发明名称
在主机计算机网络上处理客户机请求的系统和方法

摘要
本发明提供了一种在主机计算机网络上处理客户机请求的系统/方法，它包括在主机中创建持久性容器，在容器中创建对象，把容器分组构成网格容器阵列，把单个网格容器阵列内的容器内的各对象分组构成网格对象阵列，为每个网格容器阵列创建一个调度器，把每个客户机请求分割成多个任务，以及把任务组赋予微处理器，其中微处理器把各个任务赋予对象。当微处理器传送完成后，任务组时本发明对微调度器赋予其他的任务组。该方法还能包括通过网关向多个网格服务调度器传送客户机请求。
1.一种处理客户机请求的计算机系统，所述系统包括：
与多个网格容器阵列相连的网格服务调度器，其中每个网格容器阵列包含：多个持久性容器，每个容器驻留在一个计算机主机中；以及在一个所述容器中的一个微调度器；其中每个所述容器包括多个对象，其中在构成一个网格容器阵列的那些容器内的对象构成一个网格对象阵列，其中所述网格服务调度器把一个客户机请求分割成多个任务并把所述多个任务赋予每个微调度器，而且其中每个所述微调度器把从所述网格服务调度器接收的多个任务中的各个任务赋予相应网格对象阵列中的那些对象。

2.权利要求1的系统，其中，在每个网格容器阵列中，所述微调度器在拓扑结构上位于所述网格容器阵列中的所有容器中性能最高的那个容器中。

3.权利要求1或2的系统，进一步包括多个网格服务调度器和网关，其中每个网关与不同的网格服务调度器相连接。

4.权利要求3的系统，进一步包括与所述网关相连接的门户，其中所述门户从所述网关把所述客户机请求传送给所述网格服务调度器。

5.权利要求1或2中的系统，其中每个网格容器阵列驻留在一个局域网中，从而在一个网格容器阵列内对象之间的通信构成局域通信。

6.权利要求1或2中的系统，其中所述网格服务调度器以透明方式分割所述客户机请求，从而使客户机不知道所述客户机请求的分割。

7.权利要求1或2中的系统，其中所述容器包含服务容器并在跨越多个客户机请求的时间段上处理来自多个客户机的多个客户请求。

8.一种在主机计算机网络上处理客户机请求的方法，所述方法包括：
在所述主机中创建持久性容器；在所述容器中创建对象；把所述容器分组构成网格容器阵列；把在单个网格容器阵列内的容器内的对象分组构成网格对象阵列；为每个网格容器阵列创建一个微调度器；把每个所述请求分割成多个任务；以及把所述多个任务赋予所述微调度器，其中所述微调度
器把各个任务赋予所述对象。

9. 权利要求8中的方法，所述方法包括：在拓扑结构上把所述微调度器放在所述网格容器阵列中的所有容器中性能最高的那个容器中。

10. 权利要求8或9中的方法，进一步包括：当所述微调度器回送完成后的多个任务时赋予所述微调度器其他的多个任务。

11. 权利要求8或9的方法，进一步包括通过网关向多个网格服务调度器传送所述客户机请求，其中，所述多个网格服务调度器与所述网格容器阵列相连。

12. 权利要求8或9的方法，其中每个网格容器阵列驻留在一个局域网中，从而在一个网格容器阵列内对象之间的通信构成局域通信。

13. 权利要求8或9的方法，其中所述分割是以透明方式进行的，从而使客户机不知道对所述客户机请求的分割。

14. 权利要求8或9中的方法，其中所述容器包含服务容器并在跨越多个客户机请求的时间段上处理来自多个客户机的多个客户机请求。
在主机计算机网络上处理
客户机请求的系统和方法

技术领域

一般地说，本发明涉及在计算机网络上处理请求，更具体地说，涉及一种改进的系统，它使用网格（grid）服务调度器把一个客户机请求分解成许多任务。该系统通过一个网格对象阵列中的多个对象来执行这些任务，这些对象处在容器（container）内，这些容器构成一个网格容器阵列，而该网络容器阵列由一个微调度器控制。

背景技术

可以说，网格计算是群集计算的一个逻辑演变。下文中使用 Globus 工具箱（可从位于美国依利诺斯州芝加哥的 Argonne 国家实验室得到）作为实现网格体系结构的参考点来考察这种说法。在考虑 Globus 工具箱时，应该指出这一技术正以很快的速度演变，每次发布都有很明显的不同。特别是 Globus 3.0 体系结构是一个大的飞跃，这是由于它基于开放网格服务体系结构标准。所以，在讨论 Globus 时，必须说明确切的发布版本，以设定讨论的上下文。

例如，考虑 Globus 2.0 实现，它领先于开放网格服务体系结构（OGSA）。在考察 Globus 2.0 体系结构时，很快便清楚它把网格定义为“群集之群集”。这一复合观点自 Globus 程序出现便已经表现出来；用于网格计算的老的术语曾是“元计算（meta-computing）”；这是对多群集体系结构的复合性质的一个引用。“群集之群集”观点与 Globus 程序的初始目标一致，即以单一基础设施统一位于地理上分离的国家实验室的若干群集。当考虑网格的可伸缩性时，这一观点也很有意义。向网格添加各个
主机将很快造成一个不可管理的基础设施，而且把主机分组成单独管理的若干组是一个逻辑步骤。“群集之群集”范例提供一个双层的层次结构，它大大简化了对网格的管理。

然而，“群集之群集”范例的两个限制立即显现出来。第一个限制是这一方案是因有的双层结构。就是说，它只能支持双层层次结构，为了支持更深的层次，它必须扩展到原始的框架结构。第二个限制是不存在跨整个网格调度任务的中央设施；就是说，不存在元调度器。每个群集被配置一个群集调度器，它有权调度那个群集中的任务，但很快便清楚，必须有一个网格调度器（或元调度器）用于在整个网格上调度任务。设计这种调度器的主要困难在于，在原始的 Globus 工具箱（2.0 版或更低版本）中在网格世界和群集调度世界之间存在一个间断。特别是，在一个群集内和在各群集之间的任务调度使用不同的调度技术和协议。下面描述的本发明提议一个用于网格和群集两个域的统一的调度域，所以它能无瑕地支持层次结构很深的网格。

发明内容

本发明提供一种在主机计算网络上处理客户机请求的方法，它包括在各主机中创建持久的容器，在容器中创建对象，把容器分组成网格容器阵列，把处在单个网格容器阵列内的那些容器内的对象分组到网格对象阵列中，为每个网格容器阵列创建一个微调度器，把每个客户机请求分解成多个任务，以及把多个任务赋予多个微调度器，这里微调度器把各个任务赋予对象。本发明在微调度器返回已完成任务组时赋予微调度器额外的任务组。该方法还包括通过网关把客户机请求发送到多个网格服务调度器。

本发明还提供处理客户机请求的系统，该系统包括与网格容器阵列连接的网格服务调度器。每个网格容器阵列包括多个持久的容器（例如服务器），各容器驻留在不同的计算机主机中，还包括一个微调度器。每个容器包括许多对象。在构成一个单个网格容器阵列的那些容器内的对象构成一个网格对象阵列。网格服务调度器透明地把一个客户机请求分解成多个
任务把多组任务赋予每个微调度器。每个微调度器把从网格服务调度器接收的任务组中的各任务赋予其相应的网格对象阵列中的对象。

该系统能包括额外的调度器层次结构级别并包括网关，各网关与不同的网格服务调度器连接。门洞与网关相连，这些门洞沿网关把客户机请求传送给网格服务调度器。

每个容器阵列驻留在一个局域网中，使在网格容器阵列内对象间的通信为局部通信。网格服务调度器以透明方式分解客户机请求，使客户机不知道对客户机请求的分解。容器是持久性的服务容器，用于在跨越多个客户机请求的时间段上处理多个客户机的多个客户机请求。

附图说明

由下文中参考附图对优选实施例的详细描述，将更好地理解本发明，其中：

图 1 是由两个不同组织共享的一个体系结构的示意图；
图 2 是示意图，说明本发明的按需要创建容器的能力；
图 3 是示意图，说明在各种容器中创建网格对象；
图 4 是示意图，显示不同对象间的交互作用；
图 5 是示意图，说明本发明如何把任务转发给网格对象阵列中的对象；
图 6 是示意图，说明在一个复合网格容器中的多个客户机调度器；
图 7 是示意图，说明使用网格门户的层次调度；以及
图 8 是示意图，说明本发明在网格中的布局。

具体实施方式

如前述述，曾使用不同的调度技术和协议在群集内或群集间调度任务。本发明的“关注拓扑结构的网格服务调度器”（TAGSS）调度体系结构提出一个用于网格和群集两个域的统一的任务调度域，它能无缝地支持层次结构很深的网格。

本公开内容考察为网格设计元调度器所伴随的挑战。具体地说，我们
将讨论群集环境和网格环境之间的差别，而且我们认为传统的群集调度体
系结构不适用于网格环境中的元调度。当我们考虑到由于引入基于服务的
体系结构使新的网格技术与传统的群集计算的距离更远时，这一说法就是
特别正确的了。当我们讨论这些方面时，我们引入所发明的“关注拓扑结
构的网络服务调度器”（TAGSS）的体系结构，它是为面向服务的网格体
系结构设计的。

在这一节我们简单地说明群集和网格环境之间的那些影响任务调度基
础设施设计的主要差别。关于拓扑考虑，第一个考虑涉及总拓扑基础设施。
通常，群集有统一的拓扑结构，即计算主机通常有相同类型，在许多情况
下是完全相同的，而且通常通过一个共同的网络可达到所有主机。在一些
情况下，群集可装有高速网络。显然，这一环境很不同于平均情况的网格，
因为用户可通过广域网达到网格资源，而且根据广域网链路的特性，将以
不同的服务质量达到每个资源。网络速度的变化是在设计 TAGSS 时的主
要考虑之一，这在以下章节中会变得清楚。

关于服务调度与作业调度关系，要做的另一重要考虑是关于近来向基
于服务的体系结构的转变，如在 OGSA 标准中提议的那样。基于服务的体
系结构与大多数群集调度器使用的基于作业的体系结构有相当大的差别。
这里我们应该仔细地定义术语“作业”的含义。我们使用为群集计算可用
的大多数任务调度中间件已使用的术语，这些中间件的实例有：Condor（
可从位于美国 Wisconsin 的 Madison 的 Wisconsin-Madison 大学得到）、PBS（可从位于美国加州 Mountain View 的 Veridian Systems 得到）、LSF
（可从位于美国加州 Markham Ontario 的 Platform Computing 得到）以
及 Load Leveler（可从位于美国 Armonk 的国际商业机器公司得到）。所
有这些调度器都被称为作业调度器。下文中比较作业调度器和服务调度器
的特点。

关于作业和作业调度，一个作业与一个或多个进程相关联。当我们说
一个作业被调度时，这是指一个或多个进程在目标主机中被创建。利用与
作业所关联的进程的创建有关的参数描述作业；如命令行语法、环境变量；
还根据它需要的资源，如 CPU 或主机数、运行时库等来描述作业。作业的另一有意义方面是数据通信模型。业务处理输入和输出数据的方式。

使用命令行中的变量以及环境变量传送输入数据；在这两种情况中被支持的格式只有文本。就是说，输入数据是作为文本串传送的，这当然可以指含有适当输入数据的文件。输出数据以类似的方式传送；可以从标准输出管道得到文本输出，但通常输出数据是文件形式的。所以，一个作业通常与一组输入和输出文件耦合，由于这一原因，大多数群集调度器实现一个文件集合功能，它给出在执行一个作业之前将文件传送至远程主机和在作业完成时提取这组输出文件的能力。这一定义规定一个作业与单一工作单元关联，因为它包含的进程由单个一组输入参数（通常代表单一用户）启动而且通常在产生输出后作业进程终止执行。由于这一原因，与一个作业关联的进程可被描述为瞬态的。作业的瞬态特性把它的编程模式限定于批执行。

关于服务、服务请求和请求调度，一个服务可以同等地关联于更多进程之一。然而，与服务关联的进程通常是持久性的。就是说，一个服务在完成一个给定任务之后它并不停止执行，相反，它是一个持久性的应用，很像一个服务器，能执行多个工作单元或多个服务请求。所以，请求调度活动不涉及任何新进程的创建。相反，请求调度包括使来自若干用户的服务请求按照某种管理策略进行排队，然后这些请求被引导到能更好地完成给定请求的适当的服务实例。服务提供者的这种持久性特性使得能够实现实时编程模式。至于数据管理方面，一个服务使用从标准（如 WSDL 和 SOAP）中导出的更灵活的基础设施。输入和输出参数能被描述为支持若干处理器体系结构的可移植对象。

下表给出根据上述定义中讨论的各方面对作业和服务请求的比较：

表 1: 作业和服务请求的特点

<table>
<thead>
<tr>
<th>方面</th>
<th>作业</th>
<th>服务请求</th>
</tr>
</thead>
<tbody>
<tr>
<td>执行模式</td>
<td>瞬时进程</td>
<td>提供持久性服务</td>
</tr>
<tr>
<td>数据管理</td>
<td>文本参数和文件</td>
<td>结构化数据</td>
</tr>
</tbody>
</table>
关于任务调度粒度的重要性，在进入对 TAGSS 体系结构的描述之前，我们再对使用比作业更细粒度的服务请求的优点做最后一点说明。因为服务请求是小得多的工作单元，它使这些应用只是外包（outsource）计算任务的一部分。这一特性允许一个应用的大部分代码和数据被包含在单个主机或组织之内。这一特性在其应用代码具有专有性质的应用中是一个重要要求，因为只有较少的代码段需要被移到外包组织，而且在标准服务外包（如基本教学例程）的情况中不需要传送代码。这一特性的更一重要应用是应用要处于大量数据的情况。在许多情况中，应用有可能人工管理数据和只传送资源外包的操作所严格需要的数据。这是一个重要的考虑，许多应用在公开其专有代码方面对这一考虑具有敏感性。在因只需传送不可缺少的数据而获得的效率方面，这也是一个重要的考虑。图 1 显示一个群集，它是一个大部分在单个组织中运行的应用，只外包小量并执行任务。在图 1 中，对象 128 用于完成特定的任务。对象 128 被包含在群集 125 中。两个组织 120、121 使用群集调度器和网格网关 130 通过群集 125 处理数据。与此相反，本发明提供一个网格，它利用若干不同组织内的任意个节点。在讨论了群集计算和网格计算之间的差别，特别是关于拓扑和任务调度方面之后，我们已建立了讨论本发明的“关注拓扑结构的网格服务调度器（TAGSS）”的背景。

本发明提供一个面向服务的体系结构。这样，我们开始讨论一个实时调度场景，而首要考虑是准确性实时调度的含义。在上一节中已经讨论过，不同于批执行，服务提供者能提供实时编程。我们结合某些客户进程的通信模式来定义这些术语。

一个作业请求或服务请求是由执行客户进程的某一客户机发起的。在批编程模式中，客户进程在发出作业请求后退出。再有，在批编程模式下，在作业和客户进程之间通常没有进行工作的通信信道。在实时编程模式中，
客户进程能与持久性服务提供器交互作用。在考虑服务请求调度器设计时，实时交互作用特别重要。特别是，实时交互作用有可能使服务成为全状态的，即服务能保持与给定客户机的交互作用有关的数据。处置状态的能力近来已加到万维网（Web）服务技术中，而且它是在 OGSA 中的主要设计点。

为了讨论全状态模式的优点，考虑它带给编程模式的好处。因为有可能在服务提供器中保持工作状态，客户进程有能力使一个服务请求使用对先前请求的结果的引用作为输入。我们将把这一特性称作“服务器一侧高速缓存”，我们将在介绍 TAGSS 编程接口时讨论 TAGSS 体系结构如何实现这一功能。

为了介绍 TAGSS 编程模式，参考 TAGSS Java 编程接口是有帮助的。存在以本机语言（如 C）的联编，但对于这一讨论使用 Java API 是有好处的，这是由于服务和 Java 对象有很强的相似性。事实上，有可能利用自动工具把 Java 对象定义映射到 WSDL 服务描述。在这一节我们公开基于服务的定义所采用的框架。

关于容器服务，TAGSS 体系结构的目的之一是提供一个简单的实时编程模式。在 Java 中，对象是在称作 Java 虚拟机（JVM）的运行时环境内执行的，在一个服务类似物中，服务是在容器内被执行的。TAGSS 体系结构遵循这一类似，定义一个容器服务。容器服务提供与对象创建和对象方法调用关联的服务。这些概念与服务事例化和服务请求处理相关联。

本发明提供容器服务“工厂”。定义一个服务的主要方面之一是它被创建和被破坏的方式。在 OGSA 中，根据在工厂接口中实现的规格创建服务实例，在提出用于网格的编程模式时，其主要问题是如何运用这一接口。为了由一个客户进程发送一个服务请求，它必须与一个服务提供器实例绑定在一起。因为服务创建任务需要遵循更新的资源利用测量，TAGSS 体系结构定义一个资源监视服务，它在创建容器服务时起主要作用。这个资源监视服务被称作调度器服务。调度器服务恒定地监视节点池中的资源利用情况，而且它控制那个节点池中容器服务的创建。
调度器服务不直接创建容器服务。在 TAGSS 体系结构中，调度服务是由下述图 2 所示进程/系统实现的。作为中央进程的 TAGSS 调度器 200 收集节点池 210（或最小调度域）中的资源利用情况测量结果，并在这个最小调度域中处理对容器服务的所有请求。对于一个给定的最小调度域，只有一个 TAGSS 调度器实例 200。

TAGSS 代理 205 进行周期性的资源利用情况测量并把这些测量结果报告给它的指定最小调度域的 TAGSS 调度器 200。TAGSS 代理 205 还负责实现容器服务的工厂接口。容器服务可以是专用于给定客户机的，或者它们可以根据服务政策的质量由若干客户机共享。在下文的场景中讨论这两种可能性。

当绑定于排他的容器服务时，图 2 显示消息流（1-6），说明如何响应一个客户机请求按需要创建排他的容器服务。如图 2 中所示，客户进程 215 首先向 TAGSS 调度器 200 发送一个请求，以获得一个容器（1）。然后，TAGSS 调度器 200 向 TAGSS 代理 205 发送一个指令（2）以创建一个容器。TAGSS 代理 205 以产出（3）一个新容器 220 作为响应。然后 TAGSS 代理 205 向 TAGSS 调度器 200 返回容器引用指针（4）。然后，TAGSS 调度器 200 向客户进程 215 返回容器引用指针（5）。然后客户进程 215 绑定（6）于新容器 220。可以指出，客户进程 215 只与 TAGSS 调度器 220 和容器服务实例 220 直接通信，而不与 TAGSS 代理 205 直接通信。TAGSS 代理 205 的存在对于客户进程 215 是透明的，如同是调度器服务的一个集成的部分。

关于绑定于共享的容器服务，如前所述，客户进程共享容器也是可能的。这一做法的好处在于它减小了每个网格主机 210 中创建的容器进程的个数。有可能使用具有不同运行时执行优先级的共享容器提供不同级别的服务质量。于是，客户机到指定容器的映射将按管理策略或服务级别协定（SLA）来完成。然而，共享容器的布置要求一个复杂的安全性基础设施。本发明能使用一个类似于 Java Servlets（服务小程序）环境的一个框架来提供共享容器所需的适当的安全性。
下面的编程例子说明一个客户进程 215 与一容器服务 220 的绑定。TAGSS API 的这一解释不需要区分共享的和非共享的容器，因为如果给出这一公开则本领域普通技术人员将会理解这些差别。下列代码段说明一个客户机如何绑定于单个容器服务（6）。如图 3 中所示，调度器服务 200 创建网络对象（对象 1—对象 8）。当一个网格对象（对象 1—对象 8）被创建时，客户进程 215 被绑定于给定的调度器服务 200。当构建网格对象（对象 1—对象 8）时，TAGSS 调度器 200 的网络地址和端口号被作为构造器变量而传送。一旦网格对象（对象 1—对象 8）被创建，客户进程 215 能通过执行图 2 中的 Grid. get Container() 方法（1）用该网格对象绑定于若干容器服务 300-301。

```
GridmyGrid=new Grid("tagss_scheduler_hostname", tagss_scheduler_port);
GridContainer myContainer=myGridContainer.getContainer();
```

代码段 1：连接到一个网格提供器

关于创建对象（对象 1—对象 8），一旦客户进程 215 被绑定于容器服务 300-301，便能创建其他服务。在 Java API 中，这些服务由 Java 对象代表。新对象的创建是由调用 GridContainer.createObject 方法（1）完成的。这一方法需要创建完全合格的对象名以及构造器参数。如图 3 中所示，网格对象阵列 (8) 指令指导 TAGSS 调度器 200 去创建八个对象。在容器 300 和 301 每个中创建三个对象（对象 1—对象 3；对象 4—对象 6），而在容器 302 中创建两个对象。构造器参数是使用 Java ArrayList 直接从编程环境中取出的。在内部，TAGSS API 将使用 JAVA 反射类，以使传送到 ArrayList 中的对象序列匹配于给定方法的适当构造器。

```
//construct a GridObject of type myObjectClass.
//The constructor has the signature (Integer, Double, String);
Integer arg1=new Integer(123);
Double arg2=new Double(1.23);
String arg3= "hello"
ArrayList args=new ArrayList();
```
args.add(arg1);
args.add(arg2);
args.add(arg3);

GridObject myObject =
myContainer.createObject("myObjectClass", args);

// assume that myObject was built as in Code Segment 2.
// Prepare to invoke a method with the signature(int, real, String);
Integer arg1 = new Integer(123);
Double arg2 = new Double(1.23);
String arg3 = "hello"
ArrayList args = new ArrayList();
args.add(arg1);
args.add(arg2);
args.add(arg3);

MyReturnObjectClass myResult = (MyReturnObjectClass)
myObject.invoke("myMethod", args);

关于服务器一侧高速缓存, TAGSS API 提供一个 GridObject.invoke() 方法的变体, 它实现先前讨论过的服务器一侧高速缓存特性。GridObject.invokeKeep() 只返回一个到方法调用结果的引用指针，而结果本身被保持
在该方法被调用的容器中。该结果可在以后用 GridObject. getValue() 方法检索。下列例子显示一个应用怎样将若干个调用拼接到 GridObject. invokeKeep() 方法，以最大限度地减少不得不在客户机和服务提供器之间传送的数据量。在下面的例子中，一个图像增强应用把由三个过滤器组成的序列应用于一个图像并且只提取其最终结果。没有任何中间结果回送到客户进程，节省了中间结果的整个来回旅程。这一技术对于管理大宗数据的应用（如图像处理）是重要的。

```java
//construct a GridObject of type myObjectClass.
//The constructor has the signature (Integer, Double, String);
Integer arg1=new Integer(123);
Double arg2=new Double(1.23);
String arg3= "hello"
ArrayList args=new ArrayList();
args.add(arg1);
args.add(arg2);
args.add(arg3);
GridObject myObject=
   myContainer. createObject ("myObjectClass",args);
```

代码段 4：使用网格对象中的持久状态
再次说明，返回值能被转换成任何用户定义的类类型。

关于更高级的结构，上文描述的基本原语提供了很灵活的接口，以管理网格环境中的各单个对象。然而，大多数网格应用为执行并行化任务而要求更高级的结构。TAGSS 编程接口提供一组更高级原语用于并行操作。在一个实施示例中，本发明提供三个更高级操作：消息多路传播，障碍强化方法调用和分散/集合操作。这些功能是由 TAGSS 中的管理下文所述对象集合的附加原语提供的。

一个网格容器阵列 (GridContainerArray) 307 代表容器 300-302 的集合。容器阵列实现容器的同样功能组，而在一个容器阵列中被调用的方法
应用于作为该容器阵列一部分的每个容器。容器阵列工厂接口也由服务调度器实现，它由网格对象 (对象 1—对象 8) 代表。构造容器阵列的方法是 Grid.getContainerArray()。这一方法取一个整数作为变元，代表所希望的容器个数。在资源短缺时，返回的实际容器数小于所希望的个数。通过调用 GridContainerArray.getNumberContainers() 能得到返回的实际容器数。如果传送值 0（零）作为所希望的容器个数变元，则网格调度器将根据资源利用状态并根据该客户端的凭证返回全部可用容器。下例显示容器阵列构建器的语法：

GridContainerArray myContainerArray = Grid.getContainerArray(0);
int numberOfContainers = myContainerArray.getNumberContainers

代码段 5：创建一个网格容器阵列

如前所述，容器阵列 307 实现与单个容器类似的一组功能。如同一个容器是一个创建网格对象（对象 1—对象 8）的工厂，网格容器阵列 307 是类型为网格对象阵列 306 的对象的工厂。创建网格对象阵列（GridObjectArray）306 的方法是 GridContainerArray.createObject()。这一方法以要创建的对象类型名（类名）以及一个数据结构作为输入变元，该数据结构含有要被调用的每个构造器调用所用变元。该变元列表应有相同的签名，即在变元序列中以相同值索引的每个对象应该有相同的类类型。为了有助于构造变元列表以创建对象阵列，TAGSS 接口使用一个辅助数据结构，称作网格数据集（GridDataSet）。可以将网格数据集可视化为一个矩阵，它的每行是要应用于一个对象阵列中的一个构造器或方法调用的变元列表。所以，网格数据集的每列应含有相同数据类型的对象。例如，下列指令说明如何为调用签名整数、双精度数和字符串的构造器而构建网格数据集：

// assume that the input values are in intArray, doubleArray, and StringArray
GridDataSet myGridDataSet = new GridDataSet();
ArrayList singleRow = new ArrayList();
for (i = 0; i < inputSize; i++)
代码段 6: 创建网络数据集

在网格数据集中的行数不需要与容器个数相同，指出这一点是重要的。本发明只是在容器的子集中构建对象，并通过在一个容器中创建不只一个对象实例来构建多于容器个数的对象，如图 3 中所示。更具体地说，图 3 显示一个客户端，它试图创建一个具有八个对象的网格对象阵列，这里只有三个容器可用。该客户端通过在可用容器内创建多个对象，仍然成功地创建了一个具有八个对象的网格对象阵列 306。

当在对象阵列 306 中执行一个方法时，对象阵列 306 有与容器阵列 307 相似的语义。指向一个对象阵列 306 的方法调用在对象阵列 306 的每个对象实例中造成方法调用。这一多重方法调用的结果被返回到网格对象阵列，这里第 i 个元素是使用网格数据集的第 i 行的一个方法调用的结果。下列代码段说明在一个网格对象阵列上的一个方法调用:

```java
//create a Grid Object Array
GridObjectArray myGridObjectArray=
GridContainerArray.createObject (myGridDataSet);
//assume that we loaded the input data in inputSet
GridDataSet inputSet=loadDataSet();
//invoke a method on a Grid array
GridObject[]resultArray=
myGridObjectArray.invoke("methodName", inputSet, invocation_mode);
```

代码段 7：在网格对象阵列上调用一个方法
下面描述在网格对象阵列 306 中方法调用的三种方式，使用变元 TAGSSInvocationMode 指定这三种方式:

第一种方式是 TAGSS_INVOKE_MULTICAST。这一调用方式用于单个参数列表，不像下述其他两种方式那样使用网格数据集。在这一调用方式中，在网格对象阵列 306 的所有对象中，以在单个参数列表中包含的完全相同的参数集合来调用一个方法。这一调用方式用于向各对象发送同步消息。

另一种方式是 TAGSS_INVOKE_EACH。这一调用方式指定应把网格数据集中的每一行应用于阵列中的相应对象。就是说，行 0 应用于对象 0，行 1 应用于对象 1，等等。这一方式要求网格数据集中的每行等于或小于阵列中的对象数。这一方法调用方式用于障碍强制（barrier enforced）方法调用。

再另一种调用方式是 TAGSS_INVOKE_ANY。这一调用方式表明在网格数据集的各行与网格对象阵列 306 中的对象之间没有指定的映射要求。这一方式为底层调度基础设施提供了大的灵活性。例如，有可能告诉具有 10 个对象的对象阵列去调用具有 1000 行的网格数据集上的一个方法。如下所述，哪些对象将在哪些行上工作这种映射是由 TAGSS 中的底层调度基础设施在运行时完成的。这一调用方法用于分散/聚集编程模式中。上述描述的这三种调用方式实现了 TAGSS 中实现的三个基本的并行编程模式：消息多播、障碍强制方法调用以及分散/聚集。下文中将更详细讨论这三种编程模式。

消息多播模式是简单地向一个对象集合发送单个消息的一种编程模式。TAGSS API 以提供一个方法调用函数来支持这一模式，该函数取单一变元列表并把它应用于网格对象阵列 306 中的所有对象:

```java
GridObject[][] result=myObjectArray.invoke("methodName",argList,
                                      TAGSS_INVOKE_MULTICAST);
```

代码段 8：使用消息多播编程模式

障碍强制并行方法调用是能用于实现若干并行编程方法的一种编程模
式，这些方法需要多个对象工作前后紧接的数据。在对象 400-403 之间的前后紧接的操作示于图 4。在这一调用中，所有对象在下一方法能被调用之前得要完成某一方法，就是说，存在一个隐含的障碍，它同步每个方法的执行。例如，下面一段代码实现一个流水线，它把先前计算的结果以移位（shiftcarry）方式重定向到阵列中的每个对象，使用障碍同步保证每个方法调用之前那些结果是完全的：

```java
// barrier synchronization example
// assume the input is already loaded in GridDataSet
// invoke 3 methods using a result pipeline that shifts to the right
GridObject[] result=myGridObjectArray.invoke("first_stage",
        inputDataSet, TAGSS_INVOKE_EACH);

// shift the result to the right
GridObject temp=result[result.size()];// keep the last element to wrap around
for(i=0;i<result.size()-1;i++)
    result[i+1]=result[i];
result[0]=temp;// wrap around
// now call the second stage
GridObject result 2 =myGridObjectArray.invoke("second_stage",
        InputDataSet, TAGSS_INVOKE_EACH);
```

代码段 9：实现移动-结果-障碍模式

例如，对四个对象执行上述代码将会建立图 4 中所示通信模式。

当指定 TAGSS_INVOKE_ANY 调用方式时，使用分散/聚集模式。在这种情况下，根据运行时条件，网格数据集的那些行被赋予任何网格对象。根据每个对象完成方法调用的能力进行行映射。运行时调度由一个内置调度代理完成，它是为了客户机应用在运行时创建的，称作 TAGSS 客户机调度器（或微调度器）500，如图 5 中所示。

微调度器 500 是一个小的多线程对象，它是在构建网格容器阵列 307
时被隐含构成的。每个网格容器阵列 307 有它自己的 TAGSS 客户机调度器 500 的实例。TAGSS 微调度器 500 在网格对象阵列 306 中的各对象 (对象 1—对象 8) 之间分散各行，然后调度新行，这些新行是由对象 (对象 1—对象 8) 完成的任务。所以，如图 5 中所示，客户进程 215 通过一个数据集调用网格对象阵列 306 (11)。TAGSS 调度器 200 通过该数据集调用一个指定方法 (12)。TAGSS 微调度器 500 调用到各对象 (对象 1—对象 8) 的数据集行，每个对象对这些行之一完成一个指定任务。一旦由各对象完成了这些任务，TAGSS 微调度器 500 返回该网格对象 (14)。

TAGSS 客户机调度器对象的放置考虑了 TAGSS 总体结构，并根据网格拓扑来进行，考虑到 TAGSS 微调度器 500 产生大量网络传输和 I/O 处理。运行 TAGSS 客户机调度器 500 的容器 300 在消息多播和分散/聚集模式中都是中心坐标点。TAGSS 微调度器 500 与对象阵列 306 中的所有对象 (对象 1—对象 8) 通信并恒定地传送用于方法调用的输入数据以及在一旦完成时聚集输出数据。

这些情况表明客户进程 215 是为放置 TAGSS 微调度器的一个不好的选择。这是因为在网格中的通常配置把客户机进程 215 放在网络的边缘，经由速度低于网络中央的网络连接。

为了为 TAGSS 微调度器 500 选择最佳位置，TAGSS 体系结构把 TAGSS 微调度器 500 放在将提供最佳性能 (与那个网格容器阵列中的其他容器相比) 的容器中。容器的选择是根据创建容器阵列 307 时可得到的性能测量结果 (例如处理速度、输入/输出的能力和容量等)。按照约定，TAGSS 调度器服务 200 按服务客户机的下降顺序对容器排序，以编索引为 0 (零) 的容器为该网格的最佳性能容器。当创建一个网格容器阵列 307 时，TAGSS 微调度器 500 被创建在容器 0 (零) 中。

关于批调用，运行时调度进程支持成批的调用。就是说，当一个客户机希望在发出一个大的并行请求之后退出时，TAGSS 体系结构通过实现由 GridObject.invokeAsync() 方法提供的并步调用选择来提供那个功能。这一方法返回一个请求引用指针：网格批调用 (GridBatch)，它是对网格对象
的一个持久性引用指针。客户进程能把这一引用指针保存在持久性存储器中并在以后检索该网格对象。客户进程能使用方法 GridBatch.peek()查询是否完成一个网格匹配调度。该方法在完成网格调度时返回“真”，在此时能通过调用 GridBatch.getResult()方法检索结果。这一处理的示例代码如下所示:

```java
// carry out three large operations in parallel, and then wait on all complete
GridBatch batch1=gridObjectArray1.invokeAsync("method1", dataSet1);
GridBatch batch2=gridObjectArray2.invokeAsync("method2", dataSet2);
GridBatch batch3=gridObjectArray3.invokeAsync("method3", dataSet3);
// check if batch 1 is complete
Boolean isComplete=batch1.peek();
// wait on batch 1 to be complete
GridObject[] result=batch1.waitOnComplete();
// wait on both batches 2 and 3 to complete
GridWaitCondition waitCondition=new GridWaitCondition(batch2);
   waitCondition.add(batch3);
   waitCondition.waitOnComplete();
```

代码段 11：使用等待条件来同步并行调用

本发明适用于交互的和离线的客户进程两种情况。上述编程范例不仅限于交互的客户进程。客户机请求能被离线提交。在这种情况下，执行该客户进程的代码本身需要作为离线作业请求被发送。然后，客户进程的执行将遵循该作业调度范例。一旦该客户进程被调度和发起，则建立具有调度资源的实时对话，然后调度进程正如前述那样实时地工作。

TAGSS 体系结构还能在分布式环境中很好地工作。为了有最佳性能，具有相似特性的节点应分组到由单个服务调度器控制的一个极小调度域。极小调度域的大小将影响运行 TAGSS 微调度器 500 的节点的性能。

TAGSS 体系结构提供一种简单的方式跨多个极小调度域分散任务。如前所述，任务调度是由 TAGSS 微调度器 500 完成的，它是在每个网格客
器阵列 307 中的一个单独容器中创建的小型多线程对象。为了跨极小调度域进行调度，使用多个网格对象阵列构建一个组合网格对象阵列 306。然后调度任务在每个网格对象阵列 306 中的 TAGSS 客户机调度器中被分割。

这一强有力的特征对客户进程是透明的。正如 TAGSS 微调度器 500 对象是由 TAGSS 基础设施隐含构建的一样，根据上述对象创建和方法调用的语义，工作负荷在多个对象之间的分割也是隐含发生的。

为了在多个对象当中分散任务，一个客户机应用做的全部事情是构建一个组合网格对象。这是使用下述方法 Grid.add()完成的。这一方法把该网格对象绑定于另一调度域的 TAGSS 调度器 200。例如，在下文所示代码及图 6 中，创建了一个绑定于“网关 1”600 的网格对象阵列 306，然后它又被使用 Grid.add()方法绑定于“网关 2”601。第二网关 601 连接于网格容器阵列 308，它基本上与上文详细讨论的网格容器阵列 306 相似。网格容器阵列 308 利用不同的主机 211 并含有不同的但类似的网格对象阵列 307，它由容器 303-305 内的对象 9-对象 16 构成。主机 211 能在地理上与主机 210 分离。然而，优选地，多个主机 211 位于一个本地化网络（例如一个组织）之内，而多个主机 210 类似地位于它们自己的本地化网络内。如前所述，每个网格容器阵列需要它自己的 TAGSS 微调度器。所以，网格容器阵列 308 包括 TAGSS 微调度器 501。网格容器阵列 307 和网格容器阵列 308 构成组合网格容器阵列 608，它含有在所有网关处可得到的最大数量容器。图 6 还显示结果造成的组合网格对象阵列 607，它是由网格对象阵列 306 和网格对象阵列 307 构成的。组合网格容器阵列 608 和组合网格对象阵列 607 跨过多个域（例如 210 和 211），这些域可有基本上不同的计算能力。请注意，在语义学上或在组合网格对象阵列的 API 中没有区别，组合网格对象阵列跨越不只一个极小调度域这一事实对客户进程 215 是透明的。然后，下列代码例子使用指定 0（零）容器的方法 Grid.getContainerArray()的方法造成一个请求网络对象中所有容器的请求。
//connect to the Service Scheduler in 2 different domains
Grid grid1=new Grid("gateway1", 1234);
Grid1.add("gateway2", 1234);
//build a large Container Array 307 using containers from 2 different domains
GridContainerArray bigContainerArray=grid1.getContainerArray(0);
//create an Object Array that spans 2 domains
GridObjectArray bigObjectArray=
    bigContainerArray.createObject("objectType", inputArgs);
//invoke a method on the object array
GridObjectArray bigResult=
    bigObjectArray.invoke("methodName", methodArgs);

代码段 12: 使用组合网格容器

图 6 显示在一个示例多群集环境中使用上述代码构建的对象。图 7 包括网格门户 700 用于提高网关 600、601 之间的通信效率。应该指出，上述例子引入一个二层调度分层结构。利用本发明可得到多层分层结构，它利用中间进程作为到服务调度器的代理并构造组合网格对象阵列。例如，图 8 显示一个外包组织 800，它包括一个网格门户，多个客户进程 215、216 以及多个调度器 200、201，调度器 200、201 与其他网格门户 700、702 相连，网格门户 700、702 访问多个网格容器阵列 307-309。在图 8 所示系统中，组合网格容器阵列 307-309 和组合网格对象阵列 607 生长为包括外包组织 800 控制下的所有对象和容器。

网格门户被配置成从若干个单个极小服务域收集容器。再有，因为该网格门户是对服务调度器 200、201 透明的代理，本发明能级联任何数量网格门户，造成深度分层结构。对于如何在一个网格环境中布置 TAGSS 体系结构，存在若干可能的配置，而上述建议是最简单的配置之一。为简化讨论，我们考虑用于一个组织的两个基本角色：它可以是服务提供者或服务外包者。我们的基本配置表明一个网格门户被配置成用于调整全部进入和外出服务请求流。这使得一个基础设置可能审查全部请求。外包组织的
网格门户在指定的网格凭证下运行，它利用这一凭证从服务提供者组织的
网格门户获取资源。

在一个实施例中，TAGSS 进程是，由系统管理员启动和配置服务调度器 200、201 和网格门户 700-702。另一个实现是根据要求代表单个客户机（用户或组织）启动这些进程。例如，在图 8 中，一个外包组织能向启动所请求的网格门户和服务调度器的那些外包组织提交作业请求（与应用代
码一起）。以这种方式，即使该服务提供者组织没有安装 TAGSS 应用，
该外包组织也能获取资源。利用这一途径，代表单个客户机创建了一个一
组新的 TAGSS 进程。

TAGSS 调度器能使用 Globus 2.0 工具箱布署在上述网格环境中。Globus 工具箱受到本说明开头描述的基于作业的体系结构的强烈影响，而
且基本上只支持作业提交模式。但是，如前文指出的那样，作业提交模式
能把 TAGSS 软件进程作为作业来启动。只是这些“作业”是“寿命很长的”，
而且它们又代表若干客户机在服务请求级调度任务。

换言之，本发明提供一个用于处理客户机请求的计算机系统。该系统
包括与网格容器阵列相连的网格服务调度器。每个网格容器阵列包括驻留
在不同主机中的持久性容器（如服务容器）以及微调度器。每个容器包括许
多对象。在构成单个网格容器阵列的那些容器内的那些对象构成一个网
格对象阵列。网格服务调度器透明地把一个客户机请求分割成多个任务并
对每个微调度器赋予任务组。每个微调度器把从网格服务调度器接收的任
务组中的各单个任务赋予它们相应的网格对象阵列内的对象。

该系统能包括额外的调度器分层结构级别，并包括网关，每个网关连
接到不同的网格服务调度器。门户连接到网关，而且门户沿网关把客户机
请求传送到网格服务调度器。

每个容器阵列驻留在一个局域网中，从而使一个网格容器阵列内的对
象之间的通信成为局域通信。网格服务调度器以透明的方式分割客户机请
求，使得客户机不知道对客户机请求的分割。容器是持久性服务容器，用
于在跨越多个客户机请求的一个时间段内处理多个客户机的多个客户机请
本发明还提供一种方法，用于在主机计算机网络上处理客户机请求，该方法包括：在上机中创建持久性容器，在容器中创建对象，把容器分组构成网格容器阵列，把在单个网格容器阵列内的容器内的对象分组构成网格对象阵列，为每个网格容器阵列创建一个微调度器，把每个客户机请求分割成多个任务，以及把任务分组赋予权调度器，其中微调度器把各个任务赋予对象。当微调度器回送完成后的任务组时本发明赋予这些微调度器其他的任务组。该方法还能包括通过网关向多个网格服务调度器传送客户机请求。

这样，上文描述的发明使用网格服务调度器把一个客户机请求分割成许多任务。该系统通过一个网格对象阵列中的多个对象执行所述任务，这些对象在构成一个网格容器阵列的那些容器之内，而该网格容器阵列由一个微调度器控制。

本发明提供一个关注拓扑网格服务调度器（TAGSS），它是用于网格的请求级调度的新的体系结构。这一体系结构是关注拓扑结构的，因为任务调度基础设施被设计成在由不同能力的计算主机和网络构成的多样计算环境中操作。在网格计算中这种多样的计算环境是常见的，这与群集计算中通常所用的均一环境成为对照。因此，在群集计算中使用的调度技术不适用于网格计算。TAGSS 的另一重要特性在于它基于面向服务的体系结构。这与传统的群集调度成为鲜明对照，因为传统的群集调度器是基于面向作业的体系结构。最近引入的开放网格服务体系结构（OGSA）把基于服务的体系结构带入网格计算世界，同时由它带来与网格中调度服务请求的任务有关的新挑战。上述公开的内容讨论了这些挑战，并把这些挑战与 TAGSS 体系结构中的特性关联起来。

尽管已经利用优选实施例描述了本发明，但本领域技术人员将会理解，本发明能在所附权利要求的精神和范围内加以修改来实现。
图6A
图7A
图8A
图8B

图8A

图8B

图8C