



(12) 发明专利

(10) 授权公告号 CN 101512488 B

(45) 授权公告日 2013. 11. 27

(21) 申请号 200780033514. 5

(22) 申请日 2007. 08. 07

(30) 优先权数据

60/821, 673 2006. 08. 07 US

11/835, 307 2007. 08. 07 US

(85) PCT申请进入国家阶段日

2009. 03. 10

(86) PCT申请的申请数据

PCT/US2007/075407 2007. 08. 07

(87) PCT申请的公布数据

W02008/019380 EN 2008. 02. 14

(73) 专利权人 甲骨文国际公司

地址 美国加利福尼亚州

(72) 发明人 乔基姆·达尔斯泰特

(74) 专利代理机构 中国国际贸易促进委员会专

利商标事务所 11038

代理人 鲍进

(51) Int. Cl.

G06F 9/455 (2006. 01)

G06F 9/46 (2006. 01)

(56) 对比文件

WO 2006063274 A1, 2006. 06. 15,

US 6732139 B1, 2004. 05. 04,

FRASER 等. The Xenoserver

computing infrastructure. 《Technique Report》. 2003, 3-11.

审查员 张文

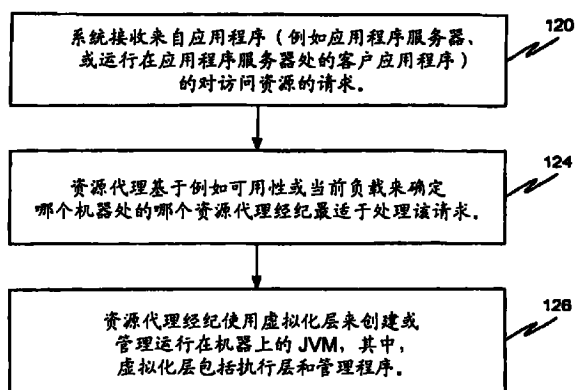
权利要求书2页 说明书16页 附图5页

(54) 发明名称

在虚拟机环境中提供硬件虚拟化的系统和方
法

(57) 摘要

用于在虚拟机环境中提供硬件虚拟化和资源管理的系统和方法。根据一个实施例,应用程序服务器环境包括计算机系统、应用程序服务器,和虚拟机(例如,Java 虚拟机或 JVM)。根据一个实施例,虚拟化层被设置在每一物理机处,包括用于在机器上划分多个虚拟机的管理程序。执行层运行单一的基于 Java 的应用程序,尽可能高效集中运行该应用程序。根据另一个实施例,该系统包括两个主要部件:第一,代替常用操作系统的低级执行环境;以及第二,高级资源代理,用于负责将各新的虚拟机分发给上述各层,并用于监控正在运行的虚拟机的当前资源的使用。



1. 一种用于在虚拟机环境中提供硬件虚拟化的系统,包括:
虚拟化层模块,其中,所述虚拟化层模块代替典型的操作系统在逻辑上位于为计算机硬件或机器的处理设备顶部,并且其中,所述虚拟化层模块包括:
管理程序层部件,其用于在所述处理设备上划分多个虚拟机,
多个执行层部件,
其中,所述执行层部件的每个适于运行单一的基于 Java 的应用程序;
多个 Java 虚拟机,所述多个 Java 虚拟机在逻辑上位于虚拟化层模块之上并运行在所述执行层部件上,其中,每个 Java 虚拟机与其自身的执行层部件相关联并被用于运行基于 Java 的应用程序;以及
其中,每个执行层部件被建立以优化在其上运行的其相应 Java 虚拟机的执行。
2. 根据权利要求 1 所述的系统,进一步包括多个所述虚拟化层模块,所述多个所述虚拟化层模块包括分别位于多个机器中的每个的虚拟化层模块。
3. 根据权利要求 1 所述的系统,其中,所述执行层部件中的每一个适于在特定时间只集中运行所述单一的基于 Java 的应用程序,而不能运行任何其他应用程序。
4. 根据权利要求 3 所述的系统,其中,所述执行层部件中的每一个适于尽可能高效的运行所述单一的基于 Java 的应用程序。
5. 根据权利要求 1 所述的系统,进一步包括多个所述执行层部件,其中所述执行层部件中的每一个均适于在一时间尽可能高效地只运行单一的基于 Java 的应用程序。
6. 根据权利要求 5 所述的系统,其中,多个所述执行层部件中的每一个均适于集中于所述单一的基于 Java 的应用程序,而不能运行任何其他应用程序。
7. 根据权利要求 5 所述的系统,其中,多个所述执行层部件位于管理程序层部件、虚拟化层模块、和机器的相同组合上。
8. 根据权利要求 1 所述的系统,进一步包括:
多个所述虚拟化层模块,位于多个机器中的每个处,其中,每个所述虚拟化层模块包括用于在所述机器上划分多个虚拟机的所述管理程序层部件;以及
资源代理服务模块,用于:
接收来自应用程序的对虚拟机提供的资源的访问请求,
确定多个所述虚拟机中的哪一个最适于处理所述请求,以及
经由其虚拟化层模块将所述请求传送至所选择的虚拟机。
9. 根据权利要求 8 所述的系统,其中,所述系统进一步包括多个资源代理经纪模块,所述资源代理服务模块经由其虚拟化层模块通过所述资源代理经纪模块将请求传送至最适合处理所述请求的虚拟机。
10. 根据权利要求 8 所述的系统,其中,所述虚拟化层模块和所述资源代理服务模块根据对作出所述请求的应用程序的预定保证来分配请求。
11. 根据权利要求 8 所述的系统,其中,所述管理程序层部件是 Xen 管理程序部件。
12. 根据权利要求 8 所述的系统,所述管理程序层部件是 VMware 管理程序部件。
13. 一种用于在虚拟机环境中提供硬件虚拟化的方法,包括以下步骤:
提供物理计算机硬件,所述计算机硬件为物理服务器、计算机机器、或网络设备;
提供虚拟化层模块,其中,所述虚拟化层模块代替典型的操作系统在逻辑上位于为计

计算机硬件或者机器的处理设备顶部,并且其中,所述虚拟化层模块包括:

管理程序层部件,用于在所述处理设备上划分多个虚拟机,以及
多个执行层部件,

其中,每个执行层部件适于运行单一的基于 Java 的应用程序;以及

提供多个 Java 虚拟机,所述多个 Java 虚拟机在逻辑上位于虚拟化层模块之上并运行在所述执行层部件上,用于运行多个基于 Java 的应用程序,其中,每个 Java 虚拟机与其自身的执行层部件相关联并被用于运行基于 Java 的应用程序,以及,其中,每个执行层部件被建立以优化在其上运行的其相应 Java 虚拟机的执行。

14. 根据权利要求 13 所述的方法,进一步包括以下步骤:

利用资源代理服务模块接收来自应用程序的对虚拟机提供的资源的访问请求;

利用所述资源代理服务模块确定多个虚拟机或多个物理机或计算机中的哪一个最适于处理所述请求;以及

使用虚拟化层模块传送所述请求以创建或管理虚拟机或所选的物理机上运行的虚拟机;以及

在所选择的虚拟机处处理所述请求。

15. 根据权利要求 14 所述的方法,其中,所述方法进一步包括提供包括经纪模块的多个资源代理经纪模块,所述资源代理服务模块经由其虚拟化层模块通过所述资源代理经纪模块将请求传送至最适合处理所述请求的虚拟机。

16. 根据权利要求 14 所述的方法,其中,所述虚拟化层模块和所述资源代理服务模块根据对作出所述请求的应用程序的预定保证来分配请求。

在虚拟机环境中提供硬件虚拟化的系统和方法

[0001] 版权声明

[0002] 本专利文件公开的一部分包括受版权保护的内容。当专利文件或专利公开中的任意一个的复制出现在专利和商标局专利文件或记录中时,该版权所有者对此没有异议,而对于其他任何情况都保留全部版权。

[0003] 优先权声明

[0004] 本发明申请要求于 2006 年 8 月 7 日提交的美国申请号为 60/821,673 的临时专利申请“SYSTEM AND METHOD FOR PROVIDING HARDWARE VIRTUALIZATION IN A VIRTUAL MACHINE ENVIRONMENT”、以及于 2007 年 8 月 7 日提交的美国申请号为 11/835,307 号专利申请“SYSTEM AND METHOD FOR PROVIDING HARDWARE VIRTUALIZATION IN A VIRTUAL MACHINE ENVIRONMENT”的权益,通过引证将其结合于此。

技术领域

[0005] 本发明大体涉及操作系统、应用程序服务器、和虚拟机,更具体地涉及用于在虚拟机环境中提供硬件虚拟化和资源管理的系统和方法。

背景技术

[0006] 当今的许多任务关键商业应用程序已经被开发出来并被部署在基于 Java 的应用程序服务器(包括,例如,符合 Java 2 企业版或 J2EE 规格的应用程序服务器)上。这些应用程序通常被部署在单售主 RISC 计算机和服务器的上,以及其他昂贵的硬件平台上。由于缺乏竞争力,不认为易于得到且低成本的基于 PC 的服务器适于替代能以这些构造使用的高性能 Java 虚拟机(JVM)。最新的虚拟机环境,诸如 BEA 系统公司的 JRockit 产品,能够解决这个问题就在于可以优化这些虚拟机以用于使 Java 应用程序能够在低成本、基于标准的平台上以提高的可靠性和性能来运行的各种环境(包括基于 PC 的环境)。JRockit 虚拟机是为功率需求服务器侧 Java 应用程序、提供较好性能、可管理性、和可靠性特别优良设计的虚拟机的一个实例。

[0007] 然而,尽管当前虚拟机产品(product offering)善于提供软件虚拟化,但是,至今很少能够以低层硬件或操作系统水平支持虚拟化,或在虚拟机水平对资源进行管理,然而,仍然允许应用程序服务器控制或参与虚拟化进程。

发明内容

[0008] 本文中公开了用于在虚拟机环境中提供硬件虚拟化和资源管理的系统和方法。根据实施例,包括计算机系统、应用程序服务器、及虚拟机(例如,Java 虚拟机或 JVM)的应用程序服务器环境被扩展为提供从应用程序服务器环境中的较高层到系统实际处理能力(power)的接口,从而可以以机器独立的方式来进行处理的分配。软件层被放置在虚拟机和提高虚拟化 Java 执行系统的效率的硬件之间。根据该实施例,该系统包括两个主要部件:第一,低级执行环境,其代替常用操作系统;以及第二,高级资源代理,用于负责将各新的虚

拟机分发给上面的层,并用于监控正在运行的虚拟机的当前资源的使用。以此方式,低层物理机可以被划分来支持多个 JVM。此外,关于 CPU 和其他资源分配,可以在系统内部设置目标,并且该系统可以适当地分配多个 JVM。

附图说明

- [0009] 图 1 示出了根据本发明一个实施例的包括虚拟机的应用程序服务器环境的示例。
- [0010] 图 2 示出了根据本发明一个实施例的分层虚拟机构造的示例。
- [0011] 图 3 示出了根据本发明一个实施例的多层域环境的示例。
- [0012] 图 4 示出了根据本发明一个实施例的可以如何将虚拟层用于共用计算环境中的示例。
- [0013] 图 5 示出了根据本发明一个实施例的连同资源代理的裸机 (bare metal) 域的示例。
- [0014] 图 6 示出了根据本发明一个实施例的连同 JVM 代理服务器的裸机域的示例。
- [0015] 图 7 示出了根据本发明一个实施例的虚拟机环境部件的逻辑图。
- [0016] 图 8 示出了根据本发明一个实施例的用于使用虚拟化的方法的流程图。

具体实施方式

[0017] 本文中公开了用于在虚拟机环境中提供硬件虚拟化和资源管理的系统和方法。根据实施例,包括计算机系统、应用程序服务器、及虚拟机(例如,Java 虚拟机或 JVM)的应用程序服务器环境被扩展为提供从应用程序服务器环境中的较高层到系统实际处理能力的接口,从而可以以机器独立的方式来进行进程的分配。软件层被放置在虚拟机和提高虚拟化 Java 执行系统的效率的硬件之间。根据该实施例,该系统包括两个主要部件:第一,低级执行环境,其代替常用操作系统;以及第二,高级资源代理,用于负责将各新的虚拟机分发给上面的层,并用于监控正在运行的虚拟机的当前资源的使用。以此方式,低层物理机可以被划分来支持多个 JVM。此外,关于 CPU 和其他资源分配,可以在系统内部设置目标,并且该系统可以适当地分配多个 JVM。

[0018] 术语

[0019] 本文中使用了下列术语。

[0020] 域:运行在管理程序层顶部的操作系统实例。多个域可以彼此不知道地运行在同一机器上。域在本文中还可以称作客户操作系统(OS)、客户(guest)、或 VM。

[0021] 裸机域:根据本发明一个实施例的运行虚拟层、虚拟机和应用程序的域。(例如,在一个实施例中,裸机域包括裸机操作系统(OS)层加诸如 JRockit JVM 的 JVM,加应用程序)。裸机域在本文中还可以称作 BM 域。

[0022] 节点:网络中的物理机或物理计算机。

[0023] 网格计算:一组一起工作一解决一个计算问题的多个节点。

[0024] 共用计算:允许多个应用程序在多个物理机上有效运行的系统或环境。共用计算环境允许对各应用程序指定高级策略以帮助系统决定将哪个(些)特定应用程序列入优先地位。共用计算环境还允许用户为系统部署新的应用程序;为每个应用程序提供高级策略和/或保证;监控应用程序和机器的状态;处理指定应用程序的资源使用的快速增长;供应

系统升级 ; 以及提供高可用性。

[0025] 图 1 示出了包括虚拟机的且可以以本发明一个实施例的方式来使用应用程序服务器环境的示例。如图 1 所示, 应用程序服务器环境 10 包括应用程序服务器 14 (例如, 由 BEA 系统公司开发的 WebLogic 服务器产品, 或其他类型的应用程序服务器), 以及虚拟机 16 (例如, 还是由 BEA 系统公司开发的 JRockit JVM, 或其他类型的虚拟机)。图 1 还示出了常用在应用程序服务器环境中的可选部件, 例如, 开发环境 18、用户集成部件 20、或线程集成部件 22。后面这些可选部件分别包括 WebLogic Workshop、WebLogic Portal、和 WebLogic 集成部件, 其中的每一个都是由 BEA 系统公司开发的。需要的话, 还可以将其他部件添加至该环境。

[0026] 图 2 示出了根据本发明一个实施例的虚拟机构造的更详细的示例。如图 2 所示, 诸如 JRockit JVM 的典型的虚拟机 16 包括用于管理系统中运行的线程的线程管理部件 ; 用于管理诸如垃圾收集的过程的存储管理部件 18 ; 代码生成部件 22 ; 以及 Java 模型部件 24。典型的虚拟机还提供外部接口 28 用于对虚拟机进行管理。尽管图 2 示出了 JRockit 产品, 但是很显然, 在本发明的精神和范围内, 还可以使用除了 JRockit 实现以外的其他虚拟机。

[0027] 为了能够有效的进行应用程序虚拟化, 就需要相应的有效硬件资源虚拟化。这使得提供诸如挂起、动态配置 (live provisioning)、和动态迁移的功能是可能的。根据本发明的一个实施例, 添加了抽象的额外层。这使得虚拟机被移动至相对更接近下层硬件。

[0028] 图 3 示出了根据本发明一个实施例的多层域环境的示例。如图 3 所示, 计算环境 30 包括物理计算机硬件或计算机 32。该计算机可以是物理服务器、计算机、网络设备、或等同类型的计算或处理设备。计算环境包括代替典型的操作系统而在逻辑上位于处理设备顶部的虚拟化层 34。根据一个实施例, 虚拟化层包括执行层部件 38 (本文中也称作“裸机层”), 和管理程序层部件 40 或管理程序层。当前可得到的管理程序层部件的实例包括 Xen 管理程序和 VMWare 管理程序。根据其他的实施例, 还可以使用其他的管理程序层和管理程序层部件。

[0029] 管理程序层插入在服务器的硬件和操作系统之间。这提供了允许每个物理服务器运行一个或多个虚拟服务器的抽象层, 并有效地将操作系统及其应用程序与下层物理服务器分离。一旦已经创建了虚拟服务器镜像, 其就可以运行在任意服务器上。基于同样的原理, 多个虚拟服务器可以同时共享单个物理服务器或机器。实际结果是增加了整个服务器的利用, 而同时满足了接收确保任何特定应用程序层性能标准的资源保证。

[0030] 如进一步在图 3 中示出的, 虚拟机 16 在逻辑上位于虚拟化层之上。根据一个实施例, 然后在虚拟机之上的层处提供资源代理 36、或一组资源代理服务模块。资源代理由应用程序服务器 14 和其他应用程序使用, 以使用由虚拟化层提供的服务。

[0031] 在一些实施例中, 通过为局域网中 Java 应用程序提供硬件虚拟化服务可以将上述系统用作整个共用计算环境的一部分。当用在共用计算环境的上下文中时, 可以提供允许网络应用程序与虚拟机连接的服务。在共用计算或多虚拟机环境中, 可以提供的服务的实例包括 : 在网络环境中某处创建新的虚拟机或 JVM 的能力 ; 资源保证, 诸如最小和最大 CPU、内存、或网络带宽使用 ; 资源计量和系统温度调节装置功能 ; 当应用程序需要更多或更少这些资源时, 动态增加或减少资源的能力 ; 诸如挂起至磁盘或从磁盘重新开始运行的应用程序 (本文中称作“冻干”) 的特征 ; 克隆“冻干”应用程序的能力 (类似于 Unix

fork-like 功能,即,以新的例程特定值来例示克隆的应用程序;以及将运行的例程移动至另一计算机以进行计划的系统维护的能力。

[0032] 图 4 示出了在共用计算环境 50 中可以如何使用虚拟化层的实施例的示例。如图 4 所示,应用程序可以使用应用程序虚拟化特征 52(包括,例如,上述的资源代理服务模块或共用计算服务)来经由应用程序服务器云 56 来访问下层硬件虚拟化层 54。根据一个实施例,应用程序服务器可以是 WebLogic 服务器环境,其还可以提供其他高级的服务器特征(诸如企业群集)。

[0033] 图 5 示出了根据本发明一个实施例的连同资源代理的裸机域的示例。根据一个实施例,资源代理向应用程序服务器和在其上运行的其他软件应用程序提供子服务。这些子服务可以包括例如:获得新的 JVM;提供运行在服务器上的 JVM;计量特定 JVM 的资源;以及控制 JVM 的资源。根据一个实施例,用于 JVM 创建的主部件是资源代理 68(如上所述),在该示例中,资源代理跟踪机器 60、62 中的哪一个可用于在其上创建新的虚拟化层例程。根据一个实施例,资源代理经纪(resource broker agent,简称 RBA)64、66 位于每个机器上。根据其他实施例,不需要物理机和经纪之间的任何一对一的映射,例如,单一的经纪可以覆盖多个物理机。资源代理经纪负责 JVM 例程 70、72、74 的局部创建,即,作为经纪来创建和管理同一物理机上的 JVM 的操作。从第一机器到第二机器或整个群集的所有外部通信首先经过资源代理,然后向相应的资源代理经纪发送请求。在此方式中,资源代理就像一个经纪池,并跟踪哪个特定经纪控制哪个特定机器,而该特定机器还具有完成来自应用程序请求的必需的可用功率。

[0034] 根据一个实施例,资源代理是通用硬件抽象层。其可以以大量不同机器类型来使用。如果机器类型支持划分,则资源代理可以在需要时使用该功能来划分该机器。根据特定机器类型支持哪种特征,可以得到不同的高端功能。至少该功能应该允许以接近的资源使用目标(例如至少受限保证,受限共享和非迁移)在机器上创建新的 JVM。

[0035] 应该注意,尽管当将资源代理与执行层组合会使系统运行的更好,并且还将支持更大的特征范围,但是对于资源代理本身来讲,其自身不需要执行层(即,裸机层)来执行其多种操作。

[0036] 应该注意,标准机,即,非虚拟机还可以被添加至资源代理,即使在这些情况下功能会减少。根据其他实施例,资源代理还可以包括对诸如 Azul、Solaris Containers 和其他管理程序堆栈的第三方系统的支撑。根据一个实施例,Xen 域和 VMWare 客户也能够支持原裸机资源保证、资源共享、挂起、迁移、可热插拔、和虚拟文件系统。

[0037] JVM 代理

[0038] 图 6 示出了根据本发明一个实施例的连同可选 JVM 代理的裸机域的示例。如上所述,在一些实施例中,资源代理服务模块可以用作独立实体。然而,在这些实例中,不会获得与通过所有元件的组合提供的相同的资源共享和相同的资源保证。如图 6 所示,物理服务器 80 可以在其上运行多个 JVM。在一些实例中,由标准操作系统(OS)90 来操纵 JNI 代码执行和实体设备驱动器访问。当运行在管理程序层 96 顶部时,建立执行层 92、94 以优化在其上运行的任一 JVM 86、88 的执行。具体地,管理程序层/执行层组合支持数据的隔离和写时复制共享以减小运行在同一物理机上的多个 JVM 的足印。根据一个实施例,OS 层上的 JVM 代理 82、82 被提供来允许经由 OS 调用执行层中的功能和服务。根据其他实施例,可以

使用裸机执行层而无需 JVM 代理。

[0039] 性能监控框架 -CPU 能力和 CPU 共享

[0040] 如上所述,该系统的有用特征之一是响应于应用程序请求来分配可用资源的可靠性。根据一个实施例,CPU 能力是应用程序消耗了或保证具有多少 CPU 时间的近似。该信息然后被用于在可用机器或其他物理硬件上分配 JVM。根据一个实施例,所有测量结果的单位都被认为是估计值。这些测量结果然后被用作相对测量结果(即,200 CPU 能力单位约为 100 CPU 能力单位计算能力的两倍)。对于一个物理机可以产生的单位的数量来说,通常是当前使用的所有虚拟机的确切总数加上空闲单位。在一个特定机器上,相对测量结果可能会比如果考虑通过若干不同机器工作地更好并且更有意义。具体地,两个机器越是不同就越不可能会有精确的相对测量结果。目标通常在于 CPU 能力应该可比得上不同机器间的最远扩展可能。

[0041] 系统接口和功能

[0042] 表 1 列出了根据一个实施例的由系统提供的一些接口和功能。这些功能允许高层应用程序创建 JVM 以及在若干个机器(包括在集群内)上分配资源。很显然,在其他实施例中,还可以提供其他的接口和功能。

[0043]

创建接口	
请求:	响应:

[0044]

给我一个允许该资源规格的新的 JVM; 并将该例程分配给一个机器, 但还不启动。	JVM-句柄
给我一个新的 JVM; 分配该例程给机器; 使用指定的镜像来初始化该例程; 但仍不启动该例程。	JVM-句柄/ 成功 (OK) /失败 (Fail)。
去除该 JVM。	
初始化界面	
请求:	响应:
在你的窗口中安装标以 X 标签的目录; 并以 (只读、可读写、排他) 模式进行安装。	成功/失败。
设置用于启动的命令行。	成功/失败。
启动 JVM。	成功/失败。
挂起/重新开始界面	
请求:	响应:
挂起该运行着的 JVM。	成功/失败。
将挂起的 JVM 镜像存储至指定位置。然后在随后的调用中使用该镜像来创建 JVM。	成功/失败。
重新开始; 重新开始之前挂起的 JVM; 任何有效的网络连接都将断开。	成功/失败。
资源界面	
请求:	响应:
报告你正在使用多少资源	使用中的资源。
请求可用资源的改变 (增加或减少)。	成功/失败。

[0045] 根据一个实施例, 系统可以请求和测量四种类型资源中的任一个或全部, 然后其可以使用这些资源来将这些资源分配到合适的 JVM 上。这些可以包括: CPU 能力测量 (如上所述); Java 堆内存; 网络带宽; 以及磁盘 I/O。其他的实施例可以提供其他类型的资源测量选项。

[0046] 当请求 CPU 能力时, 上面的层都可以设置最小量 (即, 必须保证可用的量) 和权

重。对于其他三种资源类型只能被指定最小量。基于权重所指导的先来先服务,可能提供过量。该设计选择根本上简化了优化空间,并使得服务器的布置更简单。该方法的根本原因是因为大多数的应用程序倾向于受处理限制或受内存限制(即,受内存限制如同在等待加载,不是如同需要更多的内存)。

[0047] 根据一个实施例,内存堆被设计为在应用程序的需要进行请求时可展开,因此,通常不应指定最大堆尺寸。只有在系统在地址空间(其可以是完全不固定的)不足或物理内存不足时才发生内存不足错误(这可以通过移动至具有更多内存的逻辑单元、将同一机器上的 VM 移动至另一机器、或通过减少由其他 VM 使用的内存来解决)。

[0048] 使用管理程序的好处

[0049] 使用管理程序的好处典型地包括,管理程序在隔离方面做得很好,而操作系统(OS)进程在这方面做得不好。尤其是在进程水平难以保证对 CPU 最小量的要求。诸如节省系统的全部状态的挂起和重新开始的特征在 OS 层上的任意层也是很难的。问题在于其是具有系统状态的全部信息的仅有的层(例如, JVM 可以能够跟踪打开的文件、映射的内存、原生代码中的线程等,但是在重新开始时 JVM 完全恢复状态是不可能的。动态迁移特征更简单:由于管理程序对硬件进行抽象,因此系统具有将运行着的系统迁移至另一逻辑单元的能力。通过 JVM 层之间的合作,穿过 OS 层到达管理程序层,这可以更有效地执行。使用管理程序还可以保证例如,可以使系统可用一定量的 CPU 时间,并且系统可以计量每个容器(container)正在使用多少 CPU 时间。

[0050] 使用特定 OS 层的好处

[0051] 在一些实施例中,使用了特殊的操作系统(OS)层。可以在许多个级(应用程序、应用程序服务器、JVM、OS)处进行内存共享。但是许多类型的共享在堆栈的某些层处执行得更有效。具体的,来自 JVM 和管理程序的组合支持的组合允许系统在不同 JVM 之间以反向叉(inverse-fork-like)方式共享内存。这意味着系统可以“在事实过后(after the fact)”告诉管理程序在不同的 JVM 之间一些页是相同的。管理程序于是可以以写时复制策略让这些页映射到相同的物理页。

[0052] 此外,特殊的 OS 层使得能够执行之前不可能的一些功能,例如:OS 将只处理一个进程,即运行应用程序的 JVM;以及 OS 理解 JVM 并且在 JVM 和使得能够进行有效优化的 OS 之间进行精确的合作/通信。一些特征在没有 OS 级支持的情况下是不可能的,例如:

[0053] ●在垃圾收集器在那些区中压缩时,通过使增变线程的一些页不能运行的无休止压缩;

[0054] ●代替在时钟日程上旋转的精确的线程日程,该线程保持时钟。

[0055] 特殊的 OS 层还通过 OS 层去除来产生一些有效性。去除 OS 层导致较少的暂停次数,在高连接过程期间较好的运行和低内存。

[0056] 添加管理程序添加了额外的抽象级,还自然地使系统慢下来。通过去除操作系统,系统可以补偿该降慢并且可能会通过其来进行提高。通过使用管理程序以及具有作为客户 OS 之一安装的一般的操作系统,裸机可以避免为不同的设备写设备驱动器,并且其只是通过管理程序告诉具有操作系统的客户 OS 上的设备驱动程序的简单的裸机。

[0057] 系统设计

[0058] 根据一个实施例,系统包括资源管理服务,该服务包括作为其主要接口的 Java 接

口。该接口的用户可以包括应用程序服务器、应用程序、和企业范围 (enterprise-wide) 应用程序 (例如, 群集应用程序)。一个实施例的实际 Java API 如附件 A 所示。

[0059] 图 7 示出了根据本发明一个实施例的虚拟机环境部件的逻辑示意图 100。虚拟化层理想地是尽可能的小。如图 7 所示, 根据一个实施例, 这减小至管理程序 (例如, Xen 或 VMWare, 或其他管理程序) 的薄层、操作环境 (例如, 裸机执行层) 的薄层、以及执行引擎 (例如, JRockit 或另一 JVM) 的 fatter blob, 所有的这些都运行非常大的应用程序 (例如, WebLogic 服务器或另一应用程序服务器、加上客户应用程序)。

[0060] 在一些实施例中, 假设了准虚拟化程序 (即, 在 OS 和管理程序之间存在合作, OS 是知道管理程序的), 则假设建立了 OS 层; 然而, 其他实施例可以使用合作的 OS 和管理程序。

[0061] 图 8 示出了根据本发明一个实施例的使用虚拟化的方法的流程图。如图 8 所示, 在步骤 120 中, 系统接收来自应用程序 (例如, 应用程序服务器, 或在应用程序服务器上运行的客户应用程序) 访问资源的请求, 该请求必须由 JVM 提供。在步骤 122 中, 资源代理基于例如可用性或当前负载来确定哪个机器出的哪个资源代理经纪最适于处理该请求。在步骤 124 中, 资源代理经纪使用虚拟化层创建或管理在该机器上运行的 JVM, 其中, 虚拟化层包括执行层和管理程序。

[0062] 附件 A

[0063] 下列代码限定了以本发明一个实施例使用的实际 Java API。显而易见地是, 在本发明精神和范围内的其他实施例中可以限定其他 API。

[0064] Resource Broker API

[0065] package com.bea.jvmbroker;

[0066] public interface JVMBroker

[0067] {

[0068] /**

[0069] * 创建一个新的 JVM 例程。JVM 例程受限于指定机器并且具有

[0070] UUID 并且在下文中可以共用。

[0071] *JVM 还没有使用任何资源

[0072] */

[0073] public JVM createJVM(JVMRequest rc);

[0074] /**

[0075] * 创建一个新的 JVM 例程。

[0076] *@ 注意这是一个看见 createJVM 的更多信息的方便的方法。

[0077] */

[0078] public JVM createJVM();

[0079] /**

[0080] * 从镜像创建一个新的 JVM 例程。该镜像之前挂起的 JVM。

[0081] * 当被启动时, JVM 将以挂起的状态被启动并且对 resume() 的调

[0082] * 用将被需要以重新开始执行

[0083] *@ 注意这是一个看见 createJVM 的更多信息的方便的方法。

```
[0084]      */
[0085]      public JVM createJVMFromImage(JVMRequest rc,File image) ;
[0086]  }
[0087]  /**
[0088]   * 该接口表示来自 JVM 资源代理的 JVM 句柄
[0089]   * 透视的
[0090]   * 该接口用于配置 JVM。在这点上,你具有其中之一,JVM 限于一
[0091]   * 台机器,但是它仍未启动。
[0092]   */
[0093]  public interface JVM
[0094]  {
[0095]      /**
[0096]       * 获取 JVM 的 IP 地址。这典型的是一个 IP 地址。
[0097]       *
[0098]       */
[0099]       public InetAddress[]getAddresses() ;
[0100]      /**
[0101]       * 设置 JVM 的启动命令
[0102]       *
[0103]       * @param name- 启动的主类名称
[0104]       * @param args- 传送至主类的主方法的论据
[0105]       */
[0106]       public void setStartCommand(String name,String[]args)throws
[0107]  IllegalStateException ;
[0108]      /**
[0109]       * 安装预订安装点调用路径。路径必须是有效的
[0110]       * 所述机器上的预订安装点。路径为类似 UNIX 的路径
[0111]       *
[0112]       * 即,安装 ( " /wls" ) 或安装 ( " /wls90" ) ;
[0113]       */
[0114]       public void mount(String path) ;
[0115]      /**
[0116]       * 启动 JVM。该方法只能调用一次。
[0117]       */
[0118]       public void start()throws IllegalStateException ;
[0119]      /**
[0120]       * 挂起运行的 JVM。
[0121]       */
[0122]       public void suspend() ;
```

```
[0123]    /**
[0124]    * 重新开始执行挂起的 JVM。
[0125]    */
[0126]    public void resume() ;
[0127]    /**
[0128]    * 被调用来增加或减少该 JVM 的经同意的资源使用
[0129]    */
[0130]    public boolean changeRequirements(JVMRequirements rc) ;
[0131]    /**
[0132]    * 获取对当前资源使用的估计
[0133]    *
[0134]    * 该信息可以用于在更高的级进行群集决定。
[0135]    *
[0136]    * 例如,如果你正使用与你请求相同的或更多的能力单位,则该
[0137]    * 应用程序可能被能力单位限制 (类似于 CPU/ 内存限制),从
[0138]    * 而可能的话在本地请求更多的 PU
[0139]    * 可选地,如果应用程序没有使用其全部能力单位,
[0140]    * 则瓶颈可以位于别处,并且请求更多的能力或
[0141]    * 启动更多的例程不会有帮助。
[0142]    */
[0143]    public JVMResources getResourceUsage() ;
[0144]    /**
[0145]    * 获取该 JVM 所在的物理机。此处的物理表示资源代理认为什
[0146]    * 么是物理机。
[0147]    * 一些其他层当然会欺骗我们相信虚拟机是物理机
[0148]    */
[0149]    public Machine getMachine() ;
[0150]    /**
[0151]    * 将节省对指定文件的挂起镜像 (该文件被指定为来自存储镜
[0152]    * 像调用者的本地路径)
[0153]    * 如果 JVM 位于另一机器上而不是调用者,则该操作完成该操
[0154]    * 作需要很多秒,这是因为必须将数据从该服务器移动至调用
[0155]    * 者 (或调用者所指的任何文件服务器)
[0156]    * 只有在 JVM 挂起时才可以调用该方法
[0157]    */
[0158]    public void storeImage(File f) throws IllegalStateException ;
[0159]    /**
[0160]    * 注册对 JVM 事件感兴趣的收听者
[0161]    */
```

```
[0162]     public void registerListener(JVMListener j l) ;
[0163] }
[0164] /**
[0165]  * 这是用于指定对 JVM 的局域网宽请求的接口, 其在其可以使用多
[0166]  * 少资源以及更多的全局请求 ( 诸如其可以关于其他 JVM 来定位 )
[0167]  * 之前都可以启动
[0168]  *
[0169]  * 对于 JVM 本地请求还可以见 JVMRequirements。
[0170]  *
[0171]  * @ 见 com.bea.jvmbroker.JVMRequirements
[0172]  * @ 作者 Joakim Dahlstedt(joakim.dahlstedt@bea.com)
[0173]  */
[0174] public interface JVMRequest{
[0175]     /**
[0176]      * 获取 JVM 资源请求
[0177]      */
[0178]     public abstract JVMRequirements getJVMRequirements() ;
[0179]     /**
[0180]      * 为该 JVM 设置 JVM 资源请求
[0181]      */
[0182]     public abstract void setJVMRequirements(JVMRequirements r) ;
[0183]     /**
[0184]      * 获取该 JVM 所属的 JVM 组。该组通常用于通知资源代理该组
[0185]      * JVM 为同一应用程序服务
[0186]      */
[0187]     public abstract JVMGroup getJVMGroup() ;
[0188]     /**
[0189]      * 获取该 JVM 所属的 JVM 组。该组通常用于通知资源代理该组
[0190]      * JVM 为同一应用程序服务
[0191]      */
[0192]     public abstract void getJVMGroup(JVMGroup jg) ;
[0193]     /**
[0194]      * 获取该 JVM 必须在其上运行的特定机器
[0195]      */
[0196]     public abstract Machine getMachine() ;
[0197]     /**
[0198]      * 请求该 JVM 在特定机器上运行。
[0199]      *
[0200]      * 这是不被推荐的行为。它将严格限制资源代理能力以规划资
```

```
[0201]      * 源来充分利用机器。
[0202]      *
[0203]      * 可以使用 setMachine 的情况 : 当特定机器只是可以访问某种服
[0204]      * 务的机器时 ( 就是说它只是具有到 CICS 系统或到 REUTERS
[0205]      * 终端的连接 )。
[0206]      *
[0207]      *@ 注意这不是程序设计语言中支持的功能
[0208]      *@ 注意该构思会在 V1.0 中支持它
[0209]      */
[0210]      public abstract void setMachine(Machine m) ;
[0211]      /**
[0212]      * 获取自动启动值
[0213]      */
[0214]      public abstract int getAutoRestart() ;
[0215]      /**
[0216]      * 指定在利用非零退出代码关闭 JVM 的情况下是否重新启动
[0217]      *
[0218]      * 该数字指定在放弃前尝试重启动多少次。
[0219]      *
[0220]      * 0 = 不自动重启动 ( 默认 )
[0221]      */
[0222]      public abstract void setAutoRestart(int ntimes) ;
[0223]      }
[0224]      /**
[0225]      * 该级别表示 JVM 上的本地资源限制。该情况中的本地指的是 JVM
[0226]      * 本身的本地, 如果你想对不同 JVM 之间的依赖性提出请求或想要
[0227]      * 某一 JVM 在某一机器上执行或其他类似的事情, 你应该查看
[0228]      * JVMRequest
[0229]      *@ 见 com. bea. jvmbroker. JVMRequest
[0230]      */
[0231]      public class JVMRequirements{
[0232]          private static final int DEFAULT_WEIGHT = 100 ;
[0233]          private static final long MIN_MEMORY = 16 ; //jrockit supports no
less
[0234]          private static final long MIN_PU = 1 ;
[0235]          private static final long MIN_NBW = 50 ; //50kBit ~ modem speed
[0236]          private static final long MIN_DS = 500 ; //0.5GB
[0237]          /**
[0238]          * JVMRequirements 的优选构造器。只指定你想要多少能力。能力
```

```
[0239]      * 不仅仅是 CPU-MHz,它实际上是用于应用程序的 bogo-mips。
[0240]      * 这表示 JVM 管理（诸如垃圾收集和编辑）不包括在测量结果中,
[0241]      * 而是用于获得更多的能力,该系统例如可以给你更多的内存而
[0242]      * 不是 CPU。
[0243]      */
[0244]      public JVMRequirements(long minPU) ;
[0245]      /**
[0246]      *JVMRequirements 的简单构造器。为所有请求设置默认值。
[0247]      */
[0248]      public JVMRequirements() ;
[0249]      public JVMRequirements(long minPU,long minMem,long minNB) ;
[0250]      public JVMRequirements(long minPU, long minMem, long minNB, long
minDS) ;
[0251]  }
[0252]  public class JVMResources implements Serializable{
[0253]      /**
[0254]      * 返回在用的 CPU 能力的数量（这是通过最后三秒该 JVM 正执
[0255]      * 行多少" java 操作" 计算出来的平均值）。该思想是该机器应该
[0256]      * 有两倍的执行两倍于 CPU 能力的的能力,如果是这种情况则是
[0257]      * 不清楚的。
[0258]      */
[0259]      public long getCPU Power() {
[0260]          return CPU Power ;
[0261]      }
[0262]      /**
[0263]      * 这是 JVM 使用的所有内存（包括本地内存和堆内存）
[0264]      */
[0265]      public long getMemory() {
[0266]          return memory ;
[0267]      }
[0268]      /**
[0269]      * 获取由该 JVM 当前使用的本地磁盘空间量
[0270]      */
[0271]      public long getDiskUsage() {
[0272]          return diskUsage ;
[0273]      }
[0274]      /**
[0275]      * 获取网络带宽（这是通过最后三秒计算出来的平均值）
[0276]      */
```

```
[0277]     public long getNetworkBandwidth() {
[0278]         return networkBandwidth;
[0279]     }
[0280] }
[0281] public interface JVMListener
[0282] {
[0283]     public void aboutToSuspend();
[0284]     public void resumed();
[0285]     /**
[0286]      * 在 JVM 已经退出后（关闭）后调用。传递的值是 JVM 的退出
[0287]      * 代码（如由进程返回的 ...，即，System.exit(value)
[0288]      */
[0289]     public void exited(int exitCode);
[0290] }
[0291] public interface ResourceBroker
[0292] {
[0293]     /**
[0294]      * 将返回与 ResourceBroker 相关的机器队列。
[0295]      */
[0296]     Machine[] getMachines();
[0297]     /**
[0298]      * 将返回 ResourceBroker 认为当前在其“域”中运行的 JVM 队列。
[0299]      */
[0300]     JVM[] getJVMs();
[0301]     /**
[0302]      * 由外部源作出的从池中去除机器（可能放入另一池中）的请
[0303]      * 求。
[0304]      */
[0305]     public Machine relinquishMachine();
[0306]     public void addMachine(Machine m);
[0307] }
[0308] /**
[0309]  * 该接口表示 JVM 资源代理是实际物理机的接口。如果在该机器上
[0310]  * 安装有管理程序，则可以根据资源代理的命令将该机器划分为多
[0311]  * 个虚拟机。
[0312]  */
[0313] public interface Machine
[0314] {
[0315]     /**
```

```
[0316]      * 获取该 LAN 中的机器的 IP 地址。所有的机器都应该具有该
[0317]      * LAN 中的一个精确的 IP 地址。（于是它们可以在逻辑单元上运
[0318]      * 行多个域,其中的每一个都具有自己的本地 IP）。
[0319]      */
[0320]      public InetAddress getAddress() ;
[0321]      /**
[0322]      * 获取当前有多少机器资源在用的快照。该机器的资源是大 4：
[0323]      * CPU / 内存 / 磁盘 / 网络
[0324]      */
[0325]      public ResourceUsage getResourceUsage() ;
[0326]      /**
[0327]      * 以 MB 为单位返回机器上的内存总量。
[0328]      */
[0329]      public long getTotalMemory() ;
[0330]      /**
[0331]      * 以 MB 为单位返回机器上的磁盘空间总量。
[0332]      */
[0333]      public long getTotalDiskSpace() ;
[0334]      /**
[0335]      * 检查当前机器是否支持管理程序（因此可以利用资源保证将
[0336]      * 该逻辑单元划分成多个逻辑单元,挂起 / 重新开始 / 迁移能力）
[0337]      */
[0338]      public boolean hasHypervisor() ;
[0339]  }
```

[0340] 本发明可以使用根据本公开的教导的传统的通用或特殊数字计算机或微处理器。对软件领域中的技术人员显而易见的是,可以基于本公开的教导通过本领域的程序员来容易地准备适当的软件编码。

[0341] 在一些实施例中,本发明包括计算机程序产品,该产品是包括其上存储的指令的存储介质（媒介）,其中,指令可以用于对计算机进行编程来执行本发明的任意进程。存储介质可以包括但不限于任何类型的磁盘,磁盘包括软盘、光盘、DVD、CD-ROM、微小驱动器、和磁光盘、ROM、RAM、EPROM、EEPROM、DRAM、VRAM、闪存设备、磁或光卡、纳系统（包括分子存储 IC）,或适于存储指令和 / 或数据的任何类型的介质或设备。

[0342] 为了示例和说明的目的,已经提供了本发明的前述描述。并不是用于穷尽或将本发明限制于披露的精确形式。许多改进和变化对于本领域的技术人员来说是显而易见的。特别是,当在 WebLogic 应用程序服务器和 JRockit 产品的上下文中描述上述系统和方法的实施例时,很显然,可以以其他类型的应用程序服务器、JVM 和虚拟机产品来使用这些系统和方法。类似的,当在 Xen 管理程序的上下文中描述上述实施例时,还可以使用其他类型的管理程序。选择并描述这些实施例是为了最好的说明本发明的原理及其实际应用,从而使本领域的其他技术人员能够从各实施例以及通过适于预期使用的各改进来理解本发明。

本发明的范围由所附的权利要求及其等同物限定。

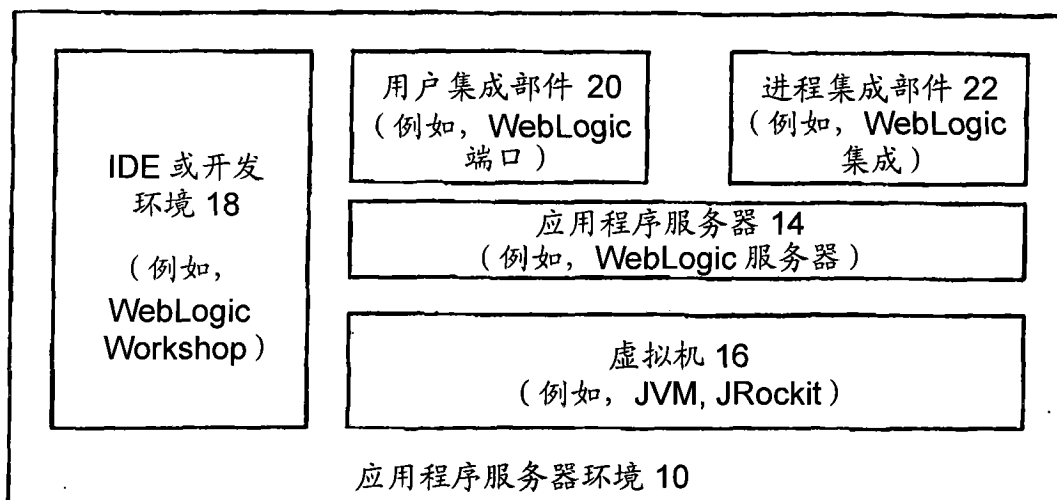


图 1

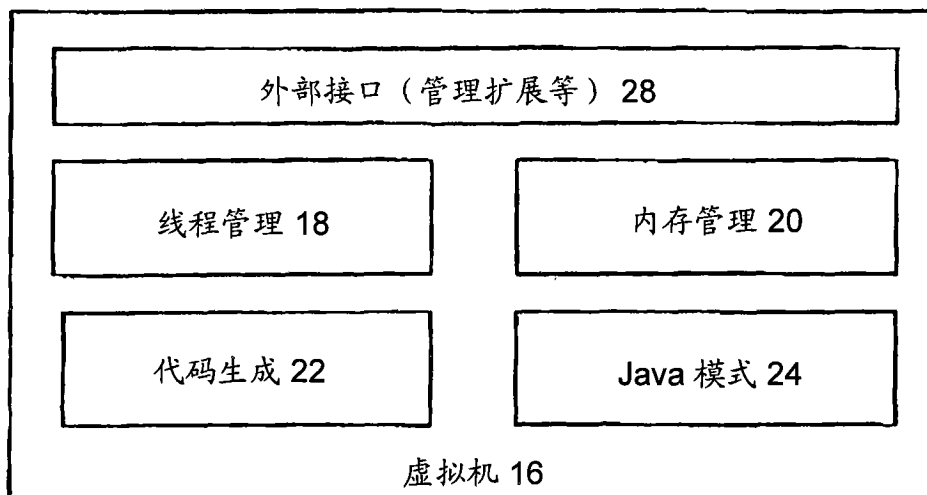


图 2

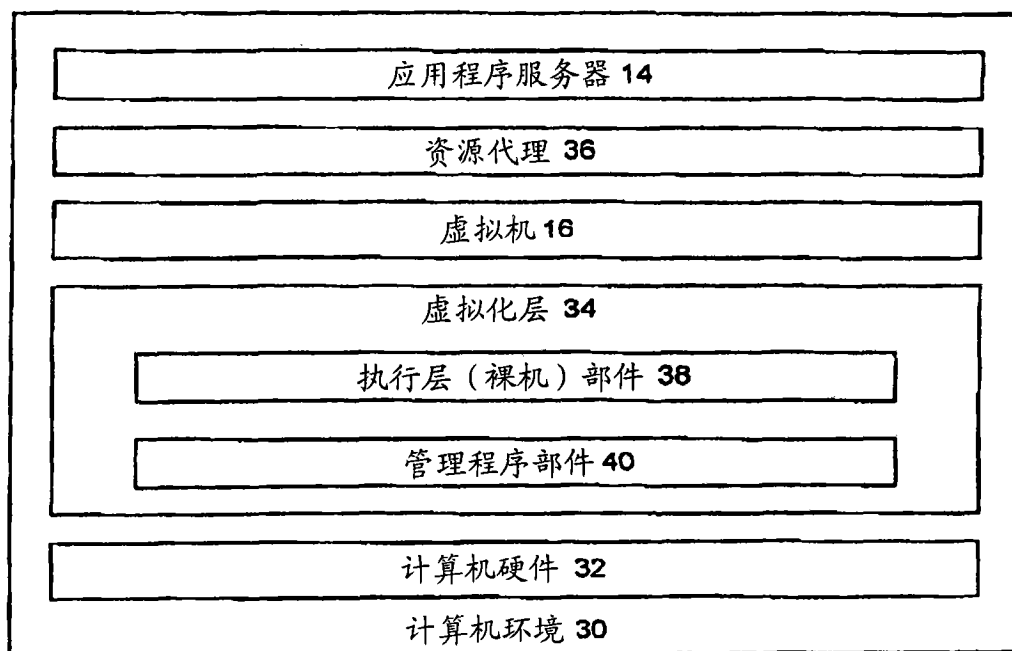


图 3

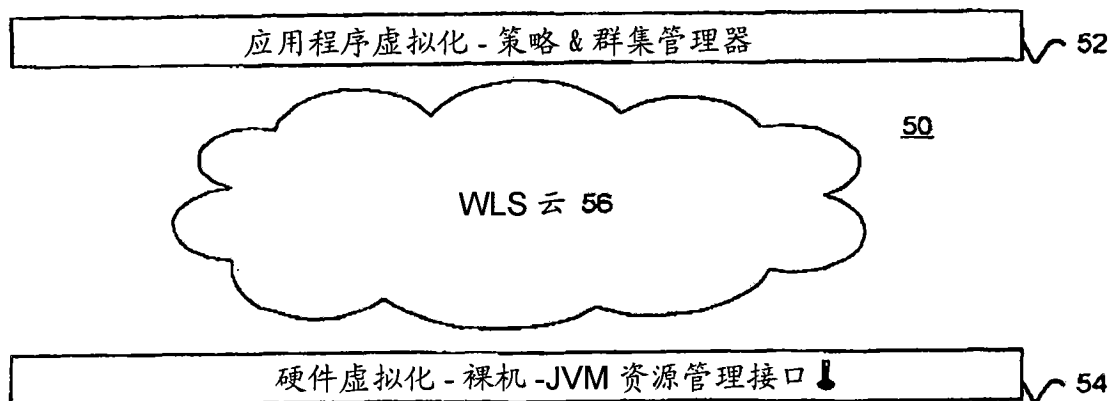


图 4

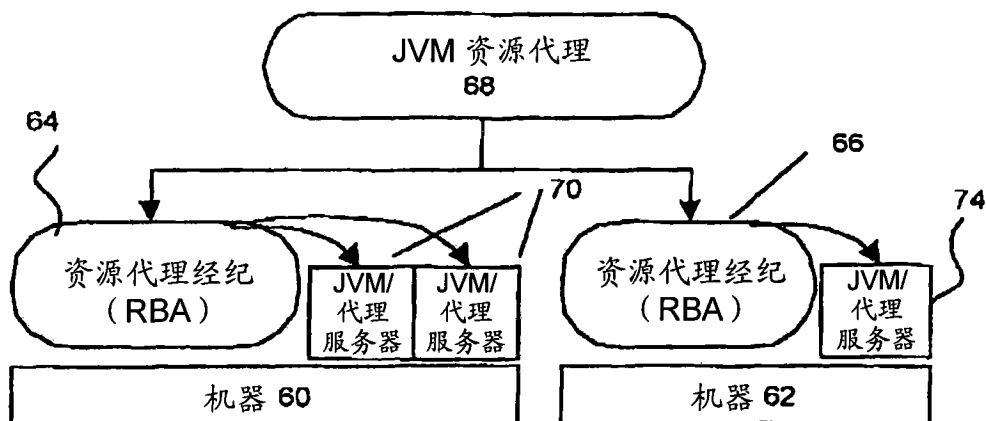


图 5

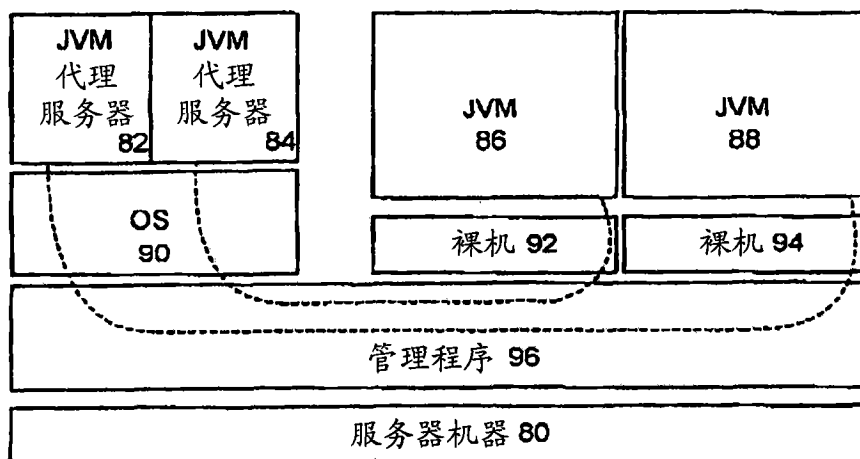
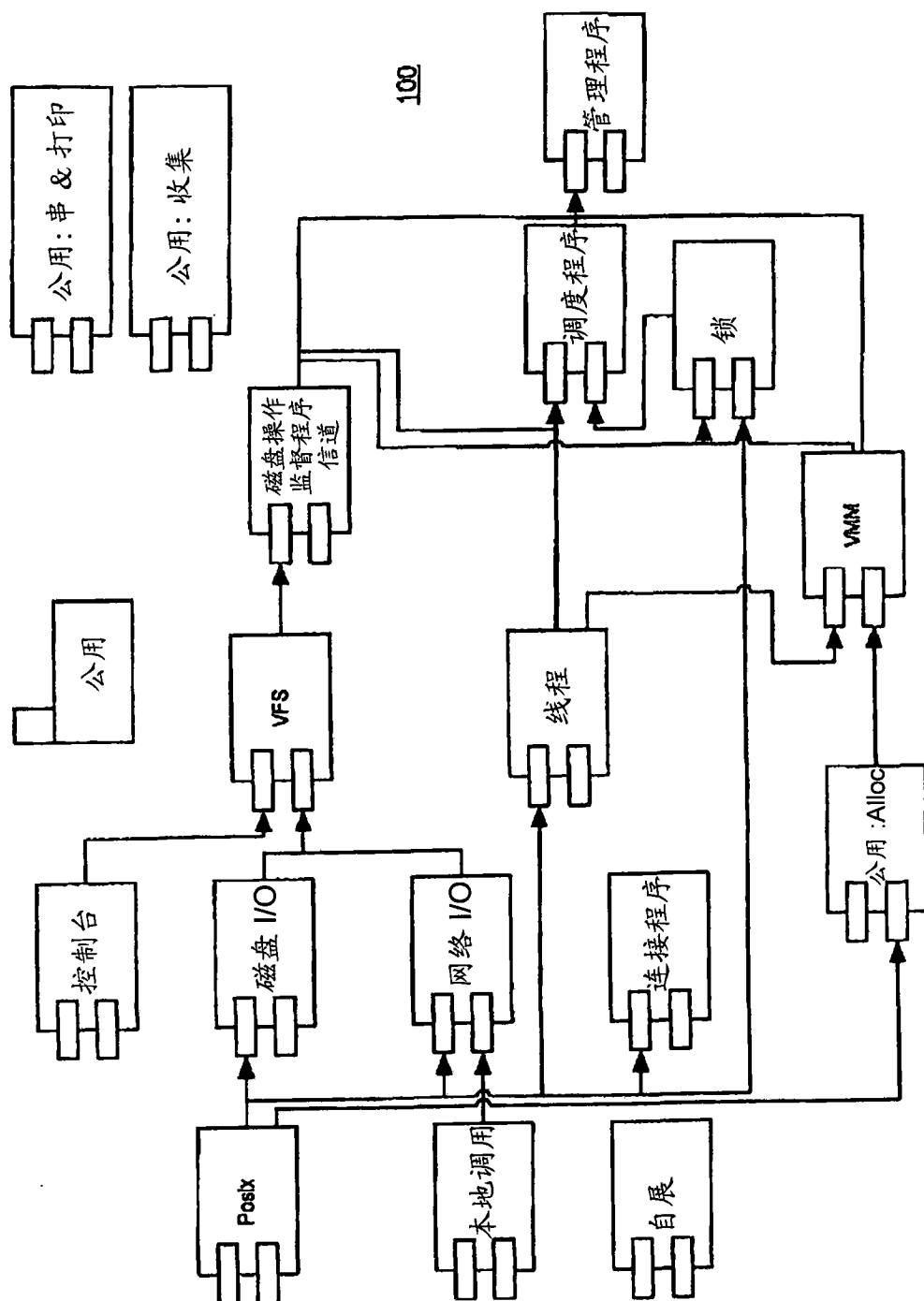


图 6



7 图

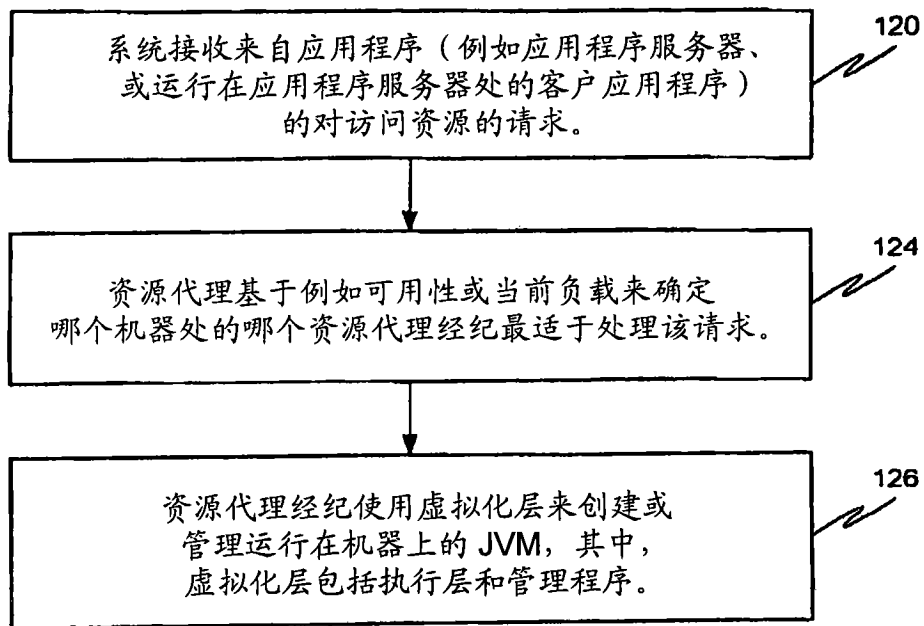


图 8