1. 为什么有时候最大程度地实现透明性并不总是好的?
    - 过高的透明性会带来一些缺点：掩盖通信的性能问题，不利于定位系统失效节点和判断系统问题，可能牺牲性能，可能会暴露系统分布特征（分布式系统为了维持透明性和一致性，需要在各节点频繁交换信息，信息中包含系统的结构相关内容，当泄露时则会暴露特征）
    - 有使用场景需要暴露系统分布特征而不需要高程度的透明性：基于位置的服务，系统的错误报告
2. 可以通过应用多种技术来取得扩展性，都有哪些技术 ？
    - 隐藏通信等待时间：
        - 异步通信技术
        - 设计分离的响应消息处理器：可以将消息识别解析分离，分发到相应的处理单元。系统需要扩展时，可以增加新的处理单元并将其连接到现有的消息分发网络。
        - 将计算从服务端移动到客户端：避免等待远程服务对请求的响应
    - 分布技术：将某个组件分割成多个部分，分散到系统中
    - 复制技术：在多个机器上创建多个数据副本，能够提高性能，增加可用性，促进组件间的负载平衡
3. 分布式系统的软件体系结构有哪几种类型，各自有什么优缺点?
    - 分层体系结构：
        - 优点：
            - 每一层之间相互独立，有利于每一层的测试和维护
            - 层与层之间保持层间接口关系不变，则层发生变化时，不影响其他层，有利于层的更新和修改
            - 结构简单，层的增加和减少简单
        - 缺点：
            - 层次之间有时候难以定义清楚
            - 当高层调用低层时，层与层之间的通信产生性能开销，响应速度降低
            - 层中可能有重复实现的功能和函数，导致冗余
    - 基于对象的体系结构：
        - 优点：
            - 对象封装了数据和操作，对外提供调用的方法但是隐藏了自己的详细信息，有利于对象的安全性和稳定性
            - 模块之间相互独立，提高灵活性，模块的修改更新更加简单
            - 通过对象继承，减少代码重复，提高代码复用性
        - 缺点：
            - 对象的动态内存分配和释放，可能会导致内存碎片问题
            - 对象之间的消息传递增加性能开销
    - 以数据为中心的体系结构：
        - 优点：
            - 所有组件使用相同数据源，避免数据冗余和不一致
            - 引用解耦，时态解耦

        - 缺点：
            - 增加数据的安全风险
    - 基于事件的体系结构
        - 优点：
            - 引用解耦合，组件不通过直接调用对方的接口通信，能使系统更加健壮（某个组件的故障不影响其他组件正常运行）和灵活
            - 时间上异步，提高系统的吞吐量
            - 可扩展性高，增加新功能时，定义新的时间和处理器，不用大程度更改原有代码
        - 缺点：
            - 需要控制事件的处理的顺序
            - 需要维护数据的一致性
            - 由于事件的异步性和不确定性，系统调试困难

4. 点对点网络中，并不是每个节点都能成为超级对等节点，
   满足超级对等节点的合理要求是什么？
    - 节点间距离：选择在网络中处于关键位置或传输路径上的节点，尽量减少其他节点和超级对等点之间的距离
    - 高算力：具备足够的能力处理各种请求和任务的节点
    - 高续航能力：在移动环境下需要节点有高续航能力
    - 具有稳定性，安全防护和高可用性
5. 代码迁移有哪些场景？为什么要进行代码迁移？
    - 场景：
        - 在客户端-服务器系统中，服务器要管理巨型数据库。客户端应用程序需要执行大量涉及大量数据的数据库操作时，需要将客户应用程序的一部分迁移到服务器上运行
        - 在交互式数据库应用程序中，客户需要填写表单，表单要被转换成数据库操作时，需要把服务器的一部分代码迁移到客户端。现在客户端处理好表单，再把处理好的操作发给服务器
        - 当更换硬件设备时，需要将代码从旧的硬件平台迁移到新的硬件平台上
        - 当需要优化系统架构时需要迁移代码构造新的系统架构
    - 为什么：
        - 为了性能的提升：让计算更加贴近数据段，最小化通信代价
        - 为了提高资源利用率：将进程从负载重的机器转移到负载轻的机器，能够更好利用空闲资源
        - 能够提高系统灵活性：可以方便动态配置分布式系统
6. 请从一些开源分布式软件如Hadoop、Ceph分布式文件系
   统、Apache Httpd、Spark等找出至少2处能够体现透明性的
   样例代码，并解释是何种类型的透明性
    - Hadoop MapReduce的位置透明性：用户只是制定了输入路径，输出路径，Mapper类，Redecuer类信息。但是没有指定数据如何分配到不同节点，分配的位置，这个由框架在底层完成

```
1  public class Dedup {
2    // 定义Dedup类
```

```
    // Mapper类，继承自Mapper接口，用于处理输入数据
    public static class Map extends Mapper<Object,Text,Text,Text>{
        private static Text line=new Text();
        public void map(Object key,Text value,Context context)
                throws IOException,InterruptedException{
            line=value;
            context.write(line, new Text(""));
        }
    }
    // Reducer类，继承自Reducer接口，用于处理Map的输出
    public static class Reduce extends Reducer<Text,Text,Text,Text>{

        public void reduce(Text key,Iterable<Text> values,Context
context)
                throws IOException,InterruptedException{
            context.write(key, new Text(""));
        }
    }
    public static void main(String[] args) throws Exception{
        Configuration conf = new Configuration();
        // 创建Hadoop配置对象
        conf.set("mapred.job.tracker", "192.168.1.2:9001");
          String[] ioArgs=new String[]{"dedup_in","dedup_out"};
        // 定义默认的输入输出路径
        String[] otherArgs = new GenericOptionsParser(conf,
ioArgs).getRemainingArgs();
        if (otherArgs.length != 2) {
            System.err.println("Usage: Data Deduplication <in>
<out>");
            System.exit(2);
        }

        Job job = new Job(conf, "Data Deduplication");
        // 创建Job对象，设置作业名称
        job.setJarByClass(Dedup.class);
        // 指定包含作业所需的所有类的jar包
        job.setMapperClass(Map.class);
        job.setCombinerClass(Reduce.class);
        job.setReducerClass(Reduce.class);
        // 设置Map、Combine和Reduce的处理类
        job.setOutputKeyClass(Text.class);
        job.setOutputValueClass(Text.class);
        // 设置输出数据的key和value类型
        FileInputFormat.addInputPath(job, new Path(otherArgs[0]));
        FileOutputFormat.setOutputPath(job, new Path(otherArgs[1]));

        // 设置作业的输入和输出目录
        System.exit(job.waitForCompletion(true) ? 0 : 1);
        // 提交作业并等待完成，根据作业是否成功完成返回相应的状态码
    }
}
```

- spark位置透明性：创建了一个rdd，无需知道rdd的物理节点，`map`操作会自动在数据所在的节点上执行

```
import org.apache.spark.{SparkConf, SparkContext}
```

```
2
3  object LocationTransparencyExample {
4      def main(args: Array[String]): Unit = {
5          val conf = new
   SparkConf().setAppName("LocationTransparencyExample").setMaster("loc
   al")
6          val sc = new SparkContext(conf)
7
8
9          val data = 1 to 10
10         val rdd = sc.parallelize(data)
11
12         val squaredRDD = rdd.map(num => num * num)
13
14         squaredRDD.collect().foreach(println)
15         sc.stop()
16     }
17 }
```