



(12) 发明专利

(10) 授权公告号 CN 101014036 B

(45) 授权公告日 2010.08.04

(21) 申请号 200710002151.0

1. 2004, 30(1), 2.

(22) 申请日 2007.01.04

审查员 许婵

(30) 优先权数据

11/344,606 2006.01.31 US

(73) 专利权人 国际商业机器公司

地址 美国纽约

(72) 发明人 康斯坦丁·M·亚当

戈瓦尼·帕斯费斯

迈克尔·J·斯普里特泽

马尔格扎塔·斯特恩德 唐春强

(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 李镇江

(51) Int. Cl.

H04L 29/06 (2006.01)

(56) 对比文件

杨俊秀,王继曾,赵文来,王娜. 一种改进的动态负载均衡算法. 兰州理工大学学报 30

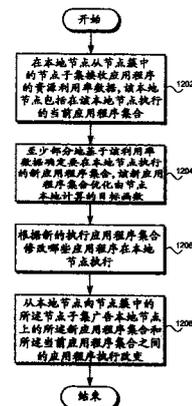
权利要求书 2 页 说明书 10 页 附图 12 页

(54) 发明名称

用于节点簇的分散应用程序资源分配的方法与系统

(57) 摘要

一种分散处理,用于确保在两种类型的同时资源请求下服务器上应用程序的动态放置,这两种类型的同时资源请求分别是依赖于放置到应用程序上的负载的资源请求和独立于该负载的资源请求。对应用程序的需求(负载)随时间变化,目标是要满足所有需求,同时尽可能少地改变解决方案(向服务器的应用程序分配)。



1. 一种用于节点簇的分散应用程序资源分配的方法,该方法包括:

在本地节点从节点簇中的节点子集接收应用程序的资源利用率数据,该本地节点包括在该本地节点执行的当前应用程序集合;

至少部分地基于该利用率数据确定要在该本地节点执行的新应用程序集合,该新应用程序集合优化由本地节点在本地计算的目标函数;

根据所述新的执行应用程序集合修改关于哪些应用程序在该本地节点执行的信息;及

从该本地节点向节点簇中的所述节点子集发送本地节点上所述新应用程序集合和所述当前应用程序集合之间的应用程序执行改变,其中确定要在本地节点执行的新应用程序集合的步骤还包括:

定义在本地节点执行的运行应用程序集合;

以交付密度的升序排序所述运行应用程序,其中交付密度定义为交付给应用程序的CPU量与应用程序所使用的存储器之比;

定义备用应用程序集合,所述备用应用程序包括在所述节点子集上执行但不在本地节点上执行的应用程序和不在节点簇中任何地方执行的应用程序;

通过从所述备用应用程序集合中除去对其没有未满足需求的应用程序来过滤所述备用应用程序集合,所述未满足需求定义为资源的需求与提供之间的差值;

以未满足密度的降序排序所述备用应用程序集合,所述未满足密度定义为所述未满足需求与应用程序所使用的存储器之比;

通过尝试从本地节点分配尽可能多的负载到所述节点子集来偏移负载;

通过用来自排序后的备用应用程序集合的应用程序连续代替来自排序后的运行应用程序集合的应用程序,建立多个本地配置;以及

从所述本地配置选择最大化本地节点上的CPU利用率的优化配置。

2. 如权利要求1所述的方法,其中所述接收、确定、修改与发送操作是在节点簇中的每个节点独立、异步执行的。

3. 如权利要求1所述的方法,还包括使用覆盖构造算法为该本地节点识别所述节点子集,该节点子集是该本地节点的逻辑邻居。

4. 如权利要求1所述的方法,其中接收资源利用率数据的动作包括:

从所述节点子集中的每个节点接收活动应用程序列表;

接收用于列表中的每个活动应用程序的资源提供与需求;及

接收用于列表中的每个活动应用程序的资源利用率数据。

5. 如权利要求1所述的方法,其中从本地节点向节点簇中的所述节点子集发送应用程序执行改变的步骤包括使用杂谈协议,在杂谈协议中每个节点聚合该节点在预定时间间隔内接收或发起的消息,然后在单个消息中向邻居节点发送聚合消息。

6. 如权利要求1所述的方法,还包括:

对于本地节点开始和停止的每个应用程序,锁定离开两个逻辑跳远的所有节点并防止所述离开两个逻辑跳远的所有节点开始和停止该应用程序;及

从指定的节点获得防止其它节点开始当前未在节点簇中任何地方运行的应用程序的锁。

7. 一种用于节点簇的分散应用程序资源分配的系统,该系统包括:

在本地节点从节点簇中的节点子集接收应用程序的资源利用率数据的装置,其中该本

地节点包括在该本地节点执行的当前应用程序集合；

至少部分地基于该利用率数据确定要在该本地节点执行的新应用程序集合的装置，其中该新应用程序集合优化由本地节点在本地计算的目标函数；

根据所述新的执行应用程序集合修改关于哪些应用程序在该本地节点执行的信息的装置；以及

从该本地节点向节点簇中的所述节点子集发送本地节点上的所述新应用程序集合和所述当前应用程序集合之间的应用程序执行改变的装置，其中至少部分地基于该利用率数据确定要在该本地节点执行的新应用程序集合的装置被配置为：

定义在本地节点执行的运行应用程序集合；

以交付密度的升序排序运行应用程序，其中交付密度定义为交付给应用程序的 CPU 量与应用程序所使用的存储器之比；

定义备用应用程序集合，所述备用应用程序包括在所述节点子集上执行但不在本地节点上执行的应用程序和不在节点簇中任何地方执行的应用程序；

通过从所述备用应用程序集合中除去对其没有未满足需求的应用程序来过滤所述备用应用程序集合，所述未满足的需求定义为资源的需求与提供之间的差值；

以未满足密度的降序排序所述备用应用程序集合，所述未满足密度定义为未满足需求与应用程序所使用的存储器之比；

通过尝试从本地节点分配尽可能多的负载到所述节点子集来偏移负载；

通过用来自排序后的备用应用程序集合的应用程序连续代替来自排序后运行应用程序集合的应用程序，建立多个本地配置；及

从所述本地配置选择最大化本地节点上的 CPU 利用率的优化配置。

8. 如权利要求 7 所述的系统，被配置为在节点簇中的每个节点独立、异步执行所述接收、确定、修改与发送操作。

9. 如权利要求 7 所述的系统，还包括使用覆盖构造算法为本地节点识别所述节点子集的装置，其中所述节点子集是该本地节点的逻辑邻居。

10. 如权利要求 7 所述的系统，其中在本地节点从节点簇中的节点子集接收应用程序的资源利用率数据的装置被配置为：

从节点子集中的每个节点接收活动应用程序列表；

接收用于列表中的每个活动应用程序的资源提供与需求；以及

接收用于列表中的每个活动应用程序的资源利用率数据。

11. 如权利要求 7 所述的系统，其中从该本地节点向节点簇中的所述节点子集发送本地节点上的所述新应用程序集合和所述当前应用程序集合之间的应用程序执行改变的装置被配置为：在杂谈协议中每个节点聚合该节点在预定时间间隔内接收或发起的消息，然后在单个消息中向邻居节点发送聚合消息。

12. 如权利要求 7 所述的系统，还包括：

对于本地节点开始和停止的每个应用程序，锁定离开两个逻辑跳远的所有节点并防止所述离开两个逻辑跳远的所有节点开始和停止该应用程序的装置；以及

从指定的节点获得防止其它节点开始当前未在节点簇中任何地方运行的应用程序的锁的装置。

用于节点簇的分散应用程序资源分配的方法与系统

技术领域

[0001] 本发明涉及计算机簇 (cluster) 上的应用程序放置, 更具体而言, 涉及当应用程序的负载波动时以分布方式的分散按需应用程序资源分配。

背景技术

[0002] 随着万维网 (WWW 或简单地是“Web”) 与数据服务的外部获取 (outsource) 的激增, 计算服务中心的大小和复杂性都增加了。例如, 服务中心可以包括称为服务器池 (farm) 的服务器集合, 该服务器池运行对特定应用程序的处理, 该服务器集合称为簇。这种中心提供多种服务, 例如 Web 内容寄放 (host)、电子商务、web 应用程序及商务应用程序。管理这种中心是有挑战性的, 因为服务提供者必须面对不可预测的负载强度和多种所提供的服务与应用程序之间的分布而管理提供给竞争应用程序的服务质量。已经介绍了几种处理这些操作性管理问题的管理软件包。这些软件系统提供包括监视、需求估计、负载平衡、动态提供、服务区分、优化资源分配及动态应用程序放置的功能。最后一个功能, 即动态应用程序放置, 是本发明的主题。

[0003] 服务请求一般是通过执行应用程序集合中每一个应用程序的一个或多个实例满足的。应用程序包括对静态与动态 web 内容的访问、企业应用程序及对数据库服务器的访问。应用程序可以由 HTTP (超文本传输协议) web 服务器、小服务程序、企业 Java Beans (EJB) 或数据库查询提供。当用于特定应用程序的服务请求的个数增加时, 负责放置应用程序的管理软件部署应用程序的附加实例, 以便适应增加的负载。具有允许应用程序实例动态部署与除去的按需管理环境常常是很重要的。问题是动态改变应用程序实例的个数, 从而在最小化开始与停止应用程序实例的开销的同时满足动态负载。

[0004] 当应用程序的负载波动时与服务器池中应用程序处理的自动实例化关联的一个问题是每个服务器机器可以运行有限数量的应用程序处理。用于特定应用程序的请求消息在该应用程序的所有实例间分开。因此, 当应用程序实例使用不同的服务器时, 簇的大小直接影响在没有性能降低的情况下簇可以维持的负载量。

[0005] 当簇的大小不够时, 应用程序用户体验性能降低或故障, 导致违背服务水平协定 (SLA)。目前, 为了避免违背 SLA, 应用程序提供者通常过多地提供应用程序实例的数量, 以便处理峰值负载。这导致正常运行条件下的资源利用率差。动态分配通过根据应用程序当前负载与 SLA 目标自动地重新在应用程序间分配服务器减轻浪费容量的问题。

[0006] 目前可用的大部分放置算法是集中式的。集中式的方法通常不具有对发生在两次放置操作之间的改变立即作出反应的能力。在集中式解决方案中, 单个控制器常常需要处理来自几个节点的约束。此外, 每个应用程序一般都需要特定的时间开始或停止。在这个时间中, 重新配置处理会占用本地机器上的大部分 CPU 功率, 因此会部分地破坏其服务能力。集中式解决方案一般需要以不会同时发生的方式调度这些改变的增强, 以避免系统总处理能力的急剧下降。

发明内容

[0007] 本发明解决了服务器池中应用程序处理的自动实例化的问题,以便当应用程序的负载波动时允许服务器池动态调整应用程序处理的数量。与集中式解决方案相比,应用程序放置的分散解决方案可以有多个概念性优点。首先,由于算法在系统中的每个机器上独立和异步地运行,因此分散放置使系统能够面对外部事件持续地重新配置。其次,由于每个节点只管理本地资源,因此分散解决方案的复杂性比较低。第三,由于与放置算法运行在单个机器上的集中式解决方案相比,每个机器具有完全相同的功能,因此在分散情况下没有配置开销。本发明有利地优化了服务器上计算应用程序的动态放置,以便在尽可能小地改变应用程序分配的同时满足整个应用程序的需求。

[0008] 本发明的一个示例方面是用于节点簇的分散应用程序资源分配的方法。该方法包括配置成在本地节点从节点簇中的节点子集接收应用程序的资源利用率数据的接收操作。本地节点包括其正执行的当前应用程序集合。确定操作形成要在本地节点执行的新应用程序集合。该新应用程序集合优化由本地节点本地计算的目标函数并至少部分地基于利用率数据。修改操作根据所述新的执行应用程序集合修改哪些应用程序在本地节点执行。发送操作从本地节点向节点簇中的所述节点子集广告在本地节点的所述新的应用程序集合与所述当前应用程序集合之间的应用程序执行改变。

[0009] 本发明的另一示例方面是用于节点簇的分散应用程序资源分配的系统。该系统包括配置成执行计算机程序的处理器和耦合到该处理器并配置成通过计算机网络发送与接收数据的网络接口。此外,存储设备包含计算机程序。该计算机程序包括计算机可执行指令,该计算机可执行指令配置成在本地节点从节点簇中的节点子集接收应用程序的资源利用率数据;至少部分地基于该利用率数据确定要在本地节点执行的新应用程序集合,该新应用程序集合优化由本地节点本地计算的目标函数;根据所述新的执行应用程序集合修改哪些应用程序在本地节点执行;及从本地节点向节点簇中的所述节点子集发送在本地节点的所述新的应用程序集合与所述当前应用程序集合之间的应用程序执行改变。

[0010] 本发明还有一个示例方面是包含在有形介质中的计算机程序产品。该计算机程序产品包括计算机可读程序代码,该计算机可读程序代码配置成使程序执行:在本地节点从节点簇中的节点子集接收应用程序的资源利用率数据;至少部分地基于利用率数据确定要在本地节点执行的新应用程序集合,该新应用程序集合优化由本地节点本地计算的目标函数;根据所述新的执行应用程序集合修改哪些应用程序在本地节点执行;及从本地节点向节点簇中的所述节点子集发送在本地节点的所述新的应用程序集合与所述当前应用程序集合之间的应用程序执行改变。

[0011] 如在附图中所说明的,本发明的以上及其它特征、使用与优点将从以下本发明各种实施方式的更具体描述中变得显而易见。

附图说明

[0012] 图 1 示出了示例系统的主要组件及其物理互连的框图。

[0013] 图 2 示出了系统组件之间逻辑连接的框图。

[0014] 图 3 是显示运行在作为示例系统一部分的每个节点上的软件程序的框图。

[0015] 图 4 是显示全局放置矩阵的示例结构的框图。

[0016] 图 5 示出了说明其中节点从其邻居检索状态信息的示例方法的流程图。

[0017] 图 6 是说明根据本发明一种实施方式分散放置算法的重新配置阶段逻辑的流程图。

[0018] 图 7 是说明根据本发明一种实施方式分散放置算法的提交 (commit) 阶段逻辑的流程图。

[0019] 图 8 是说明根据本发明一种实施方式的放置更新在系统中传播的方式的流程图。

[0020] 图 9 是说明根据本发明一种实施方式的当重新配置现有应用程序时放置更新被串行化以维护系统稳定性的方式的流程图。

[0021] 图 10 是说明根据本发明一种实施方式的当激活新应用程序时放置更新被串行化以维护系统稳定性的方式的流程图。

[0022] 图 11 示出了体现本发明的网络节点的说明性实施方式。

[0023] 图 12 示出了如本发明预期的用于节点簇的分散应用程序资源分配的示例流程图。

[0024] 具体实施方式

[0025] 以下描述具体描述如何采用本发明来优化服务器上计算应用程序的动态放置,以满足整体的应用程序需求。贯穿本发明的描述都参考图 1-11。当参考附图时,所示出的相同的结构和元素都使用相同的标号指示。

[0026] 问题阐述

[0027] 动态应用程序放置问题如下阐述:给出具有存储器容量 $\Gamma_1, \dots, \Gamma_m$ 和服务能力(每个单位时间可以服务的请求个数) Q_1, \dots, Q_m 的 m 个服务器 $1, \dots, m$ 。还给出具有存储器需求 Y_1, \dots, Y_n 的 n 个应用程序 $1, \dots, n$ 。应用程序 j 必须在时间间隔 t 内为一定数量的请求 ω_{jt} 服务。

[0028] 在时间 t 对问题的可行解决方案是向服务器分配应用程序的工作量。每个应用程序可以分配给多个服务器(在多个服务器上复制)。对于应用程序 j 分配到的每个服务器 i , 解决方案必须指定这个服务器为这个应用程序处理的请求个数 ω_{ijt} 。对于所有应用程序 j 和时间步长 t , $\sum_i \omega_{ijt}$ 都必须等于 ω_{jt} 。对于每个服务器, 必须考虑存储器与处理约束。分配给服务器的应用程序的存储器需求总和不能超过其存储器 Γ_i 和 $\sum_i \omega_{ijt}$, 即, 在时间步长 t 中由这个服务器服务的请求总数不能超过 Ω_i 。应当指出, 应用程序向服务器的每次分配(拷贝)都导致全存储器成本, 而处理负载在拷贝之间分配。

[0029] 目标是找出与在时间步长 $t-1$ 的解决方案差别不大的在时间步长 t 的解决方案。更正式地说, 对于每种可行的解决方案, 关联二分 (bipartite) 图 (A, S, E_t) , 其中 A 表示在时间步长 t 分配给服务器 i (或在服务器 i 上有拷贝) 的应用程序 j 的集合。目标函数是最小化 $|E_t \ominus E_{t-1}|$, 即, 两个边缘集合的对称差分的基数。这是在时间 t 必须关掉或加载的应用程序实例的个数。

[0030] 系统模型

[0031] 本发明的一种实施方式是在图 1、2 和 3 中总体说明的网络系统中实现的。图 1 示出了由本发明预期的示例系统的物理基础结构 102。物理基础结构 102 包括耦合到计算机网络 106 的一个或多个进入点 104。计算机网络 106 可以是局域网 (LAN)、广域网 (WAN) 或

其组合。预期计算机网络 106 可以配置为例如因特网的公共网络和 / 或例如内联网或其它专有通信系统的专用网。本领域技术人员已知的各种拓扑结构和协议都可以由网络 106 采用,例如 TCP/IP 和 UDP。此外,计算机网络 106 可以包括本领域已知的各种联网设备,例如路由器、交换机、网桥 (bridge)、中继器等。

[0032] 物理基础结构 102 还附加地包括多个节点 108-118。这些组件使用如路由器、交换机或集线器 120-124 的联网设备互连。进入点 104 是将进入的请求重定向到节点 108-118 的交换机。多种类型的硬件 (例如计算机、硬件层 4-7 交换机或者甚至是大型机) 都可以执行这种功能。节点 108-118 可以是桌面计算机、服务器、膝上型电脑或任何其它包括 CPU、存储器及可以连接到网络的硬件设备。

[0033] 图 2 示出了在图 1 中所示物理基础结构之上配置的示例系统的逻辑拓扑结构 202。进入点 104 通过逻辑链路连接到大量节点 (可能是系统中的所有节点)。为了说明,进入点和节点之间的逻辑链路在图 2 中示为虚线。进入点使用如循环 (round robin) 或随机转发的各种转发策略向节点转发进入的请求。节点 108-118 以每个节点只与其邻居通信的逻辑覆盖来自组织。因此,节点簇中的节点子集是对应于节点的邻居定义的。节点之间的逻辑对等链路示为实线。与系统中的节点总数相比,节点的邻居个数通常很小。任何覆盖构造算法都可以用于向节点分配邻居组。结果产生的覆盖可以是各种类型,例如小世界与随机图。逻辑链路可以几种方式实现。一种可能是打开处于逻辑链路末端的节点之间的 TCP 连接。另一种可能是为每个节点维护本地邻居列表并限制到这些邻居的通信。

[0034] 在一种实施方式中,节点 108-118 维护相对稳定的覆盖邻居组并在每个放置周期中收集状态信息。在另一实施方式中,节点 108-118 可能不需要使用稳定的邻居组。相反,节点 108-118 可以运行杂谈协议来发现系统中的其它节点并在不同的放置周期中对不同的节点收集状态信息。

[0035] 图 3 示出了在单个节点上执行的示例软件 302。在本发明的一种实施方式中,在分散放置算法的环境下,所有节点都具有完全相同的功能。节点可以运行一种或多种应用程序处理 304-308。每种应用程序处理用于一种类型应用程序 (该应用程序类型由 A_1 、 A_2 、...、 A_n 指示) 的请求。

[0036] 每个应用程序可以由两种类型的参数表示其特征:(1) 运行应用程序所需的独立于负载的资源需求,及 (2) 作为放置到应用程序的外部负载或需求的函数的依赖于负载的需求。独立于负载的需求的例子是存储器、通信信道及贮存器。依赖于负载的需求的例子是当前或发射的请求率、CPU (中央处理单元) 周期、磁盘活动性及执行线程的个数。

[0037] 类似地,节点 (例如,服务器) 可以由两个参数表示其特征:(1) 表示可用于在节点上寄放应用程序的资源量的独立于负载的容量,及 (2) 表示处理应用程序服务的请求的可用容量的依赖于负载的容量。

[0038] 放置执行器 310、放置控制器 312 及应用程序建档器 (profiler) 314 是配置成提供放置功能性的软件对象。放置执行器 310 具有停止或开始应用程序处理的能力。应用程序建档器 314 收集用于每个本地应用程序的统计数据,例如请求到达率、一个应用程序实例使用的总存储器及应用程序请求消耗的平均 CPU 周期数。在本发明的特定实施方式中,应用程序建档器 314 定义抽象元素集合的依赖负载的能力的集合和独立于负载能力的集合,如以下更具体讨论的。

[0039] 放置控制器 312 包含分散放置算法的内核逻辑。放置控制器 312 基于目标函数动态重新配置每个节点上应用程序的放置,以优化所有节点上应用程序的全局放置。

[0040] 放置控制器 312 在每个节点上独立并异步地执行。放置算法的两次执行之间的时间在此称为执行周期。以下具体描述放置算法。

[0041] 每个节点维护全局放置矩阵 P 的副本。全局放置矩阵将多个节点和多个应用程序描述为抽象元素集合。全局放置矩阵 402 的示例结构在图 4 中示出。矩阵的每一行对应于一个应用程序,每一列对应于一个节点。矩阵 P 的元素如下定义:如果应用程序 a 在节点 n 上运行则 $P_{a,n} = 1$, 否则 $P_{a,n} = 0$ 。更新与维护全局放置矩阵的处理是完全分散的。

[0042] 分散的放置算法

[0043] 放置控制器在三个阶段运行放置算法。首先,放置控制器收集关于系统当前状态的(部分)信息。接下来,基于这种信息,放置控制器决定在要来(incoming)的执行周期中哪些应用程序应当在本地节点上运行。最后,放置控制器在系统中分散对全局放置矩阵的更新的组,该组反映了放置算法的本地决定。

[0044] 收集状态信息

[0045] 为了确保可量测性(scalability),每个节点都从小的邻居集合中检索状态信息。覆盖构造机制建立定义每个节点的邻居的逻辑拓扑结构。

[0046] 图 5 说明了本发明预期的状态信息在邻居之间交换的一种方式。假定节点 502 和 504 是逻辑邻居。如箭头 510 所表示的,运行在节点 504 上的放置控制器 506 从运行在节点 502 上的放置控制器 508 请求状态信息。当接收到请求时,放置控制器 508 从放置执行器 512 请求活动应用程序列表。如箭头 516 所表示的,作为响应,放置执行器 512 检索该列表并将其发送到放置控制器 508。如箭头 520 和 522 所表示的,放置控制器 508 还从应用程序建档器 518 获得应用程序统计数据。在这种处理的结束时,放置控制器 508 收集以下信息:本地活动应用程序的列表 ($a_1 \dots a_m$)、交给每个应用程序的 CPU 周期数 ($\omega_{a_1}^{\text{delivered}} \dots \omega_{a_m}^{\text{delivered}}$)、每个应用程序的存储器需求 ($\gamma_{a_1} \dots \gamma_{a_m}$)、每个活动应用程序的本地经历(experienced)需求 ($\omega_{a_1}^{\text{requested}} \dots \omega_{a_m}^{\text{requested}}$) 及对在网络中任何地方都没有提供、在全局放置矩阵中没有活动项的应用程序的本地经历需求。如箭头 524 所表示的,放置控制器 508 将这种信息发送回放置控制器 506。这完成了两个逻辑邻居之间的信息交换。

[0047] 除了从其邻居检索信息,放置控制器 506 还从本地放置执行器 526 和本地应用程序建档器 528 收集本地信息。

[0048] 重新配置阶段

[0049] 在图 6 中,说明了由本发明一种实施方式执行的重新配置阶段的流程图。如以下具体讨论的,流程图的操作允许在一个或多个依赖于负载的资源约束和一个或多个独立于负载的资源约束下通过动态重新配置应用程序在节点上以分布方式的放置从而优化矩阵的多重性实现的分散的按需应用程序资源分配。应当注意,所示出的逻辑操作可以以硬件或软件或者其组合实现。实现是依赖于实现本发明的系统的性能需求的选择问题。因此,构成在此所述本发明实施方式的逻辑操作可选地称为操作、步骤或模块。

[0050] 在建立操作 602,放置控制器取在前一阶段中收集到的邻居状态作为输入,并建立运行应用程序的集合 $R = \{r_1 \dots r_r\}$ 和备用应用程序的集合 $S = \{s_1 \dots s_s\}$ 。R 包含本地节点上当前活动的应用程序。S 包含在该节点的邻居上运行但不在该节点本身上运行的应用

程序或者在系统中任何地方都没有提供的应用程序。S 是使用在前一阶段收集到的邻居信息建立的。

[0051] R 中的应用程序以其密度的递增次序排序,该密度等于交给应用程序 r 的负载除以 r 的存储器使用 ($\omega_r^{\text{delivered}} / \gamma_r$)。S 中的应用程序以其剩余密度的递减次序排序,该剩余密度等于应用程序 s 的未满足需求除以 s 的存储器使用: $\sum_{n \in \text{neighbors}} \{ \omega_{ns}^{\text{delivered}} - \omega_{ns}^{\text{requested}} \} / \gamma_s$ 。

[0052] 未满足需求为零的备用应用程序从 S 中除去,因为没有必要为那些应用程序开始附加的实例。当完成建立操作 602 后,控制转移到确定操作 604。

[0053] 在确定操作 604,检查备用集合。如果备用集合为空,则算法完成。如果备用集合包含一个或多个应用程序,则控制转移到偏移操作 606。

[0054] 在偏移操作 606,放置控制器尝试将尽可能多的负载从一个或多个运行的应用程序偏移邻居。当一个或多个邻居 (a) 运行应用程序 A 的实例且 (b) 具有空闲的 CPU 能力时,偏移应用程序 A 的负载是可能的。应当指出,由偏移操作 606 执行的负载偏移处理是可以提高算法性能的可选步骤。在偏移操作 606 完成后,控制转移到初始化操作 608。

[0055] 初始化操作 608 开始控制循环,其中放置控制器以本地 CPU 使用率被最大化的方式计算在本地节点上运行的应用程序的优化集合。在初始化操作 608,应用程序的个数设置成零,新运行集合设置成当前运行集合,而最大 CPU 利用率被设置成初始 CPU 利用率。

[0056] 接下来,控制转移到操作 610、612、614、616 和 618 的循环,在该循环中放置控制器尝试以本地节点上 CPU 使用率最大化的方式用 S 的子集替换 R 的子集。用于两个给定集合 R 和 S 的可能重新配置组合的个数可以非常大。以下启发式算法 (heuristic) 将问题的大小减小到 (r+1) 次迭代和最多 ((r+1) * s) 次运算,其中 r 是 R 的大小,而 s 是 S 的大小。

[0057] 放置控制器运行 (r+1) 次迭代,在这个过程中来检查从 R 停止应用程序并用来自 S 的应用程序替换它们的效果。在迭代描述中提到的开始与停止操作只是假设性的。放置控制器假定一系列开始与停止操作发生,然后它评估这些操作将对本地状态起的效果。

[0058] 在第一次迭代中,控制器不停止任何运行的应用程序。如果本地节点有闲置 CPU 和存储器资源 ($\Omega^{\text{available}} > 0$ 且 $\Gamma^{\text{available}} > 0$),则控制器尝试开始一个或多个备用应用程序。

[0059] 在迭代 k 中,控制器计算在停止运行应用程序 $\{r_1 \dots r_{k-1}\}$ 之后变得可用的存储器与 CPU 资源。然后,控制器向 S 中的应用程序分配可用的资源。最初,节点尝试 (将来自 S 的第一应用程序) 适配到可用存储器 s_1 。如果这个操作成功 ($\gamma_{s_1} \leq \Gamma^{\text{available}}$),则控制器尝试满足 s_1 的全部未满足的 CPU 需求。结果, $\min((\omega_{s_1}^{\text{req}} - \omega_{s_1}^{\text{del}}), \Omega^{\text{available}})$ CPU 周期分配给 s_1 。如果对 s_1 没有足够的可用存储器,则控制器继续 S 中的下一个应用程序。当没有剩余的存储器或 CPU 可分配 ($\Omega^{\text{available}} == 0$ 或者 $\Gamma^{\text{available}} == 0$) 或者当 S 中的所有应用程序都已经被考虑之后,迭代停止。

[0060] 开始或停止应用程序消耗节点的 CPU 资源一定量时间。对于每种配置,改变成本从总的 CPU 使用率中减去。例如,如果开始应用程序消耗本地 CPU 资源 15 秒,且执行周期的长度是 15 分钟,则节点总处理功率的 1.67% 将分配给重新配置处理,而剩余的 98.33% CPU 可用于处理请求。

[0061] 对于在迭代 k 结束时获得的每个集合 R^k ,控制器计算本地 CPU 使用率。最大化本

地 CPU 使用率的集合 R^k 是优化配置,其在设置操作 620 给出。

[0062] 提交并广告配置改变

[0063] 图 7 中的流程图说明了放置算法最后阶段的逻辑。算法在确定操作 702 开始,在那里检查运行应用程序的集合。如果对集合没有改变,则处理结束。但是,如果放置控制器对本地配置进行了任何改变,则控制继续到计算操作 704。在计算操作 704,计算要停止和开始的应用程序集合。例如,令 R 为节点上当前活动应用程序的集合,令 R^k 为在算法的上一(重新配置)阶段由放置控制器计算出的优化配置。计算操作 704 计算要停止的应用程序集合 ($R \setminus R^k$) 和要开始的应用程序集合 ($R^k \setminus R$)。在广告操作 706,节点使用分散机制向系统中的所有节点或节点子集广告改变。

[0064] 在确定操作 708,检查要停止的结果应用程序集合。如果要停止的应用程序集合不空,则控制进行到停止操作 710,在那里该应用程序集合停止。同样,在确定操作 712,如果要开始的应用程序集合不空,则开始操作 714 开始该应用程序集合。

[0065] 对于要变得有效的新配置,放置控制器需要停止 $R \setminus R^k$ 中的应用程序并开始 $R^k \setminus R$ 中的应用程序。开始或停止应用程序消耗大量的节点 CPU 功率一定量的时间。当采取重新配置决定的时刻和当改变变得有效且节点以全容量操作的时刻之间的延迟为:

$$[0066] \quad t^{\text{commit}} = \sum_{\alpha \in R \setminus R^k} t_a^{\text{stop}} + \sum_{\alpha \in R^k \setminus R} t_a^{\text{start}}$$

[0067] 在 t^{commit} 时间间隔内,节点不能以其全容量操作,因为其大量 CPU 功率分配给了重新配置处理(停止和开始应用程序)。为了在成功完成放置算法时通知系统的剩余部分,放置控制器在广告操作 716 中向系统中所有节点或节点子集广告配置改变的完成。依赖于放置矩阵的使用,由放置控制器发布的每个广告消息都可以到达系统中的所有节点或者仅仅是系统中节点的子集。在一种实施方式中,放置改变被广告到系统中的所有节点。广告延迟 t^{notify} 表示广告到达系统中其它节点所需的时间。有多种方式(例如,广播)分散在节点发生的放置改变。在下一部分中,讨论本发明环境下的一种实现。

[0068] 更新和维护全局放置矩阵

[0069] 在本发明的一种特定实现中,节点使用杂谈协议分散放置改变并在全局放置矩阵的本地副本中维护更新的信息。图 8 中的流程图说明了用于将改变传播到放置矩阵的示例过程。每个节点聚合在预定义的聚合时间间隔(例如,1 秒)内从其邻居接收到的所有消息,并将聚合消息的子集重新发送到其每个邻居。例如,假定节点 108 已经完成了运行放置算法并修改了其配置。节点 108 向其邻居(节点 114 和节点 118)发送更新,由线 802 和 804 表示,将改变通知它们。

[0070] 当接收到这些改变消息时,节点 114 和 118 不立即重新发送它们,而是等待聚合时间间隔到期。在聚合时间间隔结束之前由节点 114 和 118 接收或发起的任何其它消息都将与从节点 108 接收到的更新聚合。当它们各自的聚合时间间隔结束时,节点 114 和 118 分别向节点 116 和 110 发送聚合消息,包括从节点 108 接收到的更新(线 806 和 808)。节点 114 和 118 将不向节点 108 重新发送从节点 108 接收到的更新,但是它们将向节点 108 发送在聚合时间间隔中从其它源收集到的消息。类似地,节点 110 和 116 向节点 112 发送由 108 发起的更新,由线 810 和 812 表示。由于引入聚合时间间隔限制了在聚合时间间隔中每个节点发起的消息数量,因此这个过程是高度可量测的。

[0071] 在上一段中描述的杂谈过程以很高的概率确保所有节点将接收到每个广告的改变

变。但是,还是有可能某个节点将不会接收到特定的消息。因此,错误会随时间累计,导致全局放置矩阵的本地副本之间的不一致性。为了防止这种情况发生,每个节点使用上述相同的杂谈协议周期性地发送其活动应用程序的完整列表。接收到这个消息的节点相应地使用它更新放置矩阵的本地拷贝。放置矩阵中的每个项与定时器关联。如果项在预定义的时间阈值内没有更新,则项超时并从放置矩阵删除。

[0072] 进一步提高系统的稳定性

[0073] 以下所述的技术串行化在系统中发生的改变。它们帮助确保在放置过程中没有基于相同信息的同时改变发生。在作出影响相同资源或应用程序的其它决定之前系统组件观察并评估放置决定的影响的意义上,这种可选的处理稳定了系统。有两种类型的锁定请求:(a) 用于在系统中已经运行的应用程序的锁,(b) 用于在系统中任何地方都不运行的应用程序的锁。

[0074] 图 9 中的示例流程图示出了节点获得已经在系统中运行的应用程序的锁的方式。假定节点 118 运行了放置算法并决定停止(或开始)应用程序 A_1 902。还假定没有其它节点设置了用于这个应用程序的锁。节点 118 从其邻居节点 108 和 110 请求用于 A_1 的锁,如在线 904 和 906 中所说明的。如果节点 108 和 110 还没有关于应用程序 A_1 的锁,则它们向其邻居(节点 114 和 112)转发对锁的请求,如分别由线 908 和 910 所表示的。如果节点 112 和 114 还没有关于应用程序 A_1 的锁,则它们设置锁并分别应答来自节点 108 和 110 的请求(线 908 和 910)。当接收到肯定的应答时,节点 108 和 110 设置其锁,并应答节点 118(线 904 和 906)。当接收到来自节点 108 和 110 的肯定应答时,节点 118 获得锁并进行放置改变。

[0075] 对锁的请求从请求的源传播正好两个逻辑跳。锁对于应当由放置算法停止或开始的每个应用程序是必须获得的。如果在任何点对锁的请求失败,则锁预订重新进行,而请求该锁的节点等待并重新运行放置算法。如果对锁的请求成功,则节点进行放置改变,且一旦过程完成就解锁其邻居。为了处理在获得锁后节点可能的失败,锁具有超时时间,在该时间之后锁就过期了。

[0076] 图 10 是说明获得用于激活不在系统中任何地方运行的应用程序的锁的处理的流程图。假定应用程序 A_1 不在系统中任何节点运行,且就象使用散列函数确定的,用于 A_1 的锁持有者是节点 116。节点 116 从每个其它节点接收关于对应用程序 A_1 需求的更新。例如,当运行放置算法时,节点 108 经历对应用程序 A_1 的需求。如由箭头 1002 表示的,节点 108 向节点 116 发送需求。如由箭头 1004 表示的,节点 116 以(在前一个放置周期上计算的)对应用程序 A_1 的全需求应答。节点 108 当建立其备用应用程序集合时,向应用程序 A_1 分配从节点 116 接收到的需求。假定节点 118 遵循相同的过程并在放置算法后决定开始应用程序 A_1 。如由箭头 1006 表示的,节点 118 从节点 116 请求对应用程序 A_1 的锁。节点 116 检查锁的可用性,如果可用,则如由箭头 1008 表示的,向节点 118 发送锁。当接收到锁时,节点 118 使用上述杂谈协议通知其它节点它正开始应用程序 A_1 。如由箭头 1010 表示的,在开始应用程序 A_1 后,节点 118 释放来自节点 116 的锁。

[0077] 参考图 11,示出了根据本发明的网络节点 1102 的说明性实施方式。在其中本发明可能有用的一种网络节点 1102 包含通用计算机。这种计算机的例子包括 Sun Microsystems, Inc. 提供的 SPARC(r) 系统和可以从国际商用机器公司及其它各计算机制

造商获得的基于 Pentium(r) 的计算机。SPARC 是 Sun Microsystems, Inc. 的注册商标,而 Pentium 是 Intel 公司的注册商标。

[0078] 网络节点 1102 包括处理单元 1104、系统存储器 1106 及将系统存储器 1106 耦合到处理单元 1104 的系统总线 1108。系统存储器 1106 包括只读存储器 (ROM) 1108 和随机存取存储器 (RAM) 1110。包含帮助在例如启动过程中在网络节点 1102 的元素之间传输信息的基本例程的基本输入 / 输出系统 (BIOS) 1112 存储在 ROM 1108 中。

[0079] 网络节点 1102 还包括硬盘驱动器 1114、磁盘驱动器 1116 (用于向可拆卸磁盘 1118 读 / 写) 及光盘驱动器 1120 (用于读 CD-ROM 盘 1122 或 / 向从其它光介质读 / 写)。硬盘驱动器 1114、磁盘驱动器 1116 和光盘驱动器 1120 分别由硬盘接口 1124、磁盘接口 1126 和光盘接口 1128 连接到系统总线 1108。驱动器及其关联的计算机可读介质为计算机 104 提供非易失存储器。尽管计算机可读介质指硬盘、可拆卸磁介质和可拆卸光介质,但本领域技术人员应当理解可以由计算机读取的其它类型介质,例如闪存卡,也可以用在说明性节点 1102 中。

[0080] 多个程序模块可以存储在驱动器与 RAM 1110 中,包括操作系统 1130、分散放置应用程序 1132、全局放置矩阵 1134 及其它程序模块与数据 (未示出)。如上面所讨论的,节点 1102 配置成动态重新配置应用程序以分布方式的放置。

[0081] 用户可以通过键盘 1136 和例如鼠标 1138 的定点设备将命令和信息输入到节点 1102。其它输入设备 (未示出) 可以包括麦克风、调制解调器、操纵杆、游戏板、卫星反射器、扫描仪等。这些和其它输入设备常常通过耦合到系统总线 1108 的串行端口接口 1140 连接到处理单元。

[0082] 监视器 1142 或其它类型的显示设备也通过例如视频适配器 1144 的接口连接到系统总线 1108。除了监视器,节点 1102 还可以包括其它外围输出设备 (未示出),例如扬声器和打印机。

[0083] 节点 1102 使用到一个或多个远程设备的逻辑连接在联网环境中运行。远程设备可以是服务器、路由器、对等设备或其它公共的网络节点。节点 1102 当用在网络环境中时,一般通过网络接口 1146 连接到网络 1148。在网络环境中,关于节点 1102 描述的程序模块或其部分可以存储在一个或多个远程存储器存储设备中。

[0084] 转向图 12,示出了如本发明一种实施方式预期的用于节点簇的分散应用程序资源分配的示例流程图。接收操作 1202 配置成在本地节点从节点簇中的节点子集接收应用程序的资源利用率数据。在本发明的特定实施方式中,每个节点从其每个邻居检索以下信息:

[0085] (a) 活动 (执行) 应用程序的列表,

[0086] (b) 这些活动应用程序中每一个的需求与提供,及

[0087] (c) 对在系统中任何地方都没有提供的其邻居不能使用全局放置矩阵路由的应用程序的需求。

[0088] 基于这种信息,节点本地建立两组应用程序:运行的应用程序和备用的应用程序。

[0089] 本地节点包括其正执行的当前应用程序集合。用运行的应用程序集合和备用的应用程序集合,使用确定操作 1204 形成要在本地节点执行的新的应用程序集合 (这种操作的细节在上面讨论过了)。该新的应用程序集合配置成优化由本地节点本地计算的目标函数

并至少部分地基于使用率数据。在本发明的一种实施方式中，目标函数可以是最大化本地节点 CPU 使用率的函数。但是，预期其它的目标函数也可以由本发明使用。例如，目标函数可以是最小化功耗或者关于 CPU 需求的任何其它函数。

[0090] 修改操作 1206 根据新的执行应用程序集合修改哪些应用程序在本地节点执行。发送操作 1208 从本地节点向节点簇中的所述节点子集广告本地节点上新应用程序集合和当前应用程序集合之间的应用程序执行改变。

[0091] 本发明以上描述的提出是为了说明与描述。这不打算是穷尽的或者要将本发明限定到所公开的精确形式，鉴于以上教义，其它修改与变体也是可能的。因此，所公开实施方式的选择与描述是为了最好地解释本发明的原理及其实际应用，由此使本领域其它技术人员以适合于预期的特定使用的各种实施方式及各种修改最好地使用本发明。所附权利要求是要被解释成包括本发明的其它可选实施方式，除了由现有技术所限定的以外。

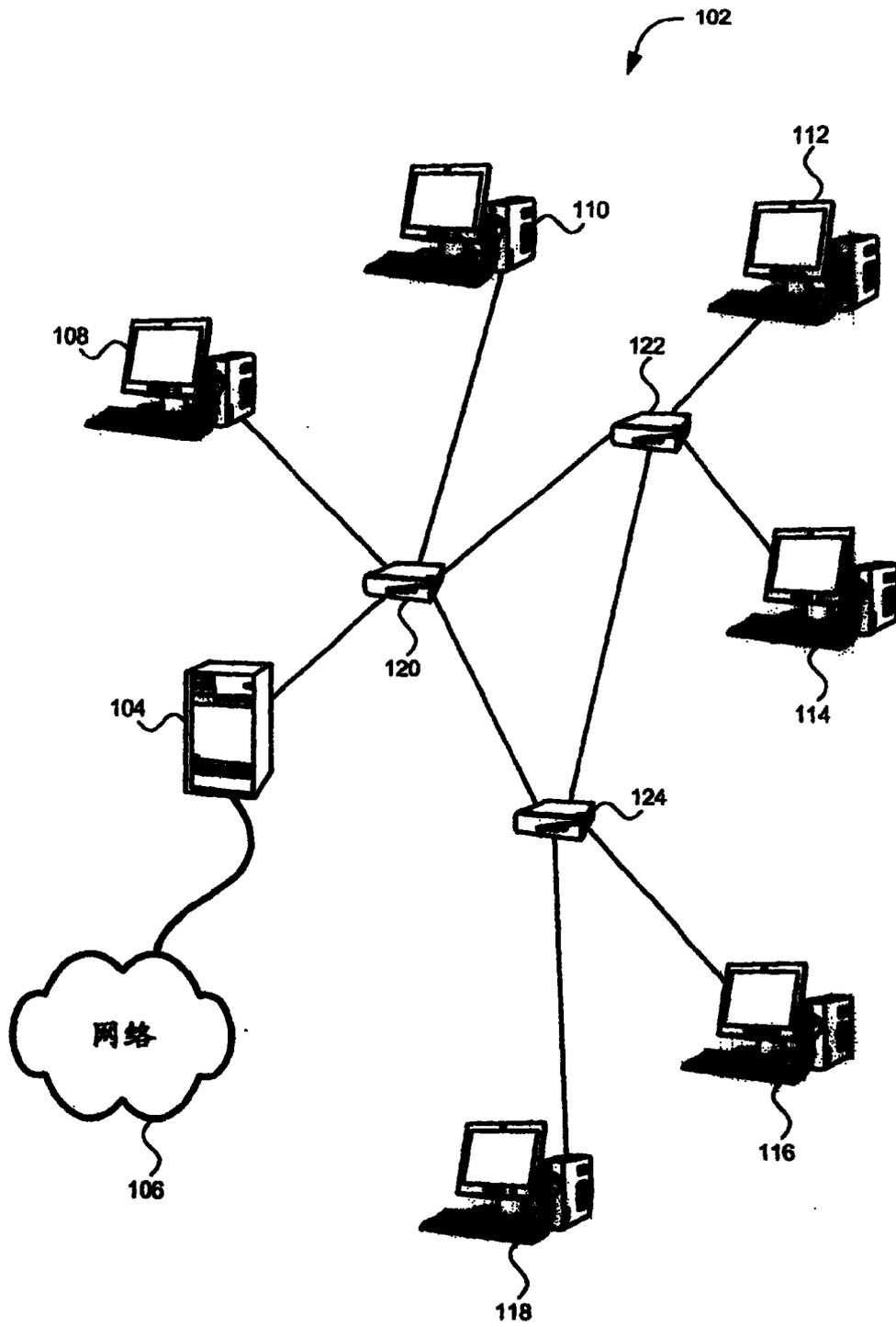


图 1

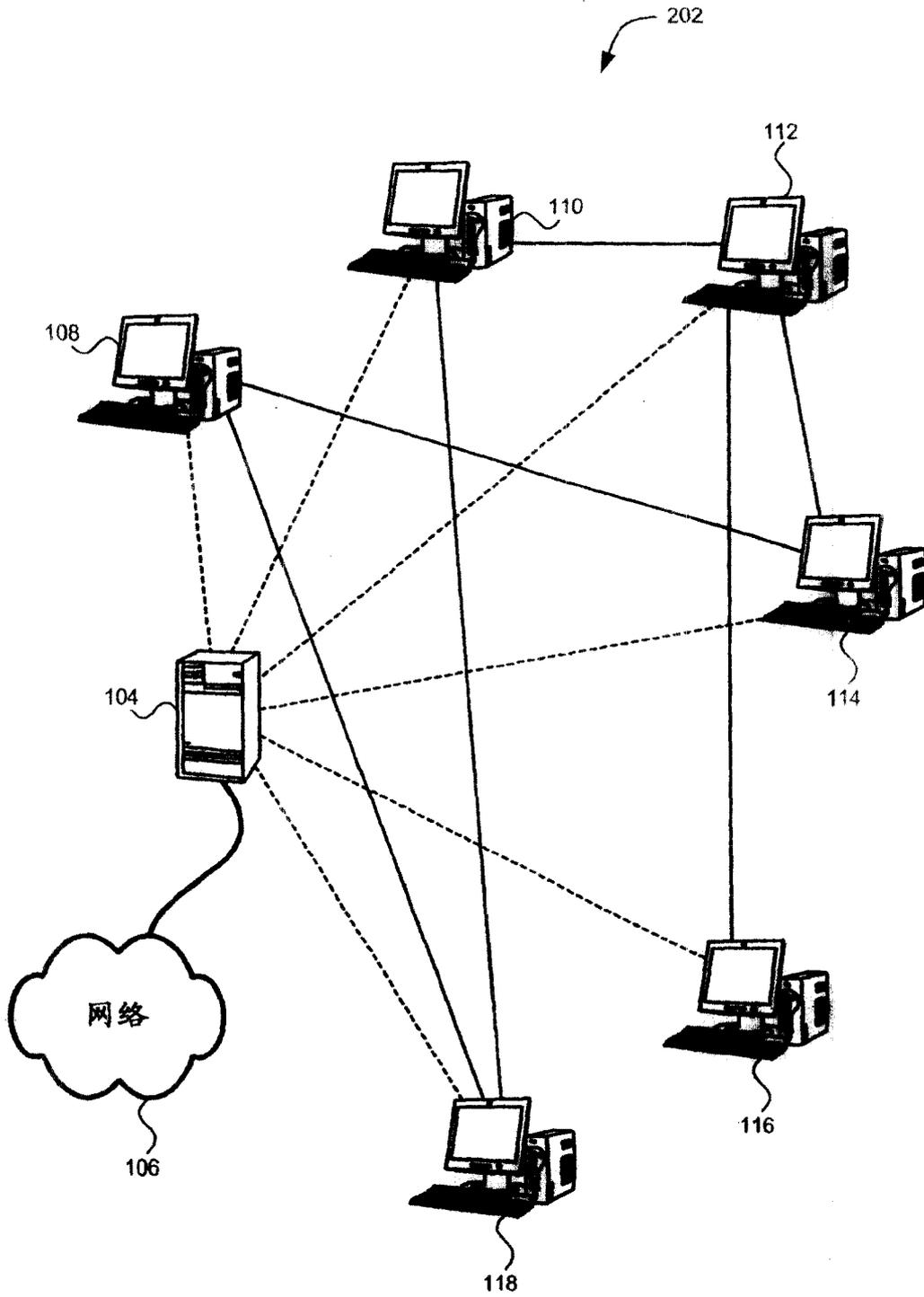


图 2

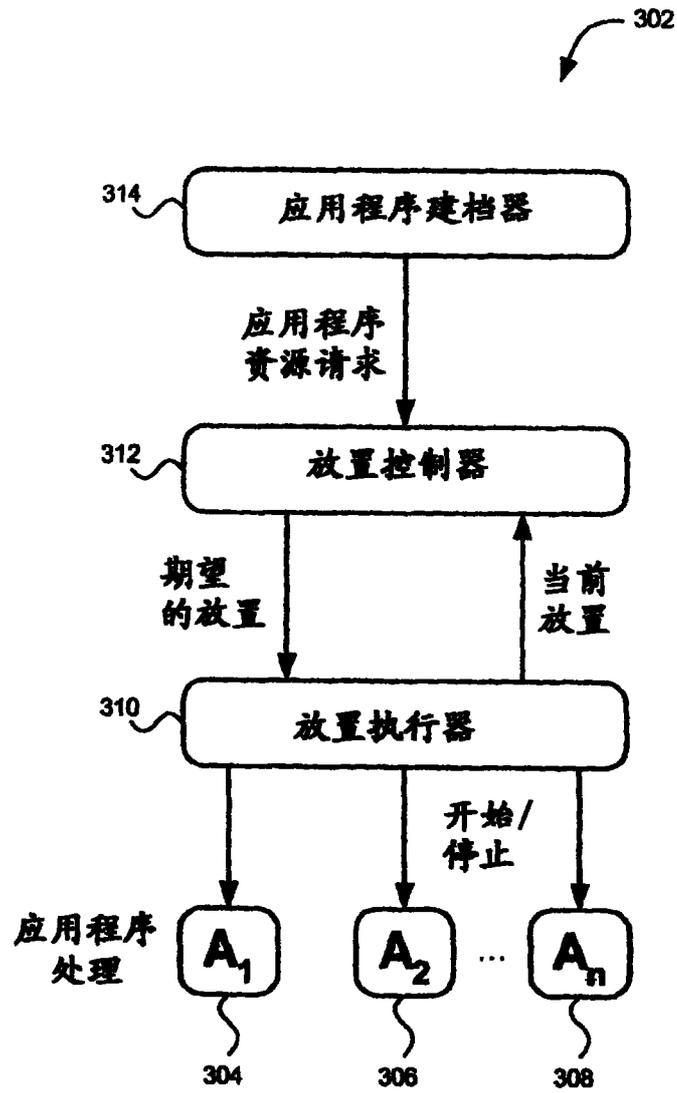


图 3

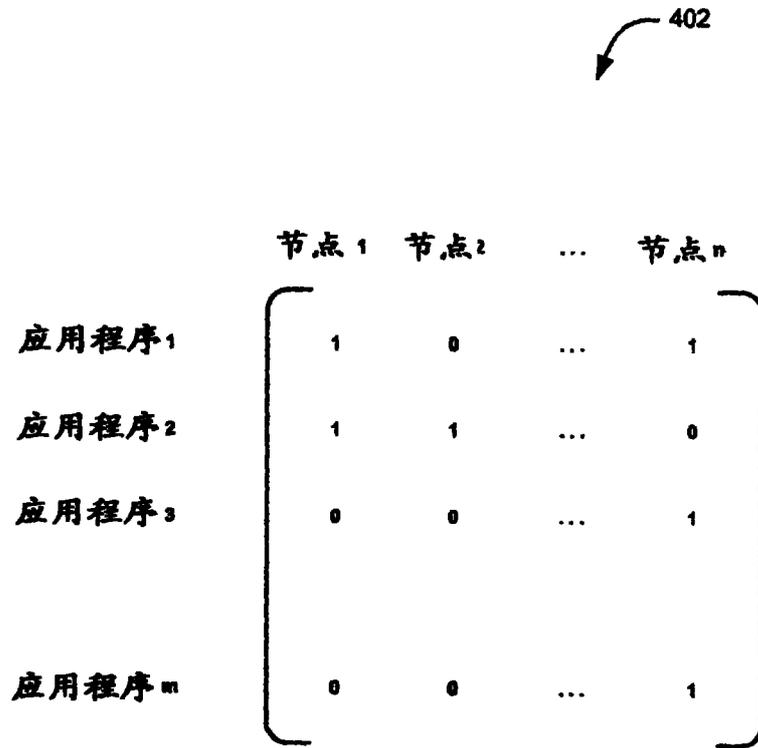


图 4

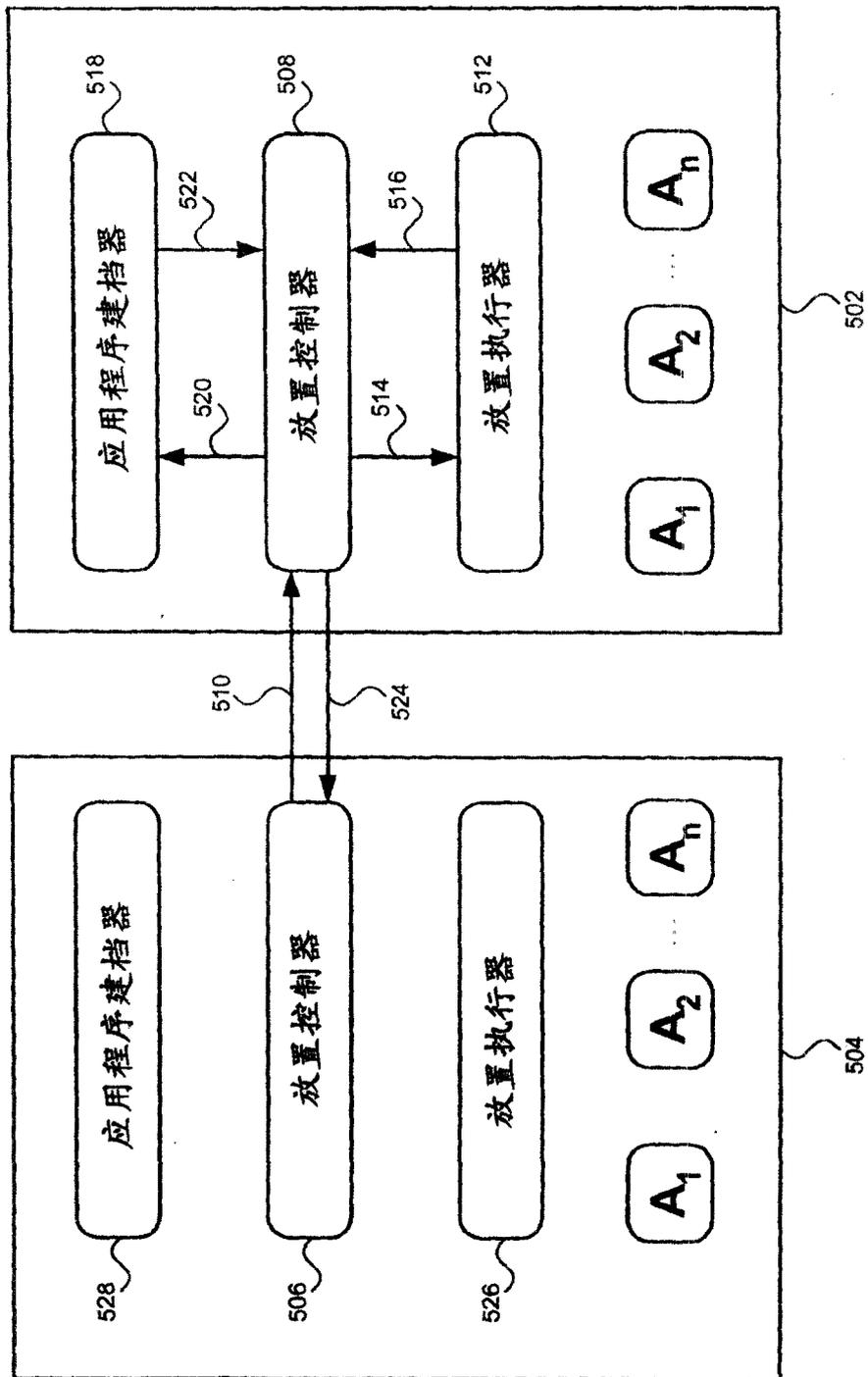


图5

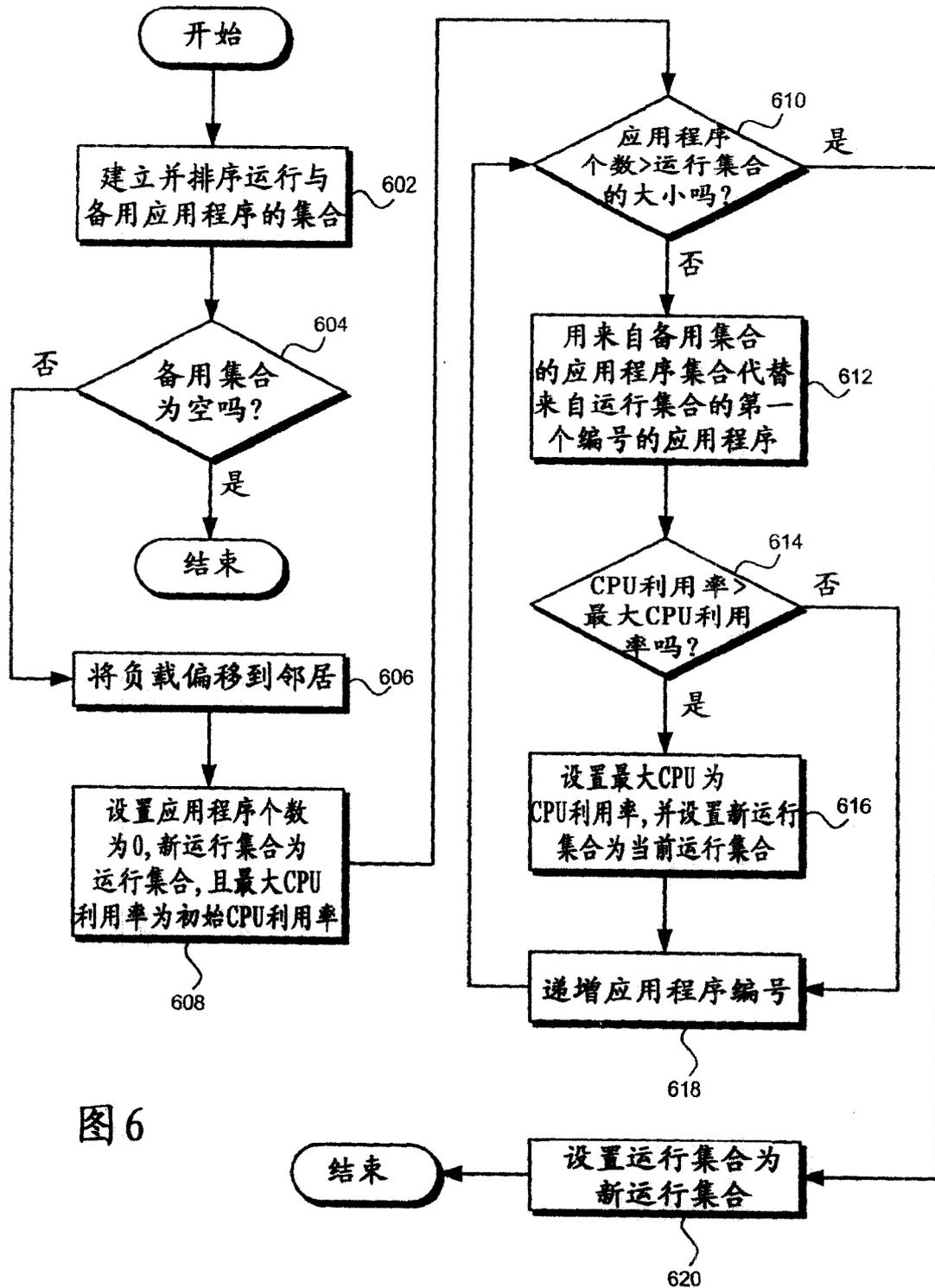
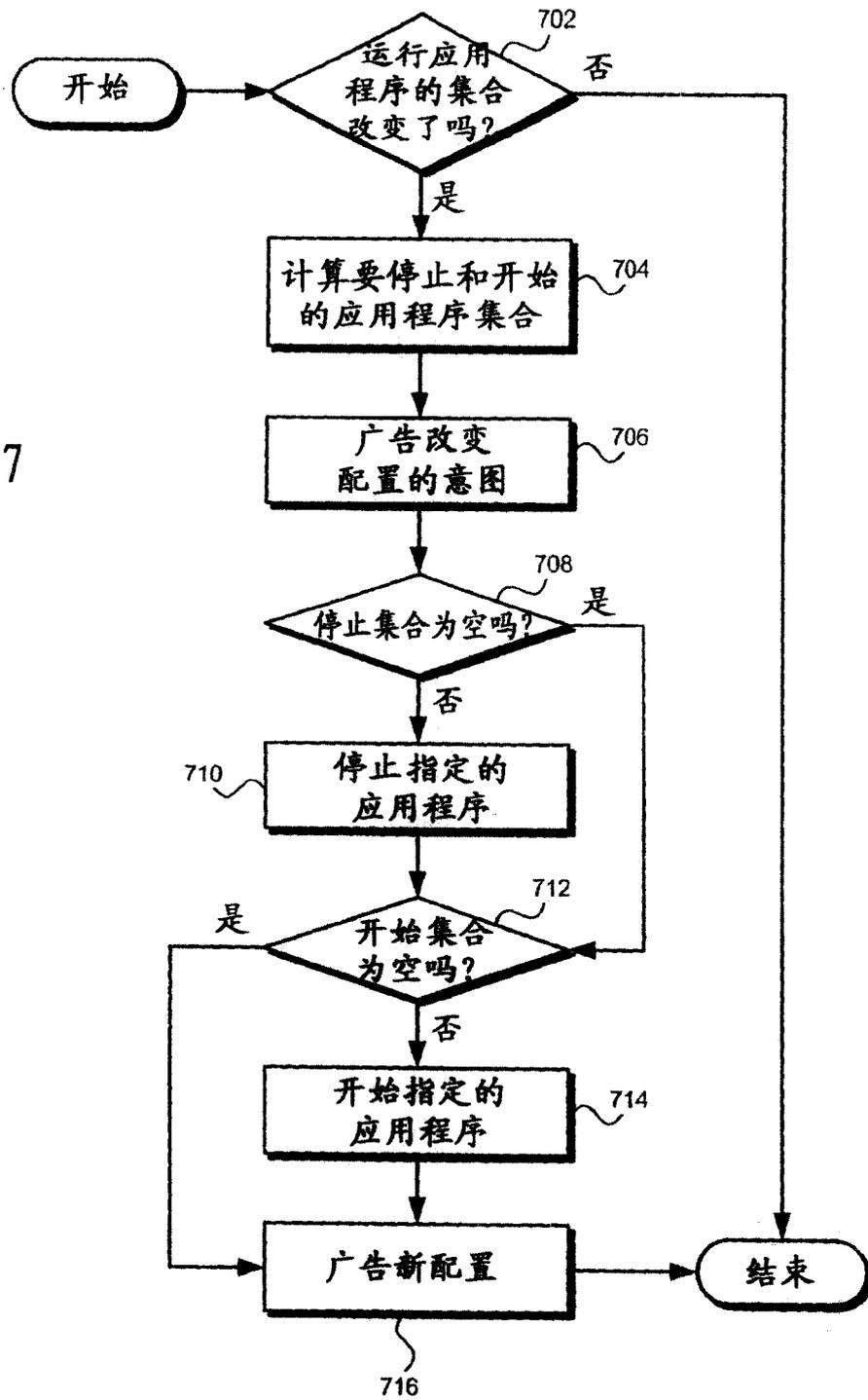


图6

图 7



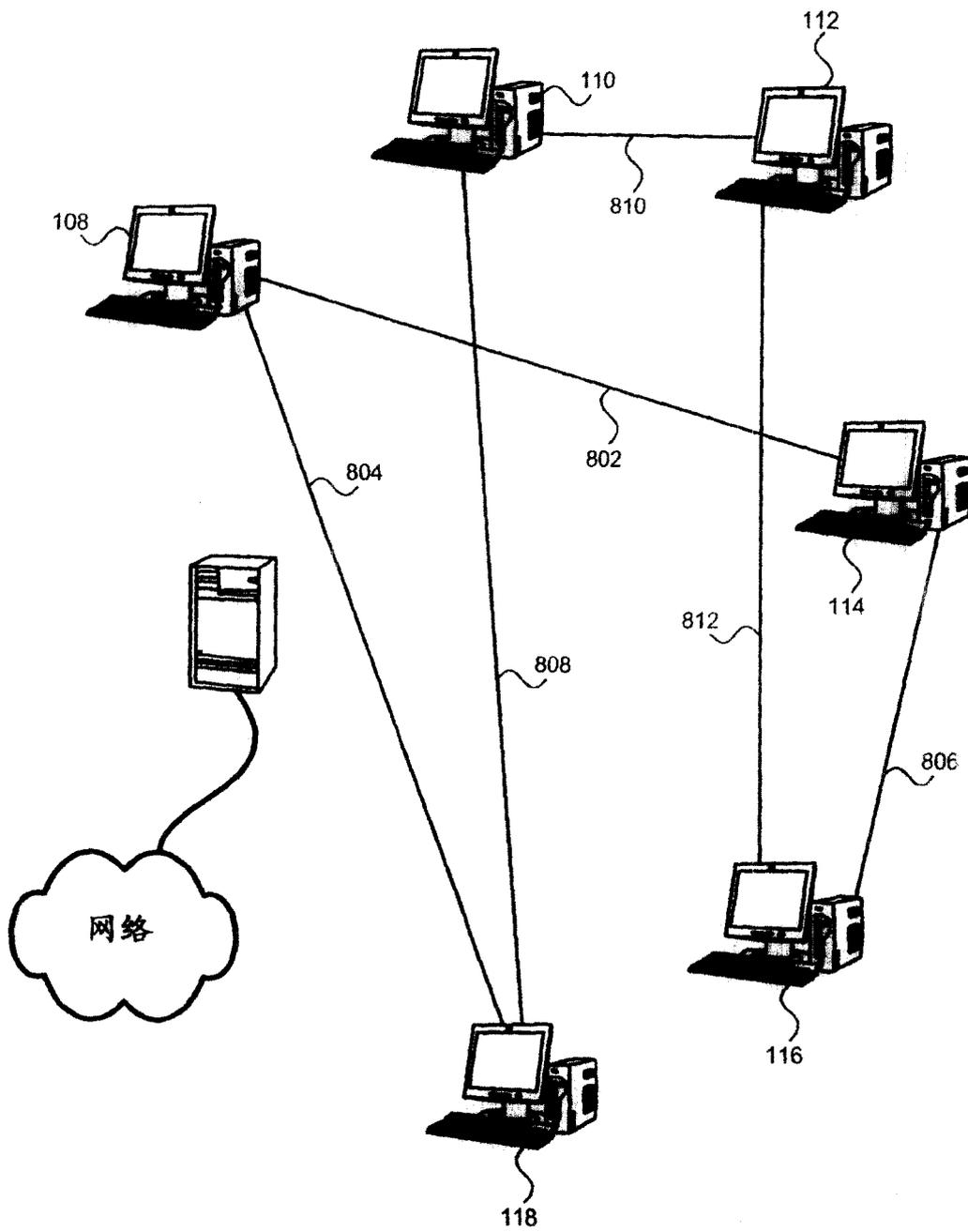


图 8

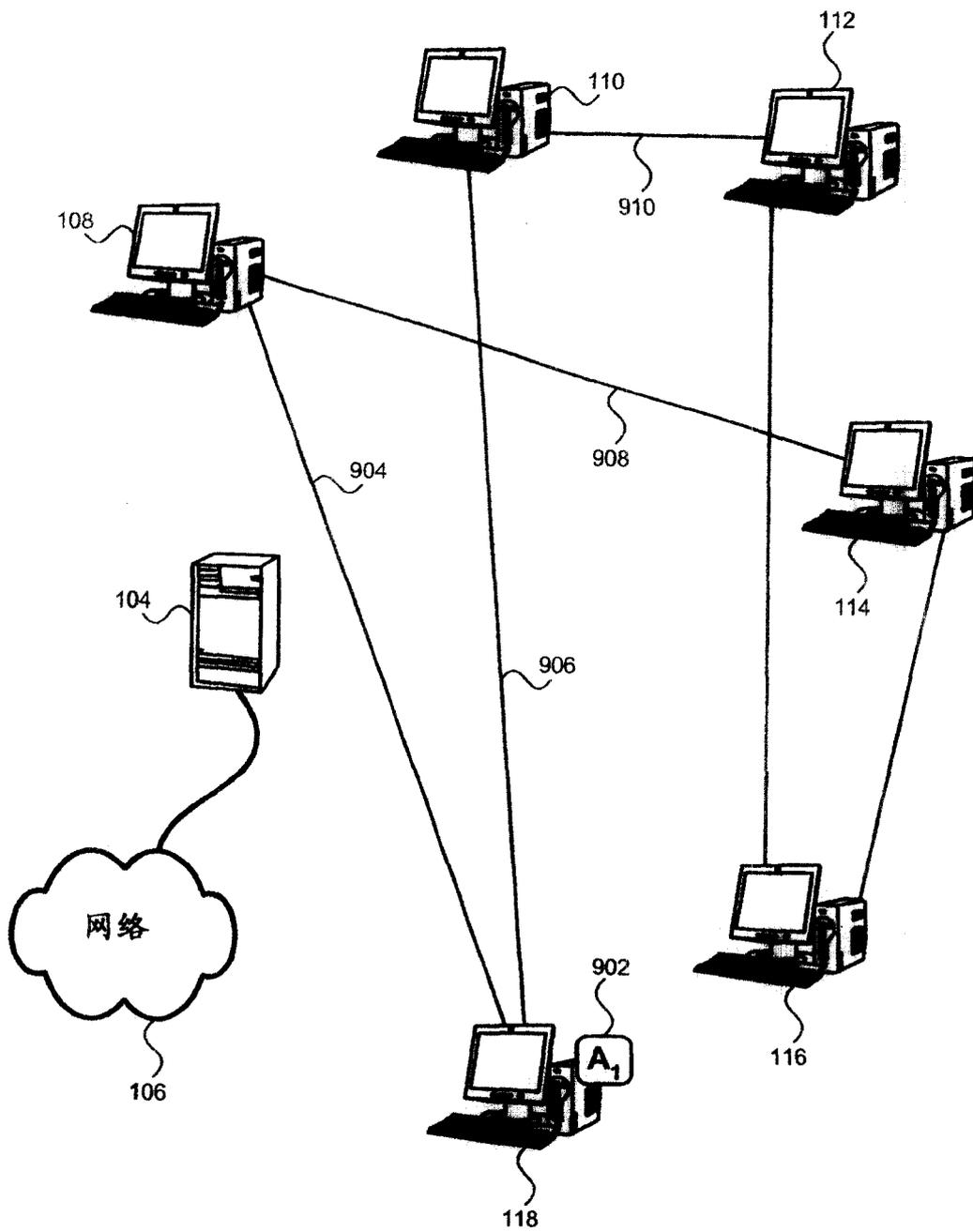


图 9

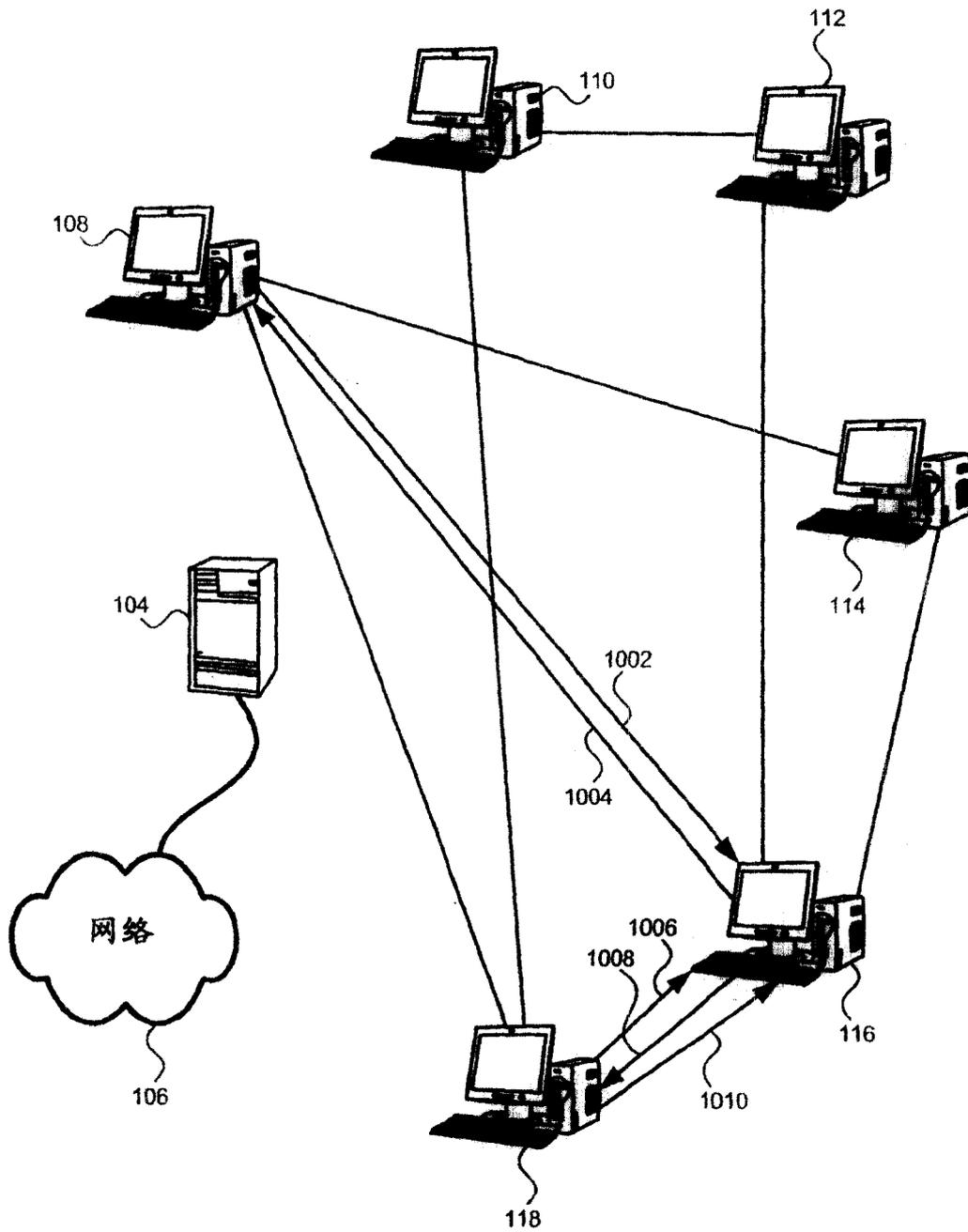


图 10

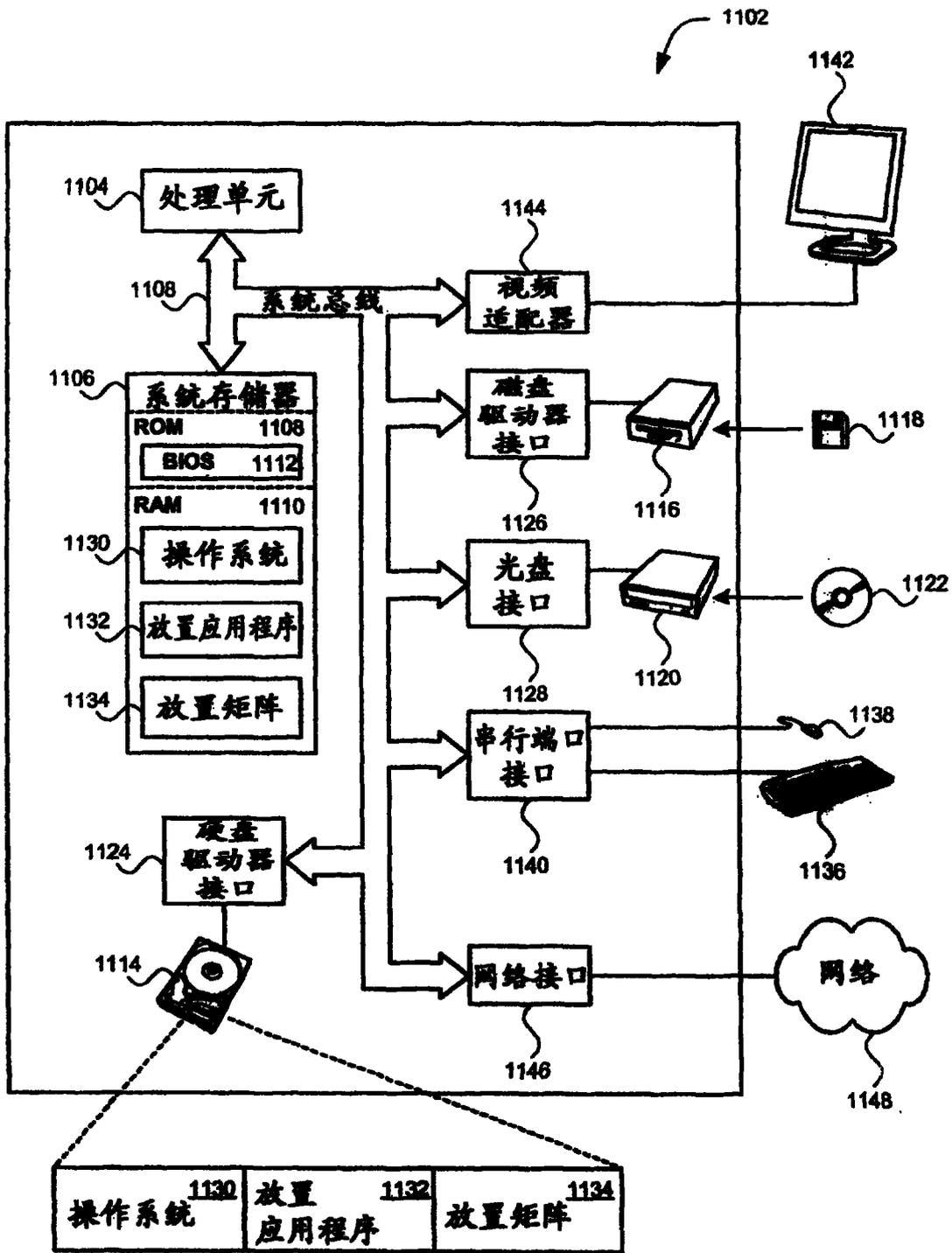


图 11

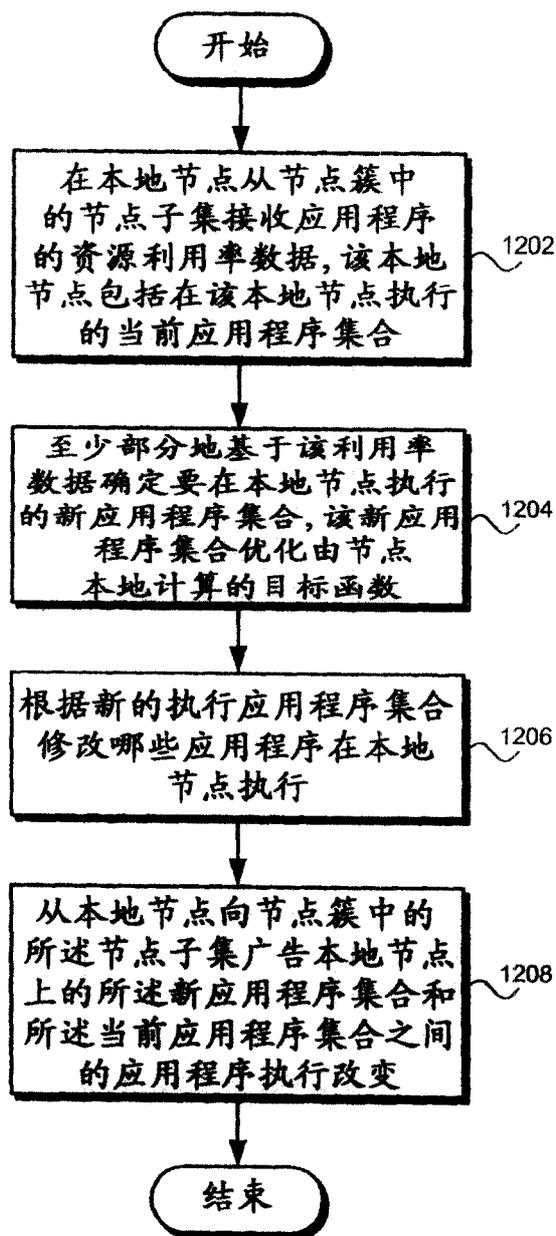


图 12