(19)中华人民共和国国家知识产权局

(12)发明专利

(10)授权公告号 CN 1774699 B
(45)授权公告日 2010.05.26

(21)申请号 200480009868.2
(22)申请日 2004.04.15
(30)优先权数据
10/422,193 2003.04.24 US
(85)PCT申请进入国家阶段日
2005.10.13
(86)PCT申请的申请数据
PCT/GB2004/001624 2004.04.15
(87)PCT申请的公布数据
WO2004/094863 EN 2004.11.04
(73)专利权人 国际商业机器公司
地址 美国纽约阿芒克
(72)发明人 特洛伊·D·阿姆斯特朗
凯尔·A·勒克
(74)专利代理机构 北京市柳沈律师事务所 11105
代理人 郭定辉 黄小临
(51)Int.Cl.
*G06F 9/52*(2006.01)

(56)对比文件
US 5805900 A,1998.09.08, 全文.
US 5774731 A,1998.06.30, 全文.
DOUGLAS E. COMER.Principles, Protocols, and Architecture
Fourth edition
1.2000,215-251.

审查员 赵伟华

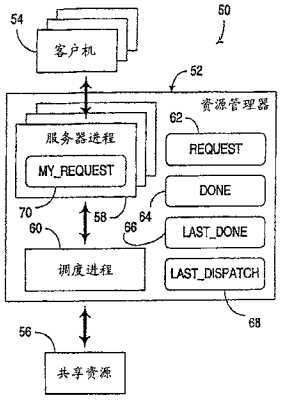
权利要求书 3 页 说明书 10 页 附图 4 页

(54)发明名称

共享资源的并发访问方法和设备

(57)摘要

凭借一组原子操作管理多线程计算机中多个进程对共享资源的访问的设备、程序产品和方法，这组原子操作跟踪接收使用共享资源的请求的顺序和在接收到这样的请求之后完成它们的处理的顺序。请求的调度被有效推迟，直到比最近完成请求早接收的所有未调度请求的处理都已完成。在许多实例中，可以非原子地进行请求的处理，从而减轻了有关共享资源的竞争。此外，可以成批调度多个请求，从而减少与各个调度操作有关的开销。




CN 1774699 B

1. 一种访问计算机中共享资源的方法，该方法包括，在能够异步发出使用共享资源的请求的多个进程当中的第一进程中：

将唯一标识符原子地指定给使用共享资源的当前请求，其中，根据相对于多个进程发出的其它请求接收当前请求的顺序，将唯一标识符指定给当前请求；

利用唯一标识符完成当前请求的处理；

在当前请求的处理完成之后，原子地递增完成请求的计数；

在当前请求的处理完成之后，原子地确定最近完成请求之前发出的每个其它请求的处理是否已经完成；以及

响应最近完成请求之前发出的每个其它请求的处理已经完成的确定，调度处理已经完成的每个未调度请求。

2. 根据权利要求 1 所述的方法，其中，非原子地进行当前请求的处理。

3. 根据权利要求 1 所述的方法，还包括，在第一进程中，在当前请求的处理完成之后，原子地确定当前请求是否是最近完成请求。

4. 根据权利要求 1 所述的方法，还包括，在第一进程中，在当前请求的处理完成之后，原子地确定当前请求是否是最后调度操作之后发出的第一请求，并且响应当前请求是最后调度操作之后发出的第一请求的确定，调度当前请求。

5. 根据权利要求 1 所述的方法，其中，调度处理已经完成的每个未调度请求包括成批地调度多个请求。

6. 根据权利要求 1 所述的方法，其中，原子地将唯一标识符指定给当前请求包括递增 request 计数器，原子地递增完成请求的计数包括递增 done 计数器，并且该方法还包括响应当前请求是最近完成请求的确定，在当前请求的处理完成之后，将 last_done 计数器原子地更新成当前请求的唯一标识符；原子地确定最近完成请求之前发出的每个其它请求的处理是否已经完成包括将 last_done 计数器与 done 计数器进行比较；以及调度处理已经完成的每个未调度请求包括将 last_dispatch 计数器更新成等于 last_done 计数器。

7. 根据权利要求 6 所述的方法，其中，利用单调增加的数字序列更新 request、done、last_done 和 last_dispatch 计数器中的每一个。

8. 根据权利要求 1 所述的方法，还包括：

响应最近完成请求之前发出的每个其它请求的处理还没有完成的确定，推迟处理已经完成的每个未调度请求的调度；以及

在多个进程当中的第二进程中：

完成比当前请求早的较早请求的处理，较早请求具有在当前请求的唯一标识符前面的指定给它的唯一标识符；

在较早请求的处理完成之后，原子地确定最近完成请求之前发出的每个其它请求的处理是否已经完成；以及

响应最近完成请求之前发出的每个其它请求的处理已经完成的确定，调度处理已经完成的每个未调度请求。

9. 根据权利要求 1 所述的方法，其中，完成当前请求的处理包括准备请求。

10. 根据权利要求 1 所述的方法，其中，共享资源包括输入 / 输出适配器中的帧缓冲器的至少一个细片，完成当前请求的处理包括利用细片传送数据，以及调度每个未调度请求

包括通知客户接收传送的数据。

11. 根据权利要求 10 所述的方法，其中，计算机包括逻辑分区计算机，而输入 / 输出适配器是与用于逻辑分区计算机中的分区间通信的虚拟局域网耦合的虚拟输入 / 输出适配器。

12. 根据权利要求 1 所述的方法，其中，共享资源是从由处理器、硬件设备、输入 / 输出设备、存储设备、适配器、控制器、数据库、文件系统、服务和它们的组件组成的组中选择出来的。

13. 一种访问计算机中共享资源的设备，包括 ：

被配置成执行能够异步发出使用共享资源的请求的多个进程的至少一个处理器 ；以及

根据当前请求相对于多个进程发出的其它请求的接收顺序，用于将唯一标识符原子地指定给使用共享资源的当前请求的装置 ；

利用唯一标识符完成当前请求的处理的装置 ；

在当前请求的处理完成之后，原子地递增完成请求的计数的装置 ；

在当前请求的处理完成之后，原子地确定最近完成请求之前发出的每个其它请求的处理是否已经完成的装置 ；以及

响应最近完成请求之前发出的每个其它请求的处理已经完成的确定，调度处理已经完成的每个未调度请求的装置。

14. 根据权利要求 13 所述的设备，其中，程序代码被配置成非原子地完成当前请求的处理。

15. 根据权利要求 13 所述的设备，其中，程序代码被进一步配置成在当前请求的处理完成之后，原子地确定当前请求是否是最近完成请求。

16. 根据权利要求 13 所述的设备，程序代码被进一步配置成在当前请求的处理完成之后，原子地确定当前请求是否是最后调度操作之后发出的第一请求，并且响应当前请求是最后调度操作之后发出的第一请求的确定，调度当前请求。

17. 根据权利要求 13 所述的设备，其中，程序代码被配置成通过成批地调度多个请求，调度处理已经完成的每个未调度请求。

18. 根据权利要求 13 所述的设备，还包括 request 计数器、done 计数器、last_done 计数器和 last_dispatch 计数器，其中，程序代码被配置成通过递增 request 计数器，原子地将唯一标识符指定给当前请求 ；程序代码被配置成通过递增 done 计数器，原子地递增完成请求的计数 ；程序代码被进一步配置成响应当前请求是最近完成请求的确定，在当前请求的处理完成之后，将 last_done 计数器原子地更新成当前请求的唯一标识符 ；程序代码被配置成通过将 last_done 计数器与 done 计数器进行比较，原子地确定最近完成请求之前发出的每个其它请求的处理是否已经完成 ；以及程序代码被配置成通过将 last_dispatch 计数器更新成等于 last_done 计数器，进一步调度处理已经完成的每个未调度请求。

19. 根据权利要求 18 所述的设备，其中，利用单调增加的数字序列更新 request、done、last_done 和 last_dispatch 计数器中的每一个。

20. 根据权利要求 13 所述的设备，其中，程序代码是第一程序代码，其中，第一程序代码被进一步配置成响应最近完成请求之前发出的每个其它请求的处理还没有完成的确定，推迟处理已经完成的每个未调度请求的调度，并且该设备还包括被配置成在第一进程中在

至少一个处理器上执行的第二程序代码，用于完成比当前请求早的较早请求的处理，较早请求具有在当前请求的唯一标识符前面的指定给它的唯一标识符；在较早请求的处理完成之后，原子地确定最近完成请求之前发出的每个其它请求的处理是否已经完成；以及响应最近完成请求之前发出的每个其它请求的处理已经完成的确定，调度处理已经完成的每个未调度请求。

21. 根据权利要求 13 所述的设备，其中，程序代码被配置成通过准备请求，完成当前请求的处理。

22. 根据权利要求 13 所述的设备，其中，共享资源包括输入/输出适配器中的帧缓冲器的至少一个细片，程序代码被配置成通过利用细片传送数据，完成当前请求的处理；以及程序代码被配置成通过通知客户接收传送的数据，调度每个未调度请求。

23. 根据权利要求 22 所述的设备，其中，该设备包括逻辑分区计算机，而输入/输出适配器是与用于逻辑分区计算机中的分区间通信的虚拟局域网耦合的虚拟输入/输出适配器。

24. 根据权利要求 13 所述的设备，其中，共享资源是从由处理器、硬件设备、输入/输出设备、存储设备、适配器、控制器、数据库、文件系统、服务和它们的组件组成的组中选择出来的。

# 共享资源的并发访问方法和设备

## 技术领域

[0001] 本发明涉及计算机和计算机软件，尤其涉及管理计算机中对共享资源的并发访问。

## 背景技术

[0002] 在当今社会里，随着对计算机的依赖性不断增加，计算机技术已向许多前沿前进，以赶上不断增加的需求。重要研究和开发工作的一个具体课题是并行性问题，即，在并行下多任务的性能问题。

[0003] 许多计算机软件和硬件技术已经被开发成有助于提高并行处理。从硬件的观点来看，计算机越来越依赖于多个微处理器来提供不断增加的工作负载容量。并且，已经开发出一些支持并行执行多线程的能力的微处理器，有效地提供了通过多微处理器的使用可达到的许多相同性能增益。从软件的观点来看，已经开发出允许计算机程序以多个线程同时执行，以便基本上可以同时执行多个任务的多线程操作系统和内核。

[0004] 另外，一些计算机实现了逻辑分区的概念，其中，允许单个物理计算机基本上像多个独立“虚拟”计算机（称为逻辑分区）那样操作，以及在各种各样逻辑分区之间分配物理计算机中的各种资源（例如，处理器、存储器、输入/输出设备）。每个逻辑分区执行单独操作系统，并从用户的角度看和从在逻辑分区上执行的应用软件的角度看，都像完全独立的计算机那样操作。

[0005] 虽然凭借同时执行多个任务的能力，并行性有效地提高了系统性能，但是，由于需要使多个并发进程的操作同步，尤其考虑到多个线程能够共享的系统资源，并行性的一种副作用是增加了系统复杂性。能够访问特定资源的独立线程或进程通常意识不到其它线程或进程的活动。这样，存在着一个进程可能以与另一个进程有关的出乎意料的方式访问特定资源，造成不确定结果和潜在系统错误的风险。

[0006] 作为许多例子当中一个，诸如以太网适配器之类的输入/输出（I/O）适配器可能具有能够被在计算机上执行的多个线程或进程共享的数量有限的资源。这种类型的一个资源是帧缓冲器，它用于存储传送到以太网兼容网络和从以太网兼容网络接收的数据的帧。帧缓冲器往往被安排成不同进程依次使用的细片，以及将消耗的每个细片通知I/O适配器。但是，消耗一个细片可能占用相当多时间量，在这个时间量内其它进程可能无法使用其它细片。此外，还存在着两个进程可能试图同时使用同一细片的担忧。

[0007] 为了消除这些担忧，诸如锁或标志之类的串行化机构可以用于每次使共享资源的访问限于一个进程。锁或标志本质上是多线程环境下一个进程或线程可以排他地获得用以访问特定共享实体或资源的“令牌”。在进程或线程可以访问资源之前，它必须首先从系统中获得令牌。如果另一个进程或线程同时处理令牌，不允许前一个进程或线程访问资源，直到其它进程或线程释放令牌为止。这样，对资源的访问被有效地“串行化”，以防止发生不确定操作。

[0008] 每当通过串行化机构管理共享资源时，由于需要访问锁定资源的任何线程或进程

在着手访问资源之前，往往必须等到对那个资源的锁定被释放，因此并发机会往往会变小。这样，人们强烈希望使锁放置共享资源上的频率最小和持续时间最短。

[0009] 对于如上所述的 I/O 适配器例子，保证无冲突访问帧缓冲器的最直截了当的解决方案是仅仅围绕着帧缓冲器的任何使用来放置锁。但是，这样做的话，由于相当大的工作量可能需要一个进程来完成，以准备、翻译或其他方式处理存储在帧缓冲器中的数据，因此并行机会受到严重限制。因此，从并行性的观点来看，在出现所有这些活动的同时把其它进程对帧缓冲器的访问锁在外面是适得其反的。

[0010] 在一些情况下可以在获得锁在共享资源上的锁之前，进行与利用共享资源有关的一些在前工作，从而缩短共享资源可能需要锁在其它进程上的时间量。但是，人们已经发现以这种方式引导的传统尝试要求重复检查资源可用性和进行更复杂的在前工作。

[0011] 因此，实际需要的是通过在多线程计算机中执行的多个线程或进程协调对共享资源的访问的改进方式。

## 发明内容

[0012] 通过提供凭借一组原子操作管理多线程计算机中多个进程对共享资源的访问的设备、程序产品和方法，本发明解决了与现有技术有关的这些和其它问题，这组原子操作跟踪接收使用共享资源的请求的顺序和在接收到这样的请求之后完成它们的处理的顺序。请求的调度被有效推迟，直到比最近完成请求早接收的所有未调度请求的处理都已完成。

[0013] 具体地说，按照本发明的一个方面，共享资源可以被能够异步发出使用共享资源的请求的多个进程访问。在第一进程中，唯一标识符被原子地指定给使用共享资源的当前请求，其中，根据相对于多个进程发出的其它请求接收当前请求的顺序，将唯一标识符指定给当前请求。此后，利用唯一标识符完成当前请求的处理，并在这样的处理完成之后，原子地递增完成请求的计数，以及对最近完成的请求之前发出的每个其它请求的处理是否已完成作出原子确定。如果是的话，接着调度处理已经完成的每个未调度请求。

[0014] 在按照本发明的一些实施例中，可以非原子地完成请求的处理，从而限制与处理请求结合在一起进行的原子操作的持续时间，从而减轻对共享资源的竞争。此外，在一些实施例中，未调度请求的调度可能导致多个请求被成批调度，从而减少单独调度的开销。

## 附图说明

[0015] 现在参照附图，仅仅通过举例的方式对本发明加以描述，在附图中：

[0016] 图 1 是以按照本发明的方式支持多个客户对共享资源的并发访问的通用计算机结构的方块图；

[0017] 图 2 是图解图 1 的计算机结构中的服务器进程执行来处理客户请求的进程客户请求例程的程序流的流程图；

[0018] 图 3 是按照本发明实现共享资源的并发访问的逻辑分区计算机中的主要硬件组件的方块图；

[0019] 图 4 是图 3 的计算机中涉及管理计算机中对共享资源的并发访问的主要组件的方块图；以及

[0020] 图 5 是图解在图 4 中引用的服务器进程执行的进程客户请求例程的程序流的流程

图。

## 具体实施方式

[0021] 下文讨论的实施例跟踪接收使用共享资源的、由多个客户发出的请求的顺序，以及与用以使用共享资源的调度请求结合起来完成这类请求的处理的顺序。这些请求能够异步发出，而且能够无序完成。结果是，随着请求被接收，根据接收它们的顺序，将唯一标识符指定给它们。此外，跟踪已完成请求的数量，并与管理每个请求结合在一起作出确定，以确定在最近完成的请求之前发出的每个其它请求的处理是否已完成。一旦确定这个条件得到满足，就接着调度处理已完成的每个未调度请求。

[0022] 正如下面更明显看到的那样，接收请求的顺序和完成这类请求的顺序的跟踪可以通过与共享资源有关和能够通过诸如标志之类的排他锁定机构原子访问的一系列计数器的使用作出。request（请求）计数器可以用于随着请求被接收，将唯一标识符指定给它们。done（完成）计数器可以用于跟踪完成请求的数量（即，诸如准备和提交之类的处理已完成的请求）。last_done 计数器可以用于跟踪最近完成的请求，而 last_dispatch 计数器可以用于跟踪最近调度请求。

[0023] 在按照本发明的许多实施例中，只需对上述计数器的访问是原子的。与管理请求有关的其它处理可能往往是非原子管理的，从而限制了与管理请求结合在一起进行的原子操作的持续时间，并且减少其他可能妨碍对共享资源的并发访问的竞争。

[0024] 另外，按照本发明的实施例支持提供请求的成批调度的能力。按照本发明，请求的调度可以以不同方式发生。例如，在共享资源是诸如网络适配器之类的输入 / 输出适配器的帧缓冲器中的细片的情况下，每当通过输入 / 输出适配器为客户接收信息包时，可以随着对那个客户的中断的上升实现调度操作。在共享资源是诸如 SCSI 网络之类的网络中的大容量存储设备的情况下，调度操作可以是将 I/O 请求已完成通知操作系统。

[0025] 请求的性质通常随所讨论对共享资源的特定类型而改变。例如，对于数据库资源，请求可以代表数据库交易，而对于文件管理系统资源，请求可以代表文件访问请求。对于存储设备，请求可以代表输入 / 输出 (I/O) 请求，而对于处理器资源，请求可以代表将处理器分配给逻辑分区的请求。其它共享资源和请求实现对从本公开受益的本领域普通技术人员来说是显而易见的。

[0026] 现在转到附图，在附图中，相同的号码贯穿几个图形表示相同的部分，图 1 图解了可以用于以按照本发明的方式支持对共享资源的同时和异步访问的通用计算机结构 50 的示范性配置。具体地说，所示的是合并了管理多个客户 54 对通用共享资源 56 的访问的资源管理器 52 的结构 50。

[0027] 可以认为共享资源 56 实际代表能够被计算机访问的任何类型的计算机资源，例如，诸如以太网适配器、SCSI 适配器和其它网络适配器之类的各种类型的输入 / 输出和其它硬件资源；存储器适配器和控制器；工作站适配器和控制器；数据库控制器；处理器；其它输入 / 输出适配器和控制器等。此外，可以认为共享资源 56 包括各种类型的软件资源，例如，内核或操作系统服务、文件管理系统、数据库等。共享资源还可以包括上述类型的资源的子组件，例如，驻留在输入 / 输出适配器中的缓冲器或其它存储器资源。

[0028] 可以认为每一个客户 54 实际上代表能够以与任何其它客户有关的异步方式访问

资源 56 的任何程序代码，或甚至是外部可编程设备。在所图解的实施例中，通常，利用以驻留在计算机中的多个并行进程或线程之一执行的计算机上的程序代码，例如，利用包括单线程和 / 或多线程处理器的一个或多个处理器、支持多线程的基于软件或操作系统、逻辑分区和 / 或使多个执行实例能够并行地异步访问共享资源的其它技术，实现每个客户 54。

[0029] 为了管理对共享资源 56 的访问，资源管理器 52 包括对于各种客户 54 生成的资源访问请求，起单独服务器作用的多个服务器进程 58。可以将每个服务器进程 58 静态或动态地指定给给定客户 54，并且，应该认识到，服务器进程和客户之间的映射不需要 1:1，可能不时地改变。

[0030] 资源管理器 52 还包括与服务器进程交互以协调对共享资源 56 的访问的调度进程 60。正如下面更明显看到的那样，调度进程 60 可以实现成批调度，以便每次调度多个访问请求。此外，调度进程 60 还可以管理数据从共享资源 56 的返回，并可以与服务器进程 58 交互，或直接与每个客户交互，以将输入数据路由给适当客户。但是，在其它实施例中，可以与调度进程 60 分开地管理返回数据。并且，应该认识到，资源管理器 52 可以管理多个共享资源，或多个资源管理器可以用于管理多个共享资源。

[0031] 为了以按照本发明的方式支持对共享资源的并发访问，资源管理器 52 包括 4 个原子计数器 62、64、66 和 68。计数器 62 在这里被称为 request 计数器，用于供应要指定给客户 54 之一生成的下一个访问请求的标识符。对于每个生成的请求，使 request 计数器 62 递增到下一个标识符，在服务器进程中将那个标识符存储在 my_request 变量 70 中，以便唯一地标识那个请求。在各种实施例中，可以事先或事后递增 request 计数器 62，以便计数器中的当前值可以代表最后接收的请求，或当接收到下一个请求时指定给它的值。

[0032] 计数器 64 在这里被称为 done 计数器，用于跟踪已经完成的访问请求的数量。计数器 66 在这里被称为 last_done 计数器，用于保存已经完成的最后（或最近）访问请求的标识符。在允许请求无序完成，即，可以允许较晚请求在较早请求之前完成的情况下，通常需要这两个计数器 64 和 66。

[0033] 计数器 68 被称为 last_dispach 计数器，用于存储已经被调度进程 60 调度给共享资源的最后（或最近）访问请求的标识符，并且通常在通知调度进程的最后时刻被设置成 last_done 计数器 66 的值。应该认识到，选择用于标识计数器 62、64、66 和 68 的特定名称仅仅是为了方便，不对本发明的范围构成限制。

[0034] 在所图解的实施例中，每个计数器 62、64、66 和 68 被初始化成 0，并且计数器 62 和 64 按 1 递增，导致每个计数器总是被总是增加的整数更新。但是，应该认识到，其它单调数字序列或其它标识符也可以用于唯一地标识按照本发明的请求。此外，虽然可以使用使计数器能够在处理了一定数量的请求之后回绕 (roll over) 的计数器量值，但已经证明，利用使计数器不能在可行时间间隔（例如，8 字节计数器）内回绕的计数器量值可大大简化处理。在系统启动期间重新初始化每个计数器，以便实际上，在计算机结构 50 的正常操作过程中从来没有计数器回绕也是人们所希望的。

[0035] 现在转到图 2，图 2 更详细图解了可以用于以按照本发明的方式处理客户请求的示范性进程客户请求例程。服务器进程 58 响应每个请求执行例程 72，这样，当多个客户同时提交请求时，例程 72 的多个实例可以在任何给定时刻并行地执行。

[0036] 例程 72 从获取计数器锁，即，获取与计数器 62-68 有关的锁或标志的方块 74 开

始。在所图解的实施例中，这样的标志可以与所有计数器 62-68 有关，或者，在可替代实施例中，可以为每个这样的计数器提供单独的锁，或可以将给定锁指定给一小组计数器。

[0037] 与锁被多个计数器共享，还是只指定给一个或一小组计数器无关，一旦在方块 74 中获得锁，控制就转到方块 76，以递增 request 计数器 62。然后，方块 78 将与请求有关的 my_request 变量 70 设置成 request 计数器 62 的当前值。此后，方块 80 释放计数器锁。结果是，方块 74-80 有效地实现了对 request 计数器的原子递增操作，还根据 request 计数器的递增后值将唯一请求标识符指定给请求。在其它实施例中，可以使用 request 计数器的递增前值。此外，只要在锁内获得 request 计数器的有效副本，可替代地，可以在锁外进行方块 78 中对 my_request 变量 70 的值指定。但是，与使用的方式无关，根据相对于多个客户发出的其它请求接收请求的顺序，将唯一标识符有效地指定给请求。

[0038] 接着，方块 82 代表客户，通过准备和提供请求完成请求的处理。但是，请注意，在锁或标志状态下不执行方块 82，这样，方块 82 不涉及由于访问共享资源的冲突而可能停止当前进程或另一个进程的任何竞争问题。与准备和提交请求有关的步骤通常随特定请求类型和被访问的资源的类型而改变。例如，为了将请求提交给输入 / 输出适配器，在方块 82 中可以进行诸如通过直接存储器存取 (DMA) 操作为传送准备网络帧之类的各种步骤。按照本发明，在方块 82 中也可以进行与准备请求、提交请求和 / 或其他完成请求的处理（例如，存储器分配、复制或传送数据、获取合并资源）有关的其它操作。

[0039] 接着，方块 84 再次获取计数器锁，如有必要，等待要获取的计数器锁。一旦获得计数器锁，控制转到方块 86 来递增 done 计数器 64，从而确立当前请求的处理已完成。

[0040] 然后，方块 88 确定当前请求的标识符是否大于存储在 last_done 计数器 66 中的值。如果是，知道当前请求是已经完成的最近或最后请求。因此，控制转到方块 90，将 last_done 计数器 66 的值设置成等于当前请求标识符。然后，控制转到方块 92。此外，返回到方块 88，如果当前请求标识符没有大于存储在 last_done 计数器 66 中的值，则方块 88 使控制直接转到方块 92，从而绕过方块 90。

[0041] 方块 92 通常通过确定存储在 done 计数器 64 中的值是否等于存储在 last_done 计数器 66 中的值，确定在最近完成请求之前发出的每个其它请求的处理是否都已完成。当这个条件得到满足时，知道时间上比最近完成请求（标识在计数器 66 中）早的每个请求都已完成，因为 done 计数器 64 中的较小值表示至少还有一个较早请求没有完成。

[0042] 如果方块 92 中的条件得到满足，则使控制转到方块 94，将 last_dispatch 计数器 68 中的值设置成等于存储在 last_done 计数器 66 中的那个值，表示直到和包括标识在 last_done 计数器中的请求的每个请求都已完成。然后，控制转到方块 96，以通知调度进程调度所有还未调度请求（通常是执行方块 94 之前 last_done 计数器 66 中的值与存储在 last_dispatch 计数器 68 中的值之间的那些）。然后，控制转到方块 98，以释放在方块 84 中获得的计数器锁，从而完成例程 72。应该认识到，在按照本发明的其它实施例中，方块 94 可以在方块 96 之后执行。

[0043] 返回到方块 92，如果并非所有较早请求都已完成，则方块 92 使控制转到方块 100，通常通过确定当前标识符是否是大于存储在 last_dispatch 计数器 68 中的值的一个，确定当前请求的标识符是否是在最后调度请求之后由调度进程启用的第一请求。

[0044] 如果是，则控制转到方块 102，以将计数器 68 的值设置成等于当前请求标识符。然

后，控制转到方块 96，通知调度进程调度当前请求。否则，如果在方块中阐述的条件未得到满足，则返回方块 100，不要求通过例程 72 的这个实例进行进一步的工作，因为以后进行的其它工作将使请求最终得到调度。于是，方块 100 使控制直接转到方块 98，释放计数器锁，并且终止该例程。

[0045] 应该认识到，在一些实施例中可以省略在方块 100 中进行的判定和因而发生的经过方块 102 的流程。就为成批调度之后启用的第一请求缩短等待时间而言，上述功能提供了好处。在一些实施例中，第一请求的等待时间可能不值得严重担忧，因此在这样的实施例中可以省略上述功能。在这种情况下，如果这里阐述的条件未得到满足，则方块 92 可以使控制直接转到方块 98。

[0046] 就管理多个客户对共享资源的访问而言，上面的请求管理技术提供了许多独特优点。从生成请求的角度看，无需要求同步或串行化，就可以生成和保留多个访问请求。例如，在将请求内置在被划分成数量有限的块或细片的缓冲器中的情况下，未必使这样细片的完全消耗同步，就可以将客户同时指定给特定细片。另外，可以成批而不是一个一个地将多个请求调度到共享资源，从而省去了与每次调度有关的许多开销（例如，与通过中断通知客户有关的开销）。并且减少了对用于保护和管理对共享资源的访问的竞争。在上述计数器的相对短更新期间，锁也许只代表特定请求保持着，从而能够获得与在不需要任何同步机构进行请求的调度之前，建立和准备与请求有关的许多开销。

[0047] 如上所述，上述管理共享资源的并发访问的技术可以用在各种应用中。下面结合图 3-5 更详细描述的一种具体应用是在诸如图 3 的计算机 10 那样的逻辑分区计算机中管理对物理或虚拟以太网适配器的访问的应用。尤其，众所周知，诸如以太网适配器和令牌环适配器之类的输入 / 输出适配器只具有数量有限的资源，在这种情况下，它们是帧缓冲器中的细片。通常允许多个客户消耗细片。并且，消耗一个细片可能在相当多时间量内占用该细片，在这个时间量内其它客户将无法使用该细片。不影响其它客户地允许多个客户同时消耗这样的细片将使计算机系统中的并行执行更加有效，因此，使系统性能得到提高。像如下所述那样的计算机 10 以按照本发明的方式支持细片的这种并发消耗。

[0048] 计算机 10 一般代表例如诸如网络服务器之类的任意个多用户计算机、中档计算机、大型计算机（例如，IBM eServer 计算机）等。但是，应该认识到，本发明可以在其它计算机和数据处理系统中，例如，在诸如工作站、台式计算机和便携式计算机等的单用户计算机中，或在其它可编程电子设备（例如，装有内置控制器等）中实现。另外，本发明也可以与非逻辑分区多线程计算机结合在一起使用。

[0049] 如图 3 最佳地示出的那样，计算机 10 一般包括通过总线 6 与存储器 14 耦合的一个或多个处理器 12。每个处理器 12 可以作为单线程处理器来实现，或者，像借助于被显示成装有多个硬件线程 18 的处理器 12a 那样，作为多线程处理器来实现。在一般情况下，多线程处理器 12a 中的每个硬件线程 18 被驻留在计算机中的软件当作独立处理器来对待。关于这一点，为了本公开的目的，将认为单线程处理器装有单个硬件线程，即，单个独立执行单元。但是，应该认识到，基于软件的多线程或多任务可以与单线程处理器和多线程处理器两者结合在一起使用，以便进一步支持计算机中多个任务的并行性能。

[0050] 另外，还如图 3 所图解的那样，处理器 12 的一个或多个（例如，处理器 12b）可以作为服务处理器来实现，服务处理器用于运行专门固件代码以管理系统初始程序装载（IPL），

并监视、诊断和配置系统硬件。一般说来，计算机 10 包括一个服务处理器和用于执行驻留在计算机中的操作系统和应用程序的多个系统处理器，但本发明不局限于这种特定实现。在一些实现中，服务处理器可以以除了通过总线 16 之外的其它方式与计算机中的各种其它硬件组件耦合。

[0051] 存储器 14 可以包括一个或多个级别的存储设备，例如，基于 DRAM 的主存储器，以及一个或多个级别的数据、指令和 / 或组合高速缓存，正如在现有技术中众所周知的那样，某些高速缓存为单独处理器服务或为多个处理器服务。并且，存储器 14 通过总线 20 与许多类型的外部设备，例如，一个或多个网络适配器 22（使计算机与网络 24 对接）、一个或多个存储器控制器 26（使计算机与一个或多个存储设备 28 对接）和一个或多个工作站控制器 30（通过多个工作站适配器与一个或多个终端或工作站对接）耦合。

[0052] 图 3 还更详细地图解了用在实现计算机 10 上包括通过分区管理器或管理程序 36 管理的多个逻辑分区 34 的逻辑分区计算环境中的基本软件组件和资源。正如在现有技术中众所周知的那样，可以支持任何数量的逻辑分区，随着分区被加入计算机中或从计算机中去除分区，任何时候驻留在计算机中的逻辑分区的数量都可以动态地变化。

[0053] 在所图解的基于 IBM eServer 的实现中，分区管理器 36 由两层程序代码组成。第一层在这里被称为不可调度部分 38，在计算机 10 的固件或许可内部代码 (LIC) 内实现，用于在将较高层（例如，操作系统）与硬件访问的细节隔离的同时，向各种硬件组件提供低级接口。固件也可以与诸如服务处理器 12b 之类的服务处理器通信。不可调度部分 38 为计算机 10 提供许多低级分区管理功能，例如，页表管理等。不可调度部分 38 也没有任务的概念，主要可通过函数调用从软件的较高层访问。

[0054] 分区管理器 36 中程序代码的第二层在这里被称为可调度部分 40。与没有任务的概念、借助于重定位逃逸并可通过函数调用从软件的较高层访问的不可调度部分 38 相反，可调度部分 40 具有任何的概念并借助于重定位继续运行。除了用户无法觉察之外，可调度部分通常以与分区几乎相同的方式执行。可调度部分一般管理较高级分区管理操作，譬如，创建和删除分区。并发 I/O 维护，将处理器、存储器和其它硬件资源分配给各种分区 34 等。

[0055] 通常静态地和 / 或动态地将计算机 10 中的一部分可用资源分配给每个逻辑分区 34。例如，可以将一个或多个处理器 12 和 / 或一个或多个硬件线程 18，以及一部分可用存储空间分配给每个逻辑分区。逻辑分区可以共享诸如处理器之类的特定硬件资源，以便多于一个逻辑分区使用一个给定处理器。在可替代实施例中，每次可以将硬件资源只分配给一个逻辑分区。

[0056] 另外的资源（例如，大容量存储器、备份存储器、用户输入、网络连接和用于它们的 I/O 适配器）通常以在现有技术中众所周知的方式分配给一个或多个逻辑分区。资源可以以许多种方式（例如，一条总线接一条总线地或一个资源接一个资源地）用同一总线上共享资源的多个逻辑分区分配。甚至可以每次将一些资源分配给多个逻辑分区。

[0057] 每个逻辑分区 34 以与未分区计算机的操作系统相同的方式使用控制逻辑分区的基本操作的操作系统 42。例如，可以利用可从美国国际商用机器公司 (IBM) 购买到的 OS/400 操作系统实现每个操作系统 42。

[0058] 每个逻辑分区 34 在分离或独立存储空间中运行，因此，从在每个这样的逻辑分区中执行的每个用户应用程序（用户应用程序）44 的角度看，每个逻辑分区起几乎与独立未

分区计算机相同的作用。这样，用户应用程序通常不需要用在分区环境中的任何特殊配置。

[0059] 在逻辑分区 34 的性质像单独虚拟计算机那样的情况下，最好支持分区间通信，以便允许逻辑分区相互通信，仿佛逻辑分区处在单独物理机器上似的。这样，在一些实施例中，最好支持不可调度部分 38 中的局域网 (LAN)46，以便允许逻辑分区 34 通过诸如以太网协议之类的连网协议相互通信。按照本发明，也可以支持支持分区间通信的其它方式。

[0060] 应该认识到，按照本发明，可以使用其它逻辑分区环境。例如，不是使用与任何分区 34 分离的可调度部分 40，而是在可替代实施例中，可以将可调度部分的功能合并到一个或多个逻辑分区中。

[0061] 一般说来，加以执行以实现本发明的实施例的例程，无论作为操作系统的一部分来实现，还是作为特定应用程序、组件、程序、对象、模块或指令序列来实现，甚至作为它们的一个小组来实现，在这里都被称为“计算机程序代码”，或简称为“程序代码”。程序代码通常包括在各个时刻驻留在计算机中的各个存储器和存储设备中，并且当由计算机中的一个或多个处理器读取和执行时，使那个计算机完成执行实施本发明的各个方面的步骤或要素所需的步骤的一条或多条指令。此外，虽然在完全功能化计算机和计算机系统的背景下对本发明作了描述和在下文中将对本发明加以描述，但本领域的普通技术人员应该认识到，本发明的各种实施例能够作为程序产品以各种形式得到分发，并使本发明与实际用于实现分配的信号承载媒体的具体类型无关地得到平等应用。信号承载媒体的例子包括诸如易失性和非易失性存储设备、软盘和其它可换盘、硬盘驱动器、磁带、光盘（例如，CD-ROM（只读光盘驱动器）和 DVD（数字多功能盘）等）等的可记录型媒体和诸如数字和模拟通信链路之类的传输型媒体，但不局限于这些。

[0062] 另外，下文所述的各种程序代码可以根据在本发明的特定实施例中实现它的应用程序或软件组件来识别。但是，应该认识到，使用如下的任何具体程序术语仅仅是为了方便，因此，本发明不应该局限于唯一地应用在这样术语所标识和 / 或暗示的任何特定应用程序中。并且，在给定可以将计算机程序组织成例程、过程、方法、模块、对象等的通常数量无限的方式，以及在驻留在典型计算机内的各种软件层（例如，操作系统、库、API、应用程序、小应用程序等）内可以分配程序功能的各种方式的情况下，应该认识到，本发明不局限于这里所述的程序功能的特定组织和分配。

[0063] 本领域的普通技术人员将认识到，图解在图 3 中的示范性环境无意对本发明加以限制。的确，本领域的普通技术人员将认识到，可以使用其它可替代硬件和 / 或软件环境，而不偏离本发明的范围。具体地说，本发明不局限于用在逻辑分区环境中，而是可用在支持对共享资源的并发和异步访问的众多其它环境中。

[0064] 现在转到图 4，图 4 图解了与实现对具体类型的共享资源（这里，虚拟输入 / 输出适配器中的帧缓冲器）的并发访问有关的主要软件组件。在这种实现中，为了通过虚拟局域网 (VLAN) 支持分区 34 之间的分区间通信，多个客户 120 被布置在各种分区 34 中。在所图解的实施例中，每个客户可以被实现成，例如，驻留在分区中的设备驱动器。

[0065] 布置在分区管理器 36 内的是为了 VLAN 上的通信，能够被客户 120 访问的虚拟输入 / 输出适配器 (IOA)122。在这样的配置下，每当一个分区中的客户需要将数据传送到另一个分区中的客户时，以将数据转发给非虚拟 IOA 几乎相同的方式，将数据从发送客户转发给分区管理器 34 中分配给发送客户的虚拟 IOA 122。数据被虚拟 IOA 格式化成传统网络

信息包（例如，以太网兼容信息包），传送给不同分区 34 中分配给接收客户 120 的另一个虚拟 IOA122。这种数据通信出现在软件中，而不是出现在像用在物理 LAN 中那样的物理电缆上，但是，另一方面，使用的通信协议是相同的。然后，分配给接收客户的虚拟 IOA 122 以与非虚拟 IOA 大体相同的方式将接收数据传递给接收客户。这样，从每个客户 120 的角度看，一个客户发送数据和另一个客户接收数据的方式是相同的，仿佛在物理 LAN 上传送数据似的。

[0066] 通过以这种方式实现虚拟 LAN，保持了分区作为逻辑独立计算机的逻辑结构，因为在一个分区中执行的应用程序将数据传送给另一个分区中的另一个应用程序，仿佛其它分区驻留在不同物理计算机上似的。在许多实例中，这使配置成在非逻辑分区计算机上执行的应用程序能够以与逻辑分区计算机完全相同的方式运行。

[0067] 每个虚拟 IOA 122 与用于以与非虚拟 IOA 相同的方式发送和接收数据的以太网兼容帧的一个或多个帧缓冲器 124 有关。每个帧缓冲器 124 又包括代表缓冲器中的不同数据区的多个细片 126。每个细片 126 还包括指向分区中供相关客户 120 使用的本地缓冲器（例如，缓冲器 128）的指针。

[0068] 在所图解的实施例中，帧缓冲器 124 中的细片 126 代表各种客户 120 能够依次使用的共享资源。多个客户 120 可以消耗这些细片，由于客户在处理请求期间消耗这些细片，因此随着这些资源被消耗掉，有必要通知客户。例如，在所图解的实施例中，通过将中断从虚拟 IOA 发布到客户来通知客户。正如下面更明显看到的那样，按照本发明，可以成批地，而不是一个请求一个请求地管理这些工作，以提高调度进程的效率。

[0069] 在图解在图 4 中的实施例中，每个虚拟 IOA 122 起与图 1 的资源管理器 52 类似的资源管理器的作用。这样，每个虚拟 IOA 122 包括多个服务器进程 130、调度进程 132、request 计数器 134、done 计数器 136、last_done 计数器 138 和 last_dispatch 计数器 140。另外，每个服务器进程 130 存储 my_request 变量 142，以便唯一地标识相关服务器进程发出的每个请求。按照本发明的资源管理通常与从虚拟 LAN 接收数据一起出现，使得随着有关的数据被相关虚拟 IOA 接收，通过中断通知客户 120。

[0070] 现在转到图 5，图 5 更详细地图解了在图 3 和 4 的环境下特别适合管理客户请求的进程客户请求例程 172。例程 172 在许多请求中与例程 72 相似，但被配置成特别适合管理客户请求，以便虚拟输入 / 输出适配器消耗帧缓冲器中的细片。一般说来，每个客户请求原子地获取帧缓冲器中的可用细片，通过原子地递增 request 计数器将新唯一标识符指定给这样的请求。请求标识符模帧缓冲器中的细片个数用于获取要使用的特定细片。为了这种实现，这里称为 used_slot 计数器的附加计数器用于保持消耗细片的总数。每当客户释放资源时，通常通过在除了例程 172 之外的其它例程中实现的功能，使这个计数器递增。

[0071] 例程 172 从确定已用细片的总数是否等于可用细片的最大数（用 total_slots 常数或变量表示）的方块 174 开始。如果是的话，客户请求失败，并且如图解在方块 176 中的那样，返回错误。否则，方块 174 使控制转到方块 178，递增已用细片的个数。接着，方块 180、182、184 和 186 以与例程 72 的方块 74-80 相似的方式原子地递增 request 计数器并获取请求标识符。

[0072] 接着，控制转到方块 188，通过求当前请求标识符模帧缓冲器中的细片总数，为当前请求设置当前细片 (my_slot)。接着，在方块 190 中，准备和向当前细片提交请求，从而

完成请求的处理。这样，以与例程 72 的方块 84 和 86 相似的方式，控制转到方块 192 以获取计数器锁，然后，转到方块 194 以递增 done 计数器。例程 172 中的其余操作（具体地说，为如方块 196、198、200、202、204、206、208 和 210 所示的操作）与上面结合例程 72 的方块 88、90、92、94、96、98、100 和 102 公开的那些相同，方块 204 中请求的调度是通过对调度进程的中断，生成对分区中需要通知的各种客户的附加中断发生的。另外，与例程 72 的方块 100 和 102 相同，在按照本发明的一些实施例中，可以从例程 172 中省略方块 208 和 210。

[0073] 作为上述实现中使用本发明的实例，考虑在虚拟 LAN 上 4 个分区相互耦合的情形，虚拟 IOA 被指定给每个逻辑分区，而 3 个逻辑分区含有需要发送给第 4 分区的数据。考虑第 1 分区含有 64KB 要发送的数据，第 2 分区只含有 2KB 要发送的数据，和第 3 分区含有 64KB 要发送的数据。在这种情形下，假设第 1、第 2 和第 3 分区将请求发送给它们各自的虚拟 IOA，然后，这些虚拟 IOA 格式化要发送到第 4 分区的虚拟 IOA 的数据的帧。另外，虚拟 IOA 按数字顺序将请求发送到第 4 分区的虚拟 IOA，即，第 4 分区的虚拟 IOA 按顺序接收来自第 1、第 2 和第 3 分区的请求。这样，利用例程 172，将按序唯一标识符指定给请求，并且这些请求将按顺序消耗第 4 分区的虚拟 IOA 的帧缓冲器中的细片。

[0074] 但是，一旦将标识符指定给请求，虚拟 IOA 中的每个服务器进程将着手并行地管理它的请求，例如，将相应帧 DMA 到那个进程消耗的细片所指的客户缓冲器。但是，在所图解的例子中，由于来自第 2 分区的数据的帧比第 1 分区的数据的帧小得多 (2KB 比 64KB)，因此在完成第 1 分区的请求之前完成第 2 分区的请求是可能的。在那种情况下，例程 172 递增 done 计数器，并将 last_done 计数器设置成来自第 2 分区的请求的唯一标识符。但是，例程在方块 200 上还将检测到并非所有较早的请求都已完成，结果是，推迟通知第 4 分区中的客户，直到来自第 1 分区的请求都已完成，此刻，将发生成批调度，将接收到数据的两个帧通知第 4 分区。另外，假设来自第 3 分区的请求晚一些时间完成，则将发生另一次调度以通知第 4 分区接收第 3 帧数据。

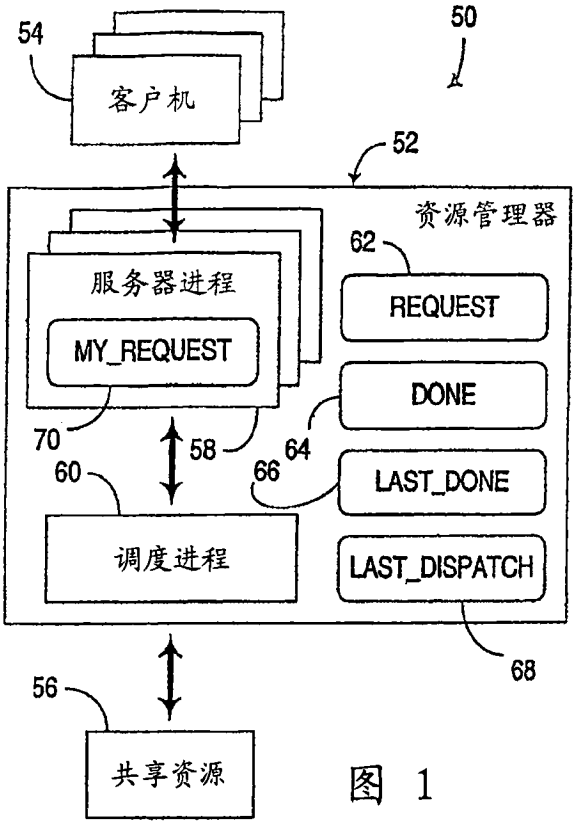

50
54
客户机
52
资源管理器
62
服务器进程
MY_REQUEST
REQUEST
DONE
70
58
64
66
60
LAST_DONE
调度进程
LAST_DISPATCH
68
56
共享资源

图 1


72
进程客户机请求
74
获取计数器锁
76
递增 REQUEST
78
MY_REQUEST = REQUEST
80
释放计数器锁
82
准备和提交请求
84
获取计数器锁
86
递增 DONE
88
MY_REQUEST > LAST_DONE?
否
是
90
LAST_DONE = MY_REQUEST
94
LAST_DISPATCH = LAST_DONE
是
DONE == LAST_DONE?
92
否
100
MY_REQUEST - LAST_DISPATCH == 1?
否
102
是
96
通知调度进程调度请求
LAST_DISPATCH = MY_REQUEST
释放计数器锁
98
完成

图 2

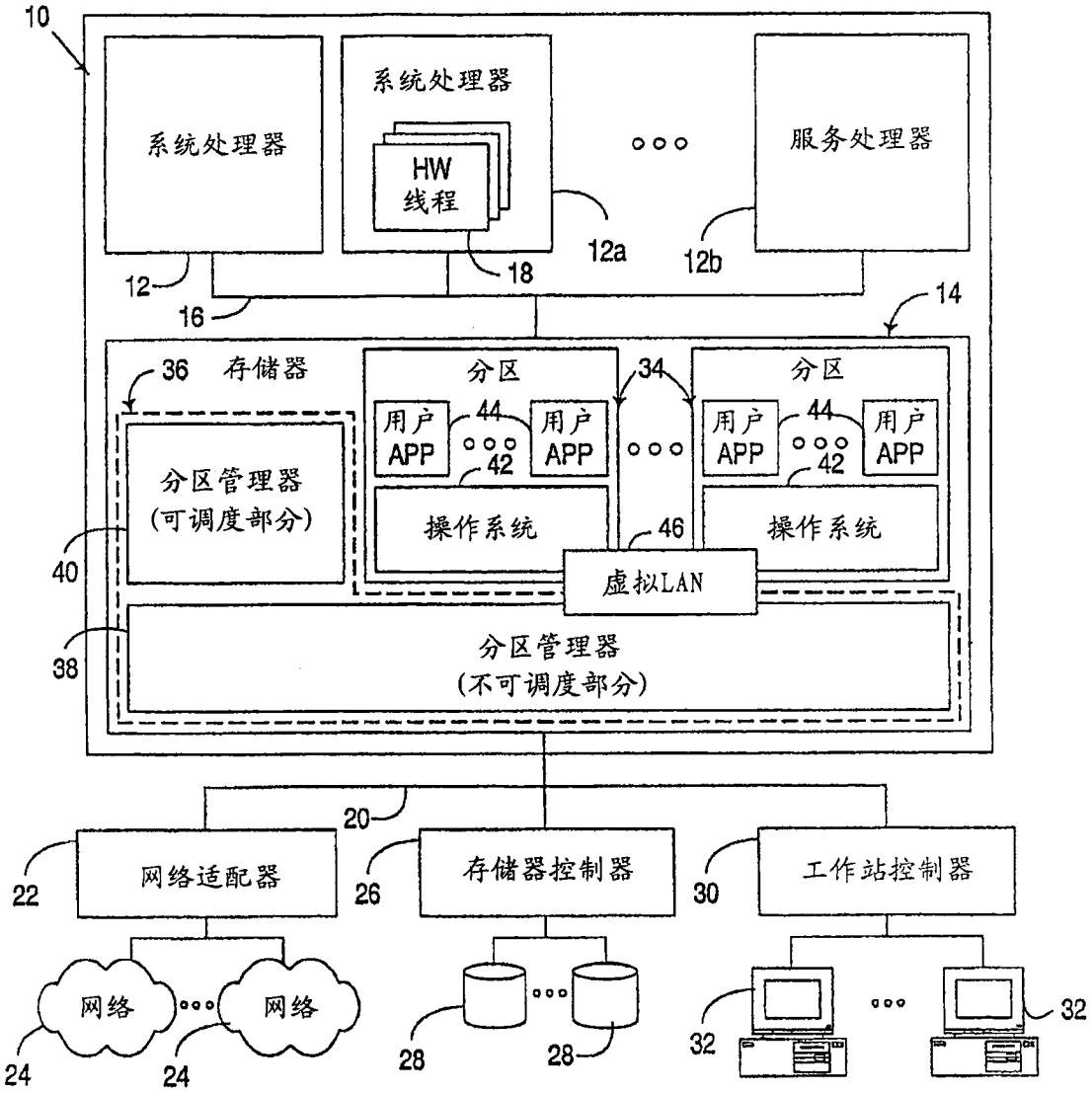

10
系统处理器
系统处理器
HW
线程
服务处理器
12a
12b
12
18
16
14
36 存储器
分区
34
分区
用户
APP
44
用户
APP
42
操作系统
46
虚拟LAN
分区管理器
(可调度部分)
40
38
分区管理器
(不可调度部分)
20
网络适配器
22
存储器控制器
26
工作站控制器
30
网络
网络
24
24
28
28
32
32

图 3

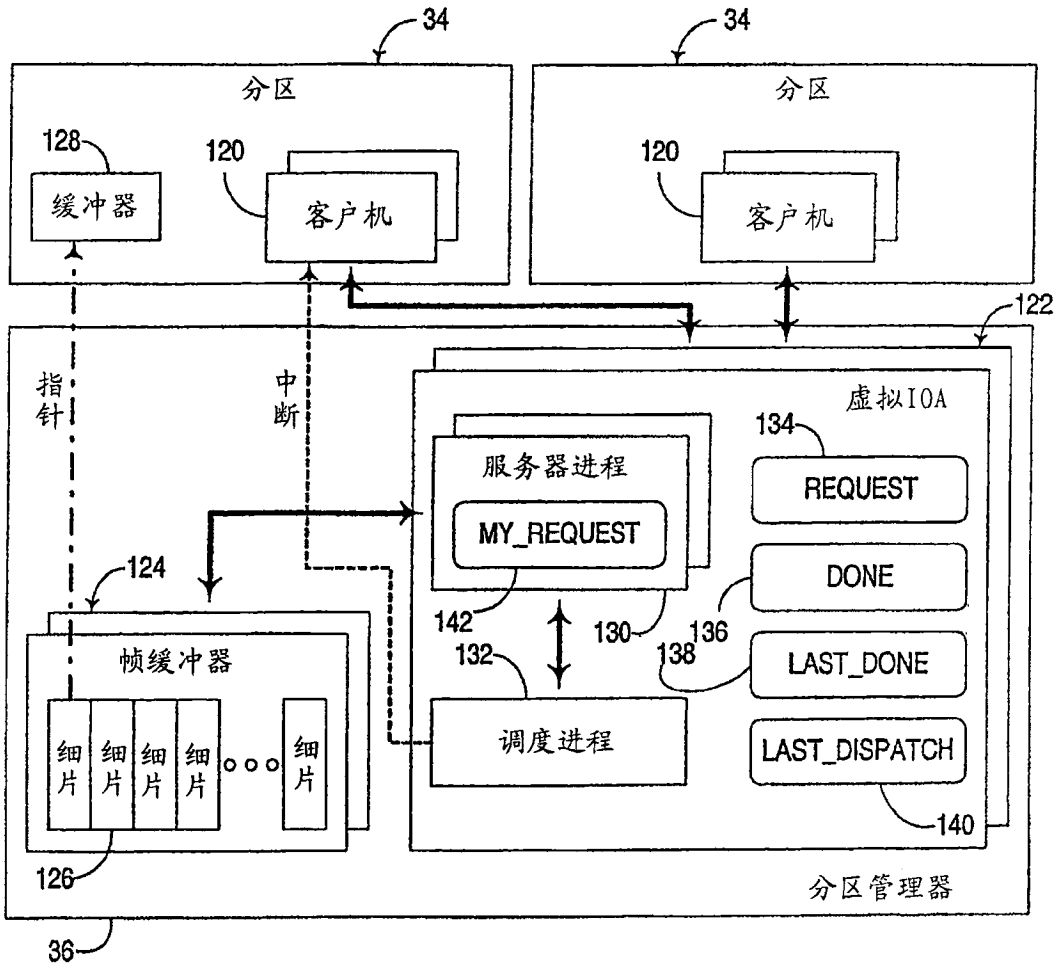

34
34
分区
分区
128
缓冲器
120
客户机
120
客户机
122
指针
中断
虚拟IOA
134
服务器进程
MY_REQUEST
REQUEST
DONE
LAST_DONE
LAST_DISPATCH
142
130
136
138
132
124
帧缓冲器
细片
细片
细片
细片
细片
调度进程
140
126
分区管理器
36

图 4

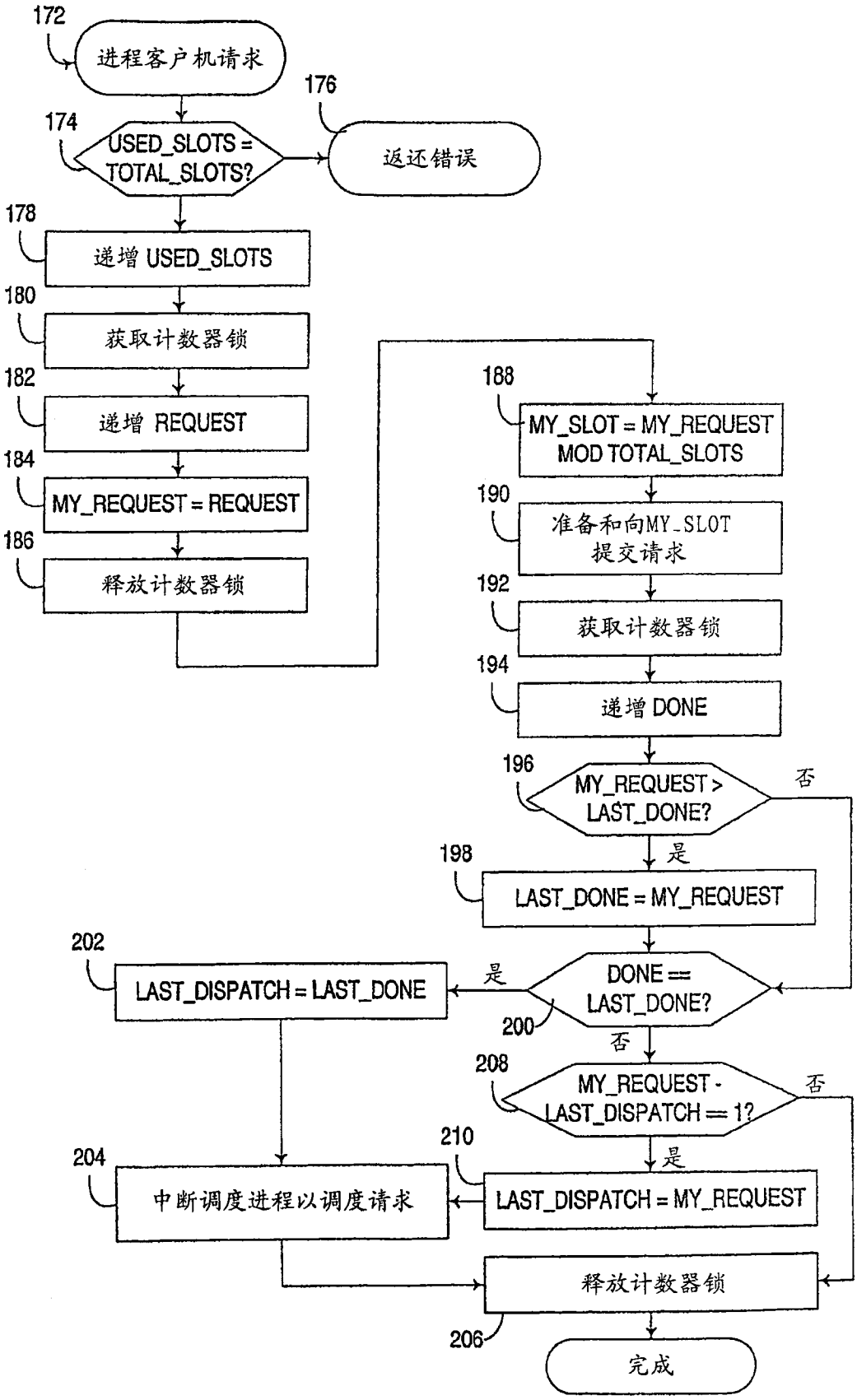

172
进程客户机请求
174
USED_SLOTS = TOTAL_SLOTS?
176
返还错误
178
递增 USED_SLOTS
180
获取计数器锁
182
递增 REQUEST
184
MY_REQUEST = REQUEST
186
释放计数器锁
188
MY_SLOT = MY_REQUEST MOD TOTAL_SLOTS
190
准备和向MY_SLOT 提交请求
192
获取计数器锁
194
递增 DONE
196
MY_REQUEST > LAST_DONE?
否
是
198
LAST_DONE = MY_REQUEST
202
LAST_DISPATCH = LAST_DONE
是
DONE == LAST_DONE?
200
否
208
MY_REQUEST - LAST_DISPATCH = 1?
否
210
是
204
中断调度进程以调度请求
LAST_DISPATCH = MY_REQUEST
释放计数器锁
206
完成

图 5