3 分布式系统中进程和虚拟化

Author： 中山大学 17数据科学与计算机学院 YSY

https://github.com/ysyisyourbrother

线程

分布式系统中的线程

基本思想

- 处理器： 提供和运行一系列指令集合的硬件平台
- 线程： 一个最小的可执行一系列指令的软件处理器。 保存 线程的上下文意味着终止线程当前的执行，在其他时刻装载 保存的线程上下文后，线程可以继续执行
- 进程： 包含多个线程的软件处理器，线程需要在进程的上下 文中执行

上下文切换

- 处理器上下文： 处理器用于运行一系列指令的保存在寄存器中的最小数据集合 （如：栈指针、地址寄存器、程序计数器）
- 线程上下文： 用于执行一系列指令的保存在寄存器和内存中的最小的数据集 合（如：处理器上下文、状态等）；
- 用于执行线程的保存在寄存器和内存中的最小的数据集合 （ 线程上下文、MMU寄存器值、TLB）

线程是共享相同的地址空间的。线程的上下文切换可以独立于操作系统。

一般来讲进程之间的切换要更复杂、代价更高，因为需 要陷入到OS内核才能完成

创建和销毁线程的代价要远远小于对进程的创建和销毁

为什么利用线程

1. 避免不必要的阻塞： 单线程的进程在进行I/O操作的 时候会被阻塞； 在多线程的进程中，操作系统可以 将CPU切换到进程的另外一个线程
2. 更好地发挥并行性：一个具有多线程的进程可以在多 核或者多处理器的CPU上并行执行
3. 避免进程上下文切换的昂贵开销，切换线程成本低。

上下文切换的代价

直接代价：用于实际切换和执行中断处理代码的时间

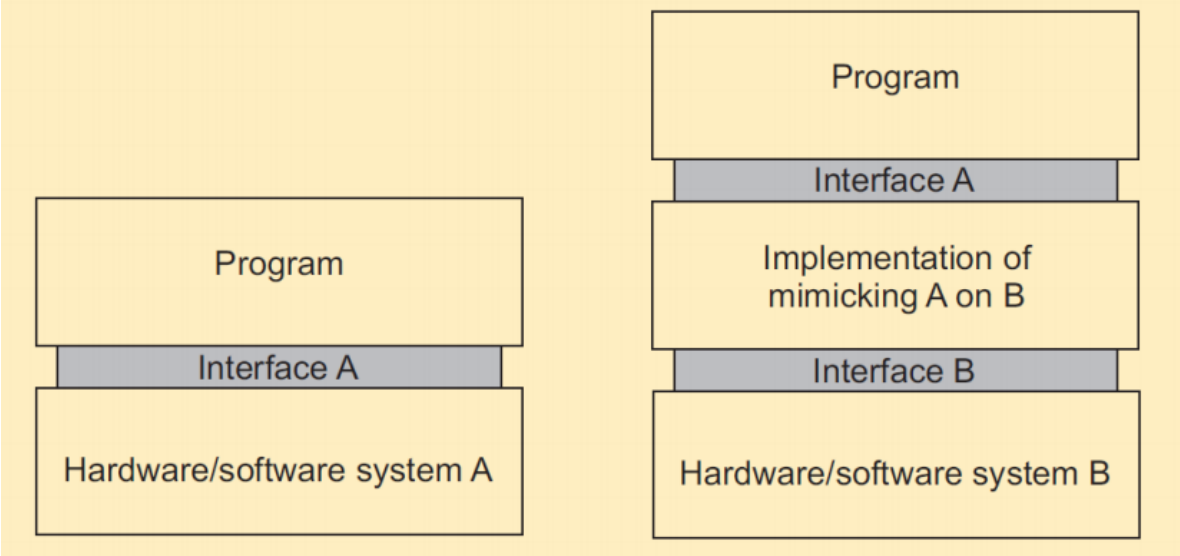
间接代价：其他代价，比较常见的是cache刷新的时间

虚拟化

虚拟化的本质：拓展或替换一个现有界面来模仿另一个系统的行为。

现在硬件和底层系统软件变化很快，我们常常面对一种情况是：旧的软件维护跟不上下层平台的更新步伐。通过移植旧有软件的底层接口到新平台，虚拟化可以帮助我们解决这个问题。

如下图，就在B的接口上，实现了A，运行本来只能在A上运行的程序：



模拟接口

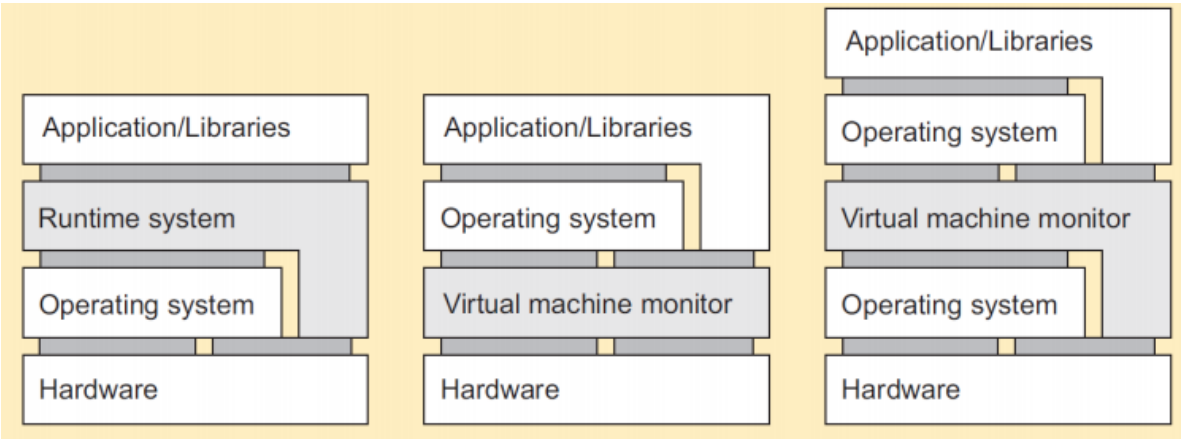
计算机系统通常在三个层次上提供的四种类型的接口：

1. 指令集架构：一系列的机器指令，主要分为两类
    - 特权指令：允许操作系统执行的执行；
    - 通用指令：可以被任何程序执行的指令
2. 系统调用： 由操作系统提供的函数
3. 库函数调用：也称为应用程序接口（API）

虚拟化的不同方式

1. 进程虚拟机（Process VM）： 构建一个运行时的系统runtime system，实际上提供一套抽象指令集来执行程序。实际是运行在操作系统之上的解释器（JVM）或者模拟器（Qemu）
2. 原生虚拟机监控器 (Native VMM)： 底层的指令，同时具有跑在硬件上的最小的操作系统 。虚拟机监控器安装在物理硬件上，将硬件拆分成多个虚拟机，在其上安装不同的操作系统
3. 主机虚拟机监控器 (Hosted VMM)： 底层指令，但是需要一个完整的OS 如：VMware

VMM是在底层实现对其上的虚拟机的管理和支持。以前的虚拟软件必须是装在一个OS上，然后再在虚拟软件之上安装虚拟机，装OS和应用。但现在英特尔 的CPU已经对虚拟化技术做了硬件支持，大多数VMM可直接装在裸机上，在其上再装几个虚拟机，这样就大大提升了虚拟化环境下的性能体验

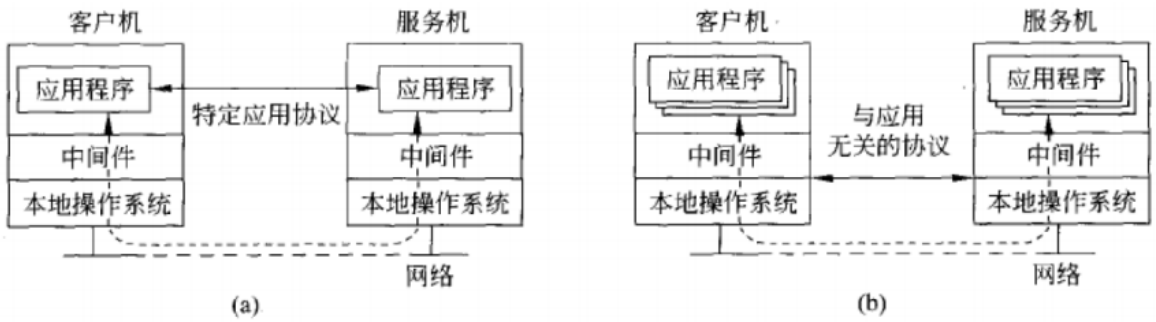


客户

网络连接的用户接口

客户机器主要是让个人用户和远程服务器交互。主要包含两种模式：

1. 对于每种远程服务，客户机都有一个独立的网络模块联系这 些服务。如图a所示，应用程序可以通过**特定的应用协议**来处理同步。
2. 通过一个方便统一的用户接口来对远程服务直接访问 。客户机只作为终端，不需要本地存储，如图b所示的不依赖于应用程序的方案。如果是这样的话客户端就可以把所有数据都在服务器上存储和处理，这就是**瘦客户**的方法，



服务器

迭代服务器： 服务器顺序处理到达的 请求

并发服务器： 利用分派器（dispatch er），选择到达的请求并将它传递给分离的线程或者进程处理

带外通信

在服务器已经接收了服务器请求后，是否有可能中断服务器的运行

解决方案1： 利用一个分开的端口用于紧急数据通信

- 服务器拥有一个分离的线程或者进程用于紧急通信
- 紧急信息到达 => 相关联的请求暂时挂起

解决方案2： 利用传输层的机制

- 样例：TCP 允许在同一个连接中传输紧急信息
- 利用OS的信号机制捕捉紧急信息

服务器和状态

无状态服务器

在处理完请求后，从来不保存关于客户端的精确的信息，也不告知客户自己的状态

- 不记录一个文件是否被打开（文件请求完成就关闭）
- 不保证清空客户端的cache
- 不去追踪客户的信息

结果

- 客户端和服务器完全独立
- 由于客户端或者服务器的宕机导致的状态不一致减少
- 由于服务器不能预测客户端的行为，可能导致性能下降

举例：web服务器

web服务器仅仅对输入http请求的时候做出响应，在处理请求的时候web服务器彻底忽略客户存在，同样 web服务器发生变化也不会告诉客户。

有状态的服务器

记录客户端的状态信息：

- 记录客户端打开的文件，这样可以提前实现预取
- 知道客户端缓存了哪些数据，允许客户端在本地保存共享数 据的备份

观察发现

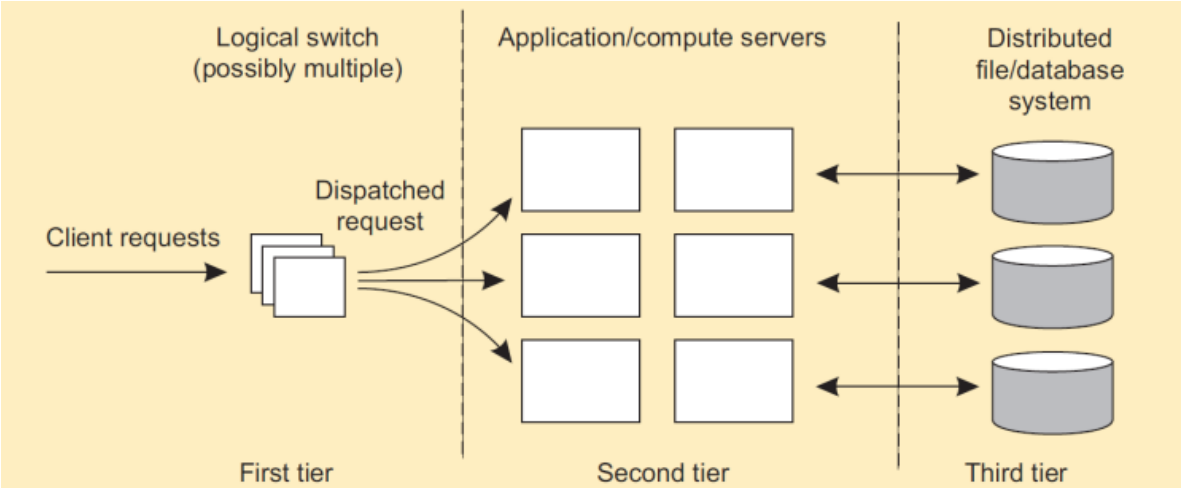
- 有状态的服务器的性能非常高
- 可靠性问题是一个主要的问题

举例：文件服务器

文件服务器允许客户保留文件的本地副本并更新，文件服务器需要跟踪目前哪些客户对那些文件有更新许可，并且哪里的文件版本是最新的，

服务器集群

多数情况下，服务器集群的逻辑由三层组成，第一层是一个交换机，由它分配客户请求给服务器。

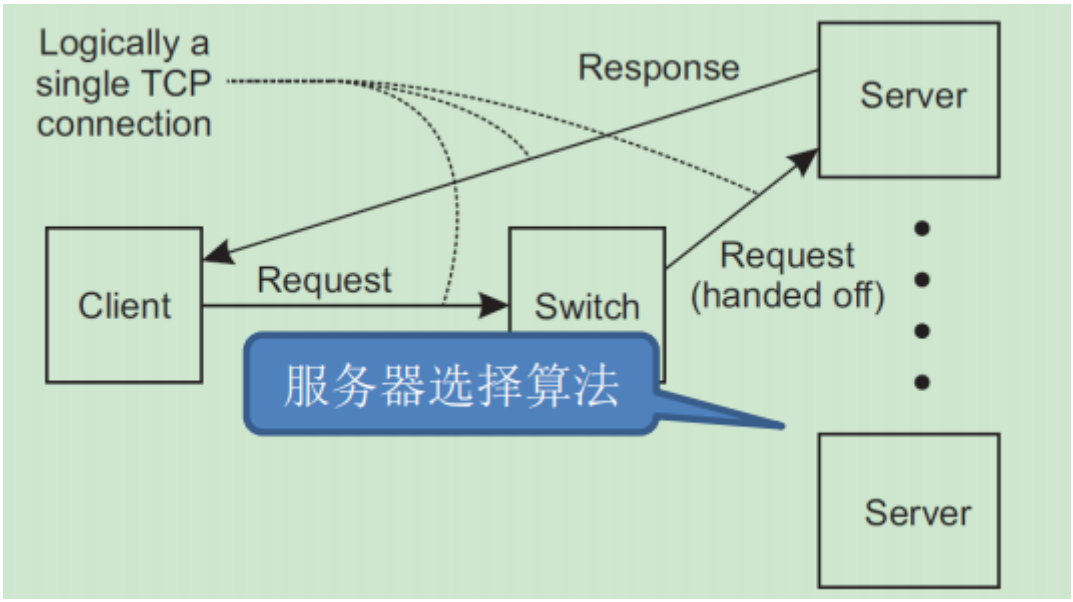


第二层是应用处理层，提供计算服务。

第三层是数据库服务器

请求处理

让第一层处理所有的出入集群的通信会导致性能瓶颈；



为了可拓展性和可用性，服务器集群可以能有多个访问点，每个访问点由一个独立的专用机器实现。

当交换机收到一个TCP连接请求后，它随机处理找到最优的服务器，并转发请求包给服务器，服务器反过来发送一个应答给请求的客户，这里要注意：**TCP数据段中返回的是交换机的IP地址**

负载均衡

交换机对多个服务器之间分配负荷起重要作用。

- 最简单的可以使用轮转法
- 基于传输层交换方法：前端简单地将TCP请求传递到服务器，只 是会考虑某些简单的性能指标如CPU利用率等
- 基于内容可感知的分派方法：前端会读取请求的内容，然后选择 最合适的服务器

代码迁移

负载分布

分布式系统中的代码迁移以**进程迁移**的形式进行。这种形式下整个进程被从一台机器搬到另一台机器上、这是一项复杂与开销庞大的任务。如果将进程由负载重的机器转移到负载轻的机器上，就可以提升系统整体性能。

1. 确保数据中心中的服务器的负载运行更加充分
2. 让计算更加贴近数据端，最小化通信代价

代码迁移模型

在代码迁移模型中，进程由如下三段组成

1. 代码段：包含构成正在运行的程序的所有指令
2. 资源段：包含指向进程需要的外部资源的指针，如打印机等
3. 执行段：存储进程当前执行状态量，比如寄存器、栈等

弱可移动性： 仅仅移动代码和数据片段（**重启执行**）

1. 相对简单，特别是如果代码是可移植的 （目标机器能够运行代码）
2. 需要区分两种模式：代码推送（Push）和代码拉取（Pull）

强可移动性： 移动组件，包括执行状态 ，可以从中断位置开始执行

1. 迁移（Migration）：将整个对象从一个机器移动到另外一个机器
2. 克隆（Cloning）：开始克隆，将其设置为相同的执行状态

异构系统中的迁移

1. 目标机器可能不适合执行迁移后的代码
2. 进程/线程/处理器的上下文的定义比较依赖于硬件、操作系统和 运行时系统

可以通过虚拟机结局：VM或VMM

迁移虚拟机

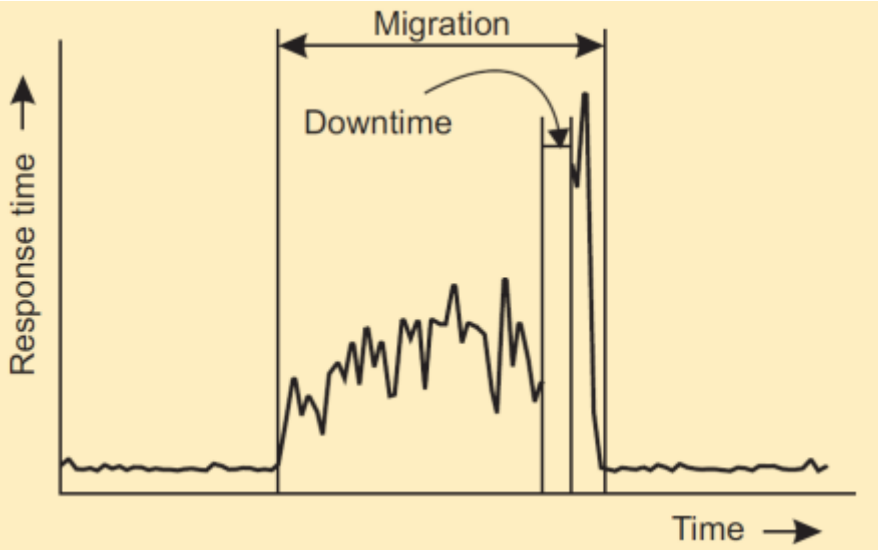
虚拟化操作系统给通过一个共享局域网实现紧密耦合的服务器集群提供了方便

迁移涉及两个问题：迁移**整个内存映象**和迁移**对本地资源的绑定。**

第一个问题有三种结局方案：

1. 将内存页面送给新机器并重发在迁移过程中修改了的页面
2. 终止当前虚拟机，迁移内存，然后重启虚拟机
3. 让新的虚拟机按需拉取内存页面：在新的虚拟机上立即创建进程， 并且按需拷贝内存页面

一次完整的虚拟机迁移可能需要几十秒。在迁移期间，服务可能处于 几秒钟的完全不可用的状态



结合第一种和第二种方案，提出了一个预复制的方法