大数据、数据挖掘和分析技术从数据中提取知识和发现新的模式

GFS 文件系统中，Master 服务器的作用是什么？

Master 是 GFS 系统的管理节点，在逻辑上只有一个，主要负责存放元数据（包括文件命名空间、地址映射、权限控制等）和实现整个分布式文件系统管理（系统容错、负载均衡、操作日志、故障恢复、和集群维护等）

GFS 文件系统中，为什么要统一存储块大小为 64MB 或 128MB?

减少元数据开销：大块大小可以减少系统需要管理的块数量，从而减少元数据的开销;提高数据读取效率：大块可以有效地利用磁盘的顺序读取性能，减少随机访问的次数;恢复效率：当一个节点故障时，GFS 可以通过副本进行数据恢复;适应大文件场景;减少碎片：使用大块可以减少文件系统中的碎片问题

主从架构中，心跳技术的作用是什么？

心跳技术的作用主要是用来监测主节点和从节点之间的连接状态，以及节点的健康状况

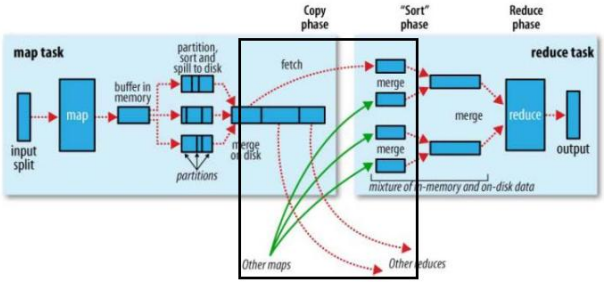
Bigtable 中，时间戳的作用是什么？

在 Bigtable 中，每一项数据都可以包含针对同一份数据的不同版本，不同版本的数据通过时间戳来区分，如果应用程序需要避免数据版本冲突，那么它必须生成具有唯一性的时间戳

Bigtable 中，如何确保数据一致性？

通过时间戳版本控制,Bigtable 使用时间戳来管理数据版本。每次写入时，数据会带上一个时间戳，读取操作可以指定要检索的时间戳，这样在多个版本之间可以保持一致性

介绍黑框中的处理过程，及红色虚线及绿色实线走向的含义



Copy Phase：在这个阶段，Reduce 任务从各个 Map 任务中获取数据。每个 Map 任务的输出被分割成多个分区，每个分区对应一个 Reduce 任务。这些分区数据被发送到对应的 Reduce 任务。Sort Phase：在 Reduce 任务接收到所有 Map 任务的数据后，会将这些数据进行合并和排序。这个阶段会将内存中的数据与磁盘上的数据混合，以确保数据的有序性。排序是按照键（key）进行的，这样具有相同键的所有值（value）都会被聚集在一起。

虚拟化技术中，二进制 BT 翻译技术的作用是什么？

二进制 BT 翻译技术的作用：用于将一种架构指令集的代码动态转换为另一种架构的指令集

简述宿主机架构和裸金属架构的差别及各自优点

裸金属架构是直接运行在机器硬件之上的，宿主机架构是运行在目标机器的操作系统上。裸金属架构的优点：有更高的资源利用率和安全性；宿主机架构优点：对于开发者和维护者要求门槛低、易于操作

相对于传统的虚拟机软件应用，Docker 容器技术有什么优点？

Docker 优点：Docker 封装了整个软件运行时的环境，开发者可以像管理应用一样管理应用运行时所需要的基础设施环境，测试和部署代码都很便捷，能够显著减少开发者写代码到发布应用之间的时间；运行系统环境时不用单独验证，只需要运行同一个 Docker 容器即可；

什么是 CAP 理论，什么是 Base 理论？

CAP 理论强调了分布式系统在一致性、可用性和分区容忍性之间的权衡，BASE 理论则提供了一种灵活的方式来处理这些权衡，以实现更高的可用性和可扩展性

什么是强一致性和最终一致性？

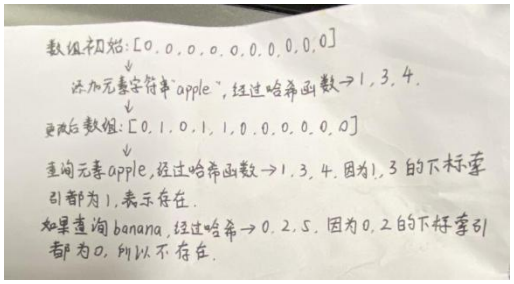
强一致性意味着在一个分布式系统中，当某个数据项被更新后，所有后续的读取操作都将返回该数据项的最新值，即无论在哪个节点上读取数据，系统都会确保读取到的信息是最新的。最终一致性强调系统在经历了一定的时间后，会达到一致状态。在这种模型下，虽然在某个时刻不同节点上的数据可能不一致，但系统会保证经过一段时间后，所有的更新会传播到所有副本，最终所有副本的数据将会一致

利用一致性哈希算法管理服务节点，体现了什么优点？什么是哈希偏斜？

哈希偏斜：在某种特殊情况下（尤其是节点较少的情况），服务器映射在哈希环中相对集中的位置，这有可能导致负载不均衡

介绍布隆过滤器的工作原理，附图举例说明。

布隆过滤器的原理：假设一个长度为 m 的字节的数组，数组中每个位置只占一个字节，每个字节只有两种状态 0 和 1，初始状态为 0，一共有 k 个哈希函数，这些函数的输出域大于或等于 m，并且这些哈希函数，彼此之间互相独立，每个哈希函数计算出来的结果是独立的，可能相同或不同，对每个计算出来的结果都对 m 取余，然后将相对应的数组下标改为 1



Merkel Tree 的构建，有什么实际应用价值？

Dynamo 中存储的数据存在多个副本，副本之间需要保持一致性，通过 Merkle Tree 可以实现数据一致性的快速判定，并能够定位出现问题的节点，从而极大地节省了比对时间和数据传输量

简述时钟向量技术的特点及作用。

特点：并发检测、无中心控制、精确性、数据开销性、实现复杂性、书简序列化性作用：向量时钟技术是用来解决数据一致性的问题

Docker 容器中，隔离划分 namespace 的作用是什么？

Namespace 的主要功能是将操作系统的某些资源（如文件系统、网络、进程 ID 等）“划分成独立空间”，确保容器之间互不干扰，同时也与宿主机隔离。

举例说明 Docker inspect 命令的作用

docker inspect 命令用于获取 Docker 对象（容器、镜像、卷、网络等）的详细信息。

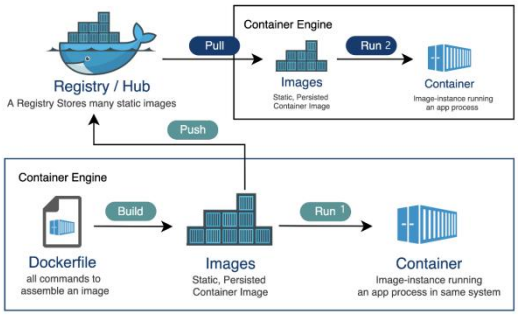
什么是写时复制技术？

写时拷贝就是等到修改数据时才真正分配内存空间，这是对程序性能的优化，可以延迟甚至是避免内存拷贝，目的就是避免不必要的内存拷贝。

简述 Docker 镜像的分层机制

在旧的镜像上构建新的镜像，新镜像不再是从底层开始，而是直接在旧的镜像上构建，这样就会出现镜像分层现象，新镜像就一层一层叠加生成的，没安装一共软件，就会叠加一层。

详细介绍以下相关组成部分以及相关流程步骤



自制一个 Docker 镜像文件 DockerFile 将其构建镜像，通过这个镜像运行(run1)容器可以部署应用程序在相同的系统中，也可以通过这个镜像推送(push)到注册表(registry/hub),其他的用户可以通过注册表拉取镜像，然后运行(run2)容器用于部署其他的应用程序

Docker 容器技术中，基本概念 Build, Ship, and Run 是什么？

Build（构建镜像）：镜像就像是集装箱包括文件以及运行环境等等资源。

Ship（运输镜像）：主机和仓库间运输，这里的仓库就像是超级码头一样。

Run （运行镜像）：运行的镜像就是一个容器，容器就是运行程序的地方。

简述 UnionFS 文件系统

UnionFS 可以把多个目录内容联合挂载到同一个目录下，而目录的物理位置是分开的。

UnionFs 可以把只读和可读写文件系统合并在一起，具有写时复制功能，允许只读文件系统的修改可以保存到可写文件系统当中

```
#include <stdio.h
#include "mpi.h"
int main(int argc, char** argv)
{
     int myid, numprocs, source;
```

```
    MPI_Status status;
    char message[100];
    MPI_Init(&argc, &argv);
    MPI_Comm_rank(MPI_COMM_WORLD, &myid);
    MPI_Comm_size(MPI_COMM_WORLD,&numprocs);

    if (myid != 0) {
        strcpy(message, "Hello World!");//为发送字符串赋值
        //发送字符串时长度要加 1，从而包括串结束标志
        MPI_Send(message,strlen(message)+1, MPI_CHAR, 0,99,MPI_COMM_WORLD);
    } else {
    //除 0 进程的其他进程接收来自于 0 进程的字符串数据
    for (source = 1; source < numprocs; source++) {
        MPI_Recv(message, 100, MPI_CHAR, source, 99,MPI_COMM_WORLD, &status);
        printf("I am process %d. I recv string '%s' from process %d.\n", myid,
message,source);
    }
    }
    MPI_Finalize();
}

#include <mpi.h>
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
// 计算一个数的真因子之和
int sum_of_proper_divisors(int n) {
    int divisors_sum = 1; // 1 是所有正整数的因子
    for (int i = 2; i <= sqrt(n); i++) {
        if (n % i == 0) {
            divisors_sum += i;
            if (i != n / i) {
                divisors_sum += n / i;
            }
        }
    }
    return divisors_sum;
}
// 查找亲和数对
void find_amicable_pairs(int limit) {
    int partner, num;
    for (num = 2; num < limit; num++) {
        partner = sum_of_proper_divisors(num);
        if (partner > num && partner < limit && sum_of_proper_divisors(partner) == num) {
```

```
                printf("(%d, %d)\n", num, partner);
            }
        }
}
int main(int argc, char *argv[]) {
    int rank, size;
    MPI_Init(&argc, &argv); // 初始化 MPI 环境
    MPI_Comm_rank(MPI_COMM_WORLD, &rank); // 获取当前进程的进程号
    MPI_Comm_size(MPI_COMM_WORLD, &size); // 获取总进程数
    if (rank == 0) { // 主进程执行计算任务
        find_amicable_pairs(10000); // 设置查找上限为 10000，可以根据需要调整
    } else { // 其他进程等待主进程完成计算后退出
        MPI_Finalize(); // 其他进程调用此函数结束 MPI 环境，避免资源占用
    }
    return 0; // 主进程返回 0 表示成功结束程序
}
```

因为 sqrt 函数在 math.h 头文件中，但 gcc 的库文件中不包含 math 库，所以需要添加-lm 参数（-l 指定库，m 为 math 库）

树目录的创建

```
qwins@qwins:~$ sudo mkdir -pv test/myfile/{bin,boot/grub,dev/{rc.d/init.d,sysconfig/network},lib/mod,a,b,c}
mkdir: 已创建目录 'test/myfile'
mkdir: 已创建目录 'test/myfile/bin'
```

```
qwins@qwins:~/test$ tree
.
└── myfile
    ├── a
    ├── b
    ├── bin
    ├── boot
    │   └── grub
    ├── c
    ├── dev
    │   ├── rc.d
    │   │   └── init.d
    │   └── sysconfig
    │       └── network
    └── lib
        └── mod

14 directories, 0 files
```

shell 脚本

```
#!/bin/bash
echo "Hello World"
chmod +x test.sh    # 赋予可执行权限
./test.sh           # 执行程序
```