中国科学: 信息科学 2012 年 第 42 卷 第 3 期: 253–263
www.scichina.com info.scichina.com
《中国科学》杂志社
SCIENCE CHINA PRESS
专 题

Cloudow: 一种基于用户层虚拟化的软件即服务模式运行系统

张悠慧*, 李艳华, 郑纬民

清华大学计算机科学与技术系, 北京 100084
* 通信作者. E-mail: zyh02@tsinghua.edu.cn

收稿日期: 2011–08–29; 接受日期: 2012–01–05
国家重点基础研究发展计划 (批准号: 2007CB310900) 、教育部 –Intel 信息技术专项科研基金 (批准号: MOE-INTEL-11-04) 和北京邮电大学智能通信软件与多媒体北京市重点实验室开放基金项目资助

摘要 “软件即服务 (SaaS)” 是一种通过网络发布与使用软件的新模式, 在很大程度上消除了用户购买、维护与升级应用程序的需要, 被认为是软件未来的主流应用模式之一. 本文提出了一种新的支持现有 Windows 桌面软件的 SaaS 模式并实现了其原型系统 Cloudow: 用户可以在任意的联网兼容计算机上按需运行现有的 Windows 软件 (无需安装) , 且软件的个性化配置可以被保留以便下次使用时恢复. Cloudow 使用用户层虚拟化技术解决了软件无需安装便能运行的问题, 并通过用户层文件系统设计实现了软件在网络环境下的透明使用. 与现有的基于远程虚拟机计算或者基于 Web 应用的 SaaS 模式相比, Cloudow 能够直接支持现有软件的服务端存储/客户端运行模式, 无需修改代码, 较好地兼顾了软件兼容性与性能. 同时, 为尽可能降低 Internet 环境所带来的远程数据访问延迟, Cloudow 大量采用了元数据/数据/文件预取与缓存策略, 显著提高了实际部署中的应用性能; 测试表明, 因为采用了这些优化策略, 对于很多常用的 Windows 桌面应用而言, 在 Cloudow 下额外运行时间开销平均为 12% 到 20%.

关键词 软件结构 面向服务架构 软件即服务 用户层虚拟化 用户层文件系统

1 引言

软件即服务 (software as a service, SaaS) 是一种通过网络部署与使用软件的模式. 与传统的软件单机安装/使用方式相比, SaaS 可以集中地由相关运营商来管理、部署和升级软件. 根据 Gartner 的研究 [1], 在商用领域, PC 的软硬件购置成本已不是其拥有成本 (total cost of ownership, TCO) 的决定性因素. 因此, SaaS 模式能够在很大程度上降低使用方的维护与管理难度, 大大降低其拥有成本.

SaaS 模式最早应用于商业软件领域, 如 SalesForce 公司提供基于 Web 的 CRM 软件, 提供商将应用软件统一部署在自己的服务器上, 客户可以根据实际需求, 通过互联网向提供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向厂商支付费用. 随着云计算模式的逐步推广, 基于 Web 的应用也拓展到个人桌面应用领域.

引用格式: 张悠慧, 李艳华, 郑纬民. Cloudow: 一种基于用户层虚拟化的软件即服务模式运行系统. 中国科学: 信息科学, 2012, 42: 253–263, doi: 10.1360/112011-931

另一方面, 通过远程桌面访问协议使用远程虚拟机应用也成为云计算领域提供按需服务的一种方式. 如 virtual appliances 技术 [2,3], 其将虚拟机映像文件与一定的软件应用绑定在一起, 提供基于云计算的按需软件运行环境, 使得用户可以在定制的远程虚拟机上快速部署服务方提供的软件.

在基于远程桌面访问协议的模式下, 应用存储并运行于服务端, 因此可以直接使用现有的桌面软件; 但在 Internet 环境下, 网络传输所带来的操作延迟对于桌面应用而言是致命的 (带宽的增加不能明显改善这个问题) , 且客户端只是作为图形终端使用, 软件集中运行在服务端. 在 Web 模式下, 应用存储于服务端, 运行于客户端的浏览器上 (或者是 Flash Player 等第三方插件) , 这一模式的运行性能较好, 但是无法直接使用现有的桌面软件. 因此, 实现一个既能够直接支持现有桌面软件, 又能够在客户端本地直接运行的 SaaS 模式是非常有吸引力的.

本文提出了这样一个模式, 设计并实现了其运行系统 Cloudow. Cloudow 的特点在于, 用户可以在任意的联网 Windows 计算机上按需运行现有的 Windows 软件 (无需安装) , 且软件的个性化配置可被保留以便下次使用时恢复. 也即提供了一个兼容已有的应用软件二进制代码、采用桌面应用模式的充分利用客户端主机处理能力的 SaaS 模式. 当然, 前两种模式的一个优势在于可以跨平台运行, 而 Cloudow 依赖于 Windows 客户端. 但是, 目前 Windows 仍是桌面系统的主流, 因此研究针对 Windows 的 SaaS 模式是具有重要意义的.

从技术角度来看, 本文做出了如下贡献:

1) 在以前工作的基础上 [4,5], 设计了多进程共享的用户层虚拟化运行时环境, 解决了 “软件无需安装便能按需运行” 的问题 (本文将此类软件称为 “按需软件”) .

2) 采用用户层文件系统方案, 使得在服务端的按需软件存储位置可被加载成本地的虚拟文件系统, 从而能够以完全透明的方式为软件本身提供运行环境. 同时, 该用户层文件系统大量采用了元数据/数据/文件预取与缓存策略, 来提升运行性能.

3) 原型系统的测试表明, 因为采用了优化策略, 对于很多常用的 Windows 桌面应用而言, Cloudow 下软件运行时的额外开销平均为 12% 到 20%.

2 系统设计框架

Cloudow 设计所要解决的核心问题有两个: 一是解决 Windows 桌面软件无需安装就能运行的问题, 这是实现 SaaS 模式的前提. 二是解决在网络环境下的透明使用问题.

针对问题一, 本文提出了一个针对软件应用的按需运行模型, 以其为基础设计了按需软件的生成与运行时环境, 并在本节做扼要介绍.

对于问题二, Cloudow 采用了用户层文件系统. 与之前的设计 [4,5] 相比, Cloudow 重新实现了整个用户层文件系统, 引入了 “锚文件系统”、文件级预取, 并重写了本地缓存实现. 本节对这些方面做了概要介绍, 具体的性能优化设计在第 3 节给出.

2.1 软件按需运行模型

一般而言, 一个软件运行所关系到的软件资源 (这儿的资源一般指文件与注册表) 可分为 3 个部分:

1) 第 1 部分指的是所有由本地操作系统提供的资源, 包括一些公用的系统运行时库、目录结构与注册表项等;

2) 第 2 部分包含软件安装过程所创建、修改的资源;

3) 软件运行过程中会创建新资源或者修改 1) 与 2) 的资源, 这些被归为第 3 部分.

在 virtual appliances 的模式下, 这 3 部分都通过硬件层虚拟化技术与整个 OS 耦合起来, 使得其只能实现整个运行环境与硬件的隔离; 而 Cloudow 通过用户层虚拟化进一步将应用与 OS 隔离, 提高了软件部署的灵活性.

基于此模型, 解决问题一主要包含两方面工作: 1) 如何分离出这 3 个部分; 2) 如何使之可按需访问.

因为假设用户是在兼容的计算机上使用 Cloudow, 所以默认操作系统可以提供第 1 部分资源, 无需对其做特殊处理; 对于第 2 部分, 采用软件安装监控的方法 —— 在一个新安装的 Windows 系统上安装目标软件, 记录下安装过程所创建/修改的资源并将其单独存储, 从而可以获得完整的第 2 部分; 对于第 3 部分, 使用系统调用插装的方法实现用户层的虚拟化运行环境, 从而可以实时截获软件运行实例的系统访问调用, 并将调用重定向到资源实际存储的位置, 具体的处理原则是:

1) 所有读取操作都在存储的当前位置完成;

2) 修改操作则采用 COW 机制, 将修改后的资源存入第 3 部分;

3) 浏览或者枚举操作则分别列出 3 个部分的相关内容并进行合并; 如果有重复项, 其优先级从高往低依次是: 第 3 部分、第 2 部分、第 1 部分.

这方面的详细设计可见文献 [4,5], 这儿具体介绍 Cloudow 在运行时环境设计中的改进.

2.2 运行时环境

Cloudow 设计了一个多进程共享的用户层虚拟化运行时环境, 而我们之前的设计思路则是每个按需软件运行实例都有一个独立的运行时环境. 其问题在于: 1) 各个独立的运行时环境无法共享数据, 从而使得多个同时运行的按需软件无法通过文件系统或者注册表共享信息或者进行信息交换; 2) 各个运行时环境间有冗余信息 (如公用的运行时库、注册表项等), 消耗了更多的运行资源.

因此, Cloudow 基于远程过程调用 (RPC) 设计了多进程共享的用户层虚拟化运行时环境. 在客户端 Cloudow 系统启动后, 其自身就作为一个 RPC 服务端读取所有按需软件的第 2, 3 部分, 然后等待接收来自于 RPC 客户端 (即任意一个按需软件的运行实例) 的请求.

以注册表资源访问为例, 对于任意一个 Windows 注册表相关的系统调用, Cloudow 都实现了一个对应的 RPC 的客户端接口, 以动态代码插装的方式将其注入按需软件的运行空间, 对原有的系统调用进行包裹. 当第 2, 3 部分的资源被访问时, 该接口就被调用, 而相应的 RPC 服务端则进行处理以便提供一个一致性的资源视图.

从性能的角度来看, 基于 RPC 的共享运行环境会带来更大的性能开销 —— 因为 RPC 自身引入了进程间数据交换, 而在原先的方案中所有的操作都在一个进程内完成. 所幸这些资源访问操作自身的频度通常只占桌面软件操作的很小一部分, 性能测试表明, 这一机制所引入的软件运行时开销不到 2%; 而原先模式的开销为 0.5%.

2.3 基于用户层文件系统的按需软件透明访问

一个较为理想的透明访问模式应该具有如下特点:

1) 对于使用者与软件自身透明. 用户应该能够像启动本地已经安装的其他软件那样来启动按需软件, 这样在使用体验上是最优的.

2) 性能优化. 按需软件是远程存储, 本地运行; 因此必须尽可能地减少远程访问的频度.

Cloudow 设计中采用了用户层虚拟文件系统方法, 将远程按需软件的存储位置加载为本地的一个虚拟文件系统, 这种方式对于使用者与软件而言都是透明的. 同时采用了元数据/数据/文件预取与缓存策略来优化性能.

用户层文件系统一般是指在系统内核空间插入文件系统过滤驱动, 将目标驱动器的相关文件系统访问重定向到用户空间, 由用户层文件系统程序来完成实质的数据/元数据访问与操作. 这种方式的优点在于其开发工作可以避开复杂的内核编程而在用户空间实现, 而且与操作系统的耦合度较小; 缺点在于会引入更多的上下文切换, 性能上有一定损失.

Cloudow 用户层文件系统的总体运行策略是:

1) 读操作会被定向到远程位置以获取数据 (该过程中本地缓存与各类数据预取机制将会起作用);

2) 对于任何写操作, COW 机制被触发: 整个目标文件将会从远程被取回来, 缓存于本地, 后续操作会直接针对这个本地版本进行.

与我们以前的工作相比 [6], Cloudow 重新实现了整个用户层文件系统, 引入了 “锚文件系统”、文件级预取, 并重写了本地缓存实现.

3 用户层文件系统与性能优化

3.1 整体流程

任何一个存储于该用户层虚拟文件系统之上的文件都被赋予一个属性. Remote: 该文件存于远程服务端; New: 该文件是新建的或者是被修改过的; Deleted: 已被删除.

当某个用户的文件系统第 1 次被启动时, 所有的文件与目录都被认为具有 remote 属性, 即存储在服务端. 在运行过程中, 当任何文件/目录被删除时, 这些文件/目录属性会被设为 deleted (而不是物理上被删除). 任何修改或者新创建的文件则会被置为 new. 特别地, 当某一远程文件被修改时, 它会首先被下载到本地缓存, 后续操作将针对此本地版本进行.

用户层虚拟文件系统的整个运行过程如下所述:

1) 系统初始化. 用户选择某一按需软件部署后, 虚拟文件系统首先下载这个按需软件的元数据包. 后者包括该软件的所有文件、目录的层次结构信息、路径名、以及文件属性 (大小、访问权限、时间戳等). 因为服务端实际上是个只读存储, 这类元数据包可以事先在服务端压缩好, 以随时访问. 原型系统的测试表明, 元数据包压缩后的尺寸非常有限: 系统中的所有按需软件共有约 11600 个文件 (包括目录), 包压缩后的大小为 190 KB.

下载完成后, 虚拟文件系统程序会在本地某个事先指定的位置创建一个 “锚文件目录” (anchor folders): 在这一目录下, 按需软件所有的文件 (包括目录) 均会被创建, 其目录结构、名字、文件属性均与服务端的一样; 唯一的区别在于所有的 “锚文件” 都是空的.

与前一工作比较, 这一设计的优点在于所有与文件元数据相关的操作 (如 SetFileAttributes/GetFileAttributes/SetFileTime/FindxxxFile 等) 可以直接在 “锚文件系统” 上完成, 使得虚拟文件系统程序专注于处理数据操作; 同时, “锚文件” 也可以直接作为相应文件的缓存以简化其管理.

2) 运行时. 总的设计原则是: 服务端的内容为只读; 客户端采用 copy-on-write 机制, 即任何用户使用软件过程中的修改都存于本地的锚文件系统, 包括新文件/新目录以及修改的原有远程文件.

3) 系统退出. 用户层文件系统退出时, 所有的 “锚文件系统” 内容以及所有的文件属性都会被保存以便下次使用.

4) 系统迁移. 当用户欲在另一台 Windows 计算机上启用 Cloudow 系统时, 他/她可以先将所有的元数据包、“锚文件系统” 里的非空文件、以及所有的文件属性信息拷贝出来 (比如放置到网络硬盘上), 以便在目标机上恢复最新的个性化虚拟文件系统视图.

5) 软件升级. 远程服务器上的软件升级后, 会生成新的元数据包. 为降低数据传输量, 新数据包里存的是两个版本间的差异. 这样当客户端虚拟文件系统启动时, 会根据这个新数据包的内容修改 “锚文件系统”, 而且任何更新文件的本地缓存会被置无效.

3.2 远程访问与本地缓存

在 Cloudow 当前的实现中, 远程存储就是一个普通的 HTTP 服务器: 所有的数据读取, 包括元数据下载、文件读取与 COW 机制都被转换为 HTTP 的 GET 请求. 选择 HTTP 作为后端主要是出于实现简便性考虑, 而不是设计上的必须, 即也可以使用其他的数据发布机制作为服务端存储.

为管理方便起见, 所有远程文件访问的数据大小与起始位置都被设为 8 KB 的整数倍, 而所获取的数据块 (8 KB 的整数倍) 都将被缓存在本地.

本地缓存 (锚文件目录) 保存有 2 类数据, 一是具有 new 属性文件的数据 (即属于本地修改的或者新创建的文件); 二是具有 remote 属性文件的缓存于本地的数据. 第 1 类数据实际上就是一个完整的文件, 可以对其进行直接的文件层次访问; 重点是第 2 类数据 ——Cloudow 维护了单独的缓存信息, 以文件的全路径名为键值, 建立一个 Hash 表, 来记录哪些文件的哪些数据块已被缓存. 进一步地, 每个文件的历史访问信息都被记录下来. 当本地缓存的总容量接近系统预设的上限时, 会向用户提出警告以扩大容量, 或者采用 LRU 策略来删除文件以腾出空间; 但是第 1 类数据是不会被删除的.

3.3 文件预取

在元数据预取之外 (即事先获取元数据包), 针对按需软件 IO 访问行为特征的分析, Cloudow 设计了一个文件级的数据预取机制. 研究发现, 一个程序的启动与运行期间的文件访问特征是不同的, 针对这两个不同阶段, Cloudow 采用了不同的优化策略.

3.3.1 启动期间

在这一阶段, 按需软件的执行文件与相关的动态链接库被映射到系统虚存中, 同时会因为大量的缺页中断而产生密集的文件读取操作. 通过对于这些文件访问踪迹的分析, 我们观察到了下列现象:

1) 无论文件的启动方式如何 (是远程按需启动, 还是本地直接启动), 文件访问特征是一致的;

2) 在启动期间, 只有有限的软件数据被访问, 测试表明, 这个比例占软件整体大小的 1/3 左右. 对较大规模的软件而言, 这个比较更低;

3) 对于运行期间的一些常见软件操作, 其所访问的软件数据大部分属于启动过程中被访问过的数据, 其他数据平均只占 12%;

4) 在这一期间, 踪迹显示任一个文件被打开后几乎立即就是该文件的第一次读取 (相隔不超过 2 μs); 而连续的文件打开操作间的间隔也很短, 大约为 107 μs.

明显地, 第 4 点使得运行时的文件级预取几乎成为不可行. 因此, Cloudow 引入了一种 “推送” 方式: 一旦用户订阅了某种 (些) 按需软件, 系统就自动地将相关的文件数据下载到本地缓存中, 而且不

同文件数据的下载优先级是不同的 —— 那些在软件启动过程中被访问到的数据将会优先下载, 跟着的是那些在运行过程中被访问到的数据. 这一策略不但提高了软件启动/运行的速度, 而且可以有效降低下载数据量.

而如何区分这几类数据的方法是很直观的: 基于第 1 点, 对于任何一个按需软件, 可以记录其启动过程的 IO 踪迹从而获得相关的数据信息. 而对于运行时数据, 我们倾向于采集常用软件操作的文件访问踪迹: 即通过分析普通用户的运行时记录来获得较为精确的数据集. 目前, 我们只是通过有限几位志愿者的数据来获得这些信息.

3.3.2 运行期间

这一期间的文件访问特征很大程度上依赖于使用中的具体操作类型, 而且因为桌面软件会有大量的操作间隔 (如使用者的思考时间), 所以文件预取有可能被用来掩盖运行过程中的 IO 操作延迟.

踪迹分析也证实了这一点 —— 运行过程中 (启动后) , 连续的文件打开的间隔较之第一阶段延长了很多, 平均约为 700 μs. 因此, 我们采用了一种自动的预测机制 [7] 来进行文件级预取, 其关键问题是如何根据访问历史来构建预测模型.

基于运行时的访问踪迹, 我们构建了一个 Probability Graph[7]. 图中的每个结点表示一个文件, 而结点间的有向边则表示其连接的两个文件是相关的, 意味着一个文件是在另一个被打开后的一个阈值区域 (lookahead) 内被打开: 比如 lookahead 为 5 就意味着某一文件打开后, 接连打开的 5 个文件都被认为是与其相关的. 每一条边的值则被标注为在踪迹提取过程中这个相关文件被实际打开的次数 (在前一个文件被打开后). 这样, Probability Graph 就可以用来表示文件打开关系间的相关性与权重.

进一步地, 某个文件 B 在 A 后被打开的可能性可以被看作是从 A 到 B 这条边的权值除以由 A 出发的所有边的权值之和所得的商. 这样就引入了另外一个重要参数, minchance: 当这个可能性的值超过了 minchance, B 就会被预取 (在 A 被打开时).

预取机制的实现设计也是很直观的: Cloudow 能够获取虚拟文件系统上所有的文件打开操作, 然后基于事先构建好的 Probability Graph 来进行文件预取 (在满足 lookahead 和 minchance 参数设定的前提下), 取来的数据存放于内存中的预取缓存, 其中被真正访问到的数据将被进一步移至本地缓存中.

从性能的角度来看, 3 个参数, lookahead、minchance 与缓存大小, 对于预取性能有着直接影响. 较大的 lookahead 值扩大了预取范围, 因此比较适用于较大的 cache 尺寸; minchance 则标示了了文件在一定时限内被实际读取的最小可能性. minchance 越小, 适用的 cache 越大.

在当前的设计中, lookahead 被设为 10, minchance 是 0.3, 预取缓存的大小为 16 MB.

4 原型与测试

4.1 原型系统

Cloudow 可以将常用的 Windows 桌面软件透明地转换为按需软件. 在大学校园内部署了这一系统原型, 该原型包含下列主要部分:

1) Apache HTTP 服务器, 存储所有的按需软件;

2) 安装有 Cloudow 客户端程序的多台 Windows PC 与笔记本, 用户可以通过该程序使用这一系统; 每个客户端以千兆以太网或者 802.11g 无线网络连入校园网; 该客户端是系统的主要部件, 其本身

包括如下主要模块: 用户层虚拟文件系统程序、按需软件的用户层虚拟化运行时环境、本地缓存管理模块、文件级预取模块、Shell 程序.

4.2 测试方法

本测试中使用了 11 个不同的按需软件, 它们在不同环境下的启动时间与运行时间被分别记录下来以作比较.

软件启动后, 采用了模拟使用者键盘鼠标输入的方法来操作这些软件以获得其运行时间, 步骤如下:

第 1 步: 几位志愿者被请来使用这些软件, 同时采用鼠标/键盘记录程序来记录他们的所有操作, 包括操作之间的间隔.

第 2 步: 测试时, 这些操作记录被回放以模拟人为操作, 所有操作的完成时间 (在不同测试环境下) 被记录下来作为软件的运行时间. 需要指出的是, 记录操作是在软件存储在本地的时候做的, 而在测试环境下, 因为软件反应速度会减慢, 所以各个操作间的间隔需做一定的调整以确保所有操作都能够完成.

测试用客户端安装 Windows 7 系统, 配备有 2 GB 内存, 一个 Intel I5 处理器 (2.53 GHz 主频), 以及千兆互联网. 硬盘 160 GB 的 SATA 盘.

有两台服务器 (HTTP server) 被部署. 第 1 台位于校园内 (但不是与客户端位于一个大楼), 两端的网络实测流量约为 1.96 Mbps, 平均网络响应时间是 5 ms. 该服务器是一台 Windows 2003 server, 配有一个 Intel Core 2 Duo E4500 处理器 (主频 2.2 GHz), 2 GB 内存, 240 GB SATA 盘; 第 2 台位于外网, 但仍在中国教育科研网内 (CERNET) 内, 配置与第 1 台相同. 我们认为, 这一部署方式是符合实际情况的: 因为随着 CDN (content delivery network) 技术的发展与推广, 一般可以在某些 edge server 上发现需要的内容.

4.3 测试用例 1: 用户层虚拟化的开销

包含有 3 个测试用例:

(1) 基准测试: 软件安装在本地, 且没有使用虚拟化技术, 直接使用;

(2) 软件在用户层虚拟化环境中运行 (即按需软件), 且所有的文件都存于本地磁盘 (即不采用虚拟化文件系统);

(3) 软件在用户层虚拟化环境中运行, 应用了虚拟化文件系统, 但是本地缓存的命中率设为 100%, 即测量虚拟化文件系统自身引入的开销 (排除网络互联的因素).

各个测试用例下软件的启动与运行时间被分别记录下来以作比较, 具体结果见图 1. 可以看到, 用户层虚拟化运行环境自身引入的开销非常小: 启动过程约为 2.0%, 运行过程则几乎可以忽略 (比较测试用例 1 与 2).

因为会引起较多的运行状态切换, 用户层文件系统引入了较大开销: 与虚拟化运行环境一起, 此两者几乎使得软件启动过程的时间加倍, 约为基准时间的 1.8 倍; 但运行时间的额外开销很小, 约为 7%, 这是因为桌面软件运行过程中有大量的思考时间, 使得这一因素的影响得到相对降低.

4.4 测试用例 2: 校园内系统运行测试

包含有两个测试用例:

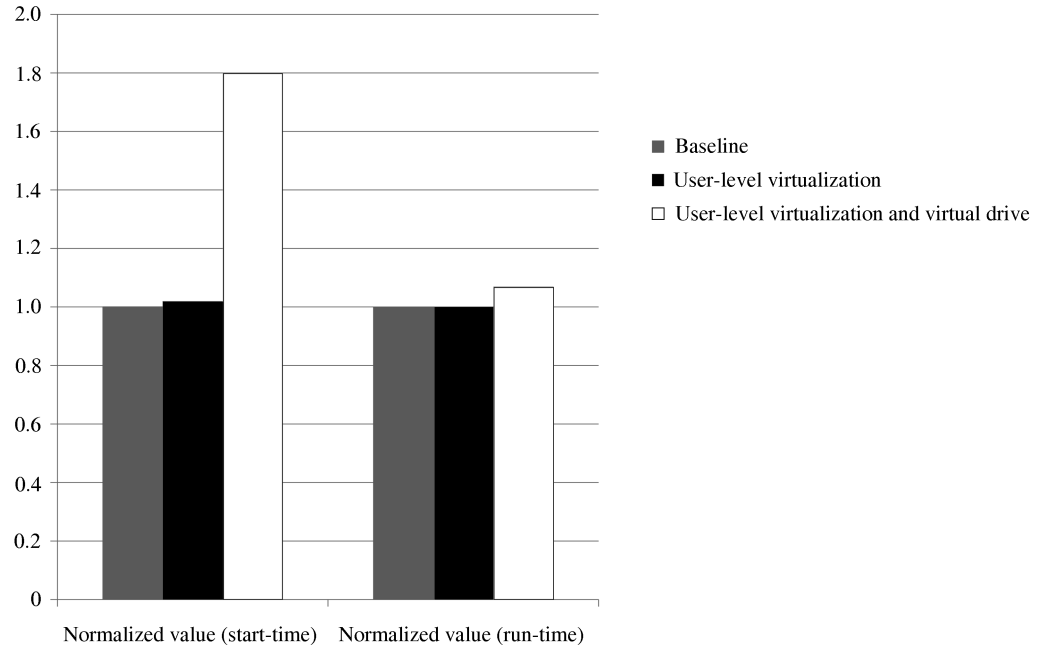


图 1　用户层虚拟化的性能开销

Figure 1　Performance overheads of user-level virtualization

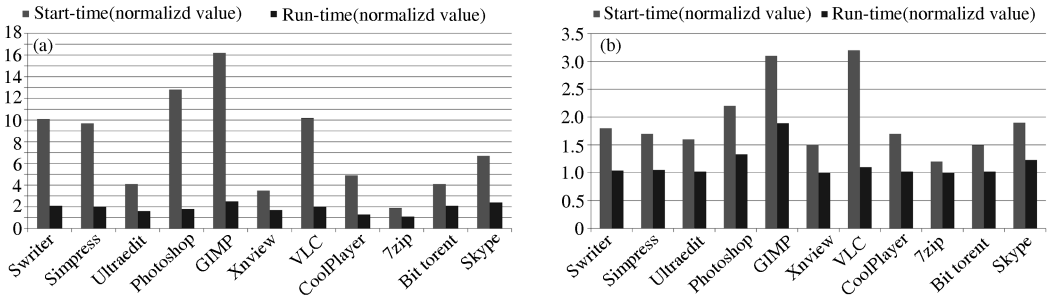


图 2　校园网内运行性能

Figure 2　Running performance inside the campus. (a) The 1st time; (b) the 2nd time

(1) 本地缓存为空 (相当于是这个系统第 1 次运行). 启动与运行时间被分别记录下来并与基准测试进行比较 (见图 2(a)).

可以看到, 在这个用例中软件启动时间被大大延长: 平均是基准时间的 7.65 倍. 这是因为本地缓存为空, 导致大量的虚拟文件系统访问被转换为远程 HTTP 访问, 而且文件预取在这一过程中几乎无法起作用 (见 3.3 小节). 为解决这一问题, 我们在第 3 节引入了软件推送模式, 以期提升这一过程的速度.

而在运行测试中, 这一状况好了很多: 运行时间平均被延长约 87%. 这是因为启动过程缓存了大量数据, 而运行过程中需要访问的大部分数据是属于这一部分被缓存数据的. 而且思考时间与预取机制又可以掩盖一部分网络访问延迟, 使得总体的影响大大下降.

(2) 第 1 次运行后, 再次运行一遍测试并记录时间 (相当于在缓存已填充的情况下运行, 结果见图 2(b)).

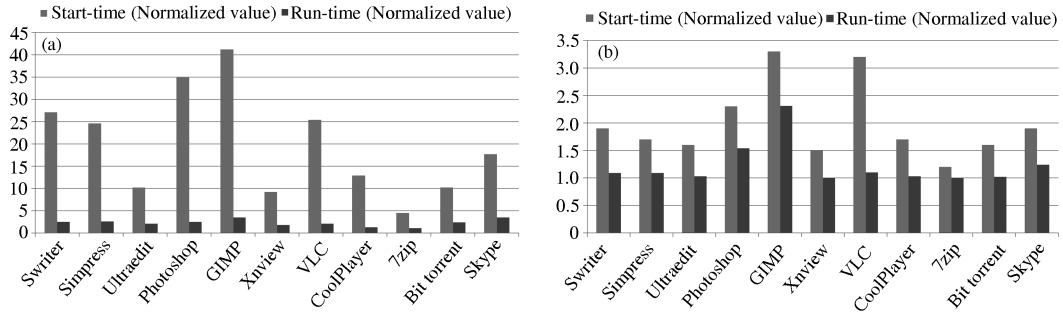


图 3 校园网外运行性能

Figure 3 Running performance outside the campus. (a) The 1st time; (b) the 2nd time

这一测试的情况好了很多: 启动时间被延长了约 90%; 而运行时的额外时间开销仅为 12%. 这明显归功于第 1 次运行已将大部分所需数据填入了本地缓存, 使得远程数据访问次数大大降低.

另外, 第 1 次运行后本地缓存的容量为 260 MB, 而所有测试软件的总的大小为 900 MB.

4.5 测试用例 3: CERNET 内系统运行测试

具体测试方法与上一个用例相同, 唯一的不同是使用了位于外网的第 2 台 (仍在 CERNET 内) 服务器.

测试结果见图 3. 可以看到, 如果本地缓存为空, 那么软件启动时间平均是基准时间的 19.8 倍, 这是因为访问外网的时延更长. 第 1 次启动后运行时间的额外开销平均约为 126%.

第 1 次运行后, 再次运行的情况要好很多: 启动时间延长了约 97%, 而运行时间延迟约 20%, 原因与上一测试用例相同.

5 相关工作

多种虚拟化技术已被用来将传统软件转换为这种按需使用的模式, 这些技术可以并分为两类: 基于虚拟机模式与基于用户层虚拟化模式.

第 1 种类型的一个早期代表是 ISR (Internet suspend/resume)[8,9]: ISR 使用虚拟机技术 (如 VMWare Workstation) 以及网络文件系统技术, 使得可以通过互联网来随时恢复用户的工作环境. 类似的一个商业化案例是 MokaFive LivePC[1)]. 一个 LivePC 实例包含有一个完整的虚拟机, 包含操作系统及一系列的应用软件; 同时其允许用户安装个人软件. 用户可以通过网络服务器来访问其个性化的虚拟机映像文件, 并通过本地缓存与增量传输模式来降低系统启动过程中的数据传输量.

一些云计算服务提供商 [10~14] 在数据中心内通过互联网为客户提供按需的虚拟软硬件环境服务. 绝大多数采用 VA 技术来提供这类服务, 因为 VA 是基于虚拟机技术的, 客户只能在服务方提供有限的虚拟机镜像文件中选择应用程序.

第 2 种类型而言, IBM 的 PDS (progressive deployment system) [15] 是一个早期的研究实例. PDS 也采用了截获相关系统调用的方法来实现虚拟运行环境, 适用于局域网内的部署.

1) Http://www.mokafive.com/

另外一个产品是微软的 Application Virtualization[2)], 可以按需求将程序包发布到客户端, 支持客户端主动升级应用程序包, 适用于域架构下的具有高可靠、高带宽的本地客户端.

我们的前期工作 [4,5] 支持将现有的 Windows 软件转换为网络服务, 这些工作的成果被直接用于本研究, 并进行了优化, 包括多进程共享虚拟运行环境, 重新设计实现了整个用户层文件系统, 引入了“锚文件系统”、文件级预取, 并重写了本地缓存实现等.

6 结论

本文提出并实现了一种基于用户层虚拟化的支持现有 Windows 软件的 SaaS 模式的技术方案. 在使用者看来, 他/她可以订购按需软件, 并以使用本地软件的方式直接使用这些远程存储、本地运行的桌面软件. 而且, 用户可以在不同的兼容计算机上, 恢复这些按需软件的个性化配置 (包括这些软件的元数据、COW 文件、新建文件以及缓存数据).

参考文献

1 Troni F, Silver M A. Use Processes and Tools to Reduce TCO for PCs, 2005-2006 Update. Gartner Group. ID Number: G00136769

2 Sapuntzakis C, Brumley D, Chandra R, et al. Virtual appliances for deploying and maintaining software. In: Proceedings of the 17th USENIX Conference on System Administration, USENIX Association, Berkeley, 2003. 181–194

3 Krsul I, Ganguly A, Zhang J, et al. VMplants: providing and managing virtual machine execution environments for grid computing. In: Proceedings of the ACM/IEEE SC2004 Conference on High Performance Networking and Computing, Pittsburgh, 2004. 1–7

4 Zhang Y H, Wang X L, Hong L. Portable desktop applications based on P2P transportation and virtualization. In: Proceedings of the 22nd Large Installation System Administration Conference (LISA '08), San Diego, 2008. 133–144

5 Zhang Y H, Su G L, Zheng W M. Converting legacy desktop applications into on-demand personalized software. IEEE Trans Serv Comput, 2010, 3: 306–321

6 Zhang Y H, Su G L, Zheng W M. A user-space file system for on-demand legacy software. Sci China Inf Sci, 2010, 54: 1142–1150

7 Griffioen J, Appleton R. Reducing file system latency using a predictive approach. In: Proceedings of USENIX Summer 1994 Technical Conference, Boston, 1994

8 Kozuch M, Satyanarayanan M. Internet Suspend/Resume. In: Proceedings of the 4th IEEE Workshop on Mobile Computing Systems and Applications, New York, 2002. 40–48

9 Chen P M, Noble B D. When virtual is better than real. In: Proceedings of IEEE 8th Workshop on Hot Topics in Operating Systems, Elmau/Oberbayern, 2001. 133–138

10 Buyya R, Yeo C S, Venugopal S, et al. Cloud computing and emerging it platforms: vision, hype, and reality for delivering computing as the 5th utility. Fut Gener Comput Syst, 2009, 25: 599–616

11 Bradshaw R, Desai N, Freeman T, et al. A scalable approach to deploying and managing appliances. In: Proceedings of TeraGrid '07 Conference, Madison, 2007. 12–18

12 Kecskemeti G, Kacsuk P, Terstyanszky G, et al. Automatic service deployment using virtualisation. In: Proceedings of 16th Euromicro International Conference on Parallel, Distributed and Network-Based Processing. Tolouse: IEEE Computer Society, 2008. 628–635

13 Kecskemeti G, Terstyanszky G, Kacsuk P, et al. An approach for virtual appliance distribution for service deployment. Fut Gener Comput Syst, 2011, 27: 280–289

2) Http://www.microsoft.com/systemcenter /softgrid/default.mspx

14 Epstein A, Lorenz D H, Silvera E, et al. Virtual appliance content distribution for a global infrastructure cloud service. In: Proceedings of 2010 IEEE InfoComm, San Diego, 2010. 1–9
15 Alpern B, Auerbach J, Bala V, et al. PDS: a virtual execution environment for software deployment. In: Proceedings of the 1st ACM International Conference on Virtual Execution Environments, Chicago, 2005. 175–185

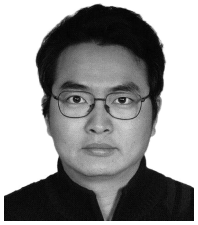
Cloudow: A SaaS runtime system based on the user-level virtualization technology

ZHANG YouHui*, LI YanHua & ZHENG WeiMin

Department of Computer Science and Technology, Tsinghua University, Beijing 100084*, China*
*E-mail: zyh02@tsinghua.edu.cn

Abstract SaaS is a promising mode for software delivery and usage across the Internet, which eliminates the need of users to purchase, maintain and upgrade software to a large extent. This paper proposes a new SaaS mode for legacy Windows software and its prototype, Cloudow. Cloudow enables ordinary users of Windows PC to use legacy desktop software from the Internet without installation, and the system supports to restore the user's personal software configurations on another PC. User-level virtualization technology is employed to enable legacy software to run without installation. Moreover, an on-demand delivery method based on the user-space file system is adopted, so the user can launching the remote software just as launching the local counterpart. Compared with the existing SaaS modes (like the VM-based solution and the Web applications), the legacy software can run on the client end directly while stored on the remote site, which gets good balance between compatibility and performance. Furthermore, aggressive pre-fetch mechanisms (including metadata/data/file pre-fetch) and caching technologies are employed to improve the IO access performance remarkably. The tests show that for much common software, the runtime overhead is about 12%–20%.

Keywords software architecture, service oriented architecture, software as a service, user-level virtualization, user-level file system



ZHANG YouHui was born in 1974. He received the Ph.D. degree in computer science from Tsinghua University. Currently he is an Associate Professor at the Department of Computer Science and Technology in Tsinghua University. His research interests include micro-architecture, portable computing and virtualization.

ZHENG WeiMin was born in 1946. He received the M.S. degree in computer science from Tsinghua University. Currently he is a Professor at the Department of Computer Science and Technology in Tsinghua University. His research interests include high performance computing, network storage and parallel compiler. He is the Chair of China Computer Federation.