[19] 中华人民共和国国家知识产权局

[51] Int. Cl.
G06F 17/30 (2006.01)

[12] 发明专利说明书

专利号 ZL 02818424.6

[45] 授权公告日 2007 年 3 月 28 日

[11] 授权公告号 CN 1307580C

[22] 申请日 2002.9.20 [21] 申请号 02818424.6
[30] 优先权
[32] 2001. 9.26 [33] US [31] 60/324,578
[86] 国际申请 PCT/US2002/031928 2002.9.20
[87] 国际公布 WO2003/027909 英 2003.4.3
[85] 进入国家阶段日期 2004.3.19
[73] 专利权人 EMC 公司
地址 美国麻萨诸塞州
[72] 发明人 马克·萨克 理查德·鲁夫
库尔特·埃弗森
[56] 参考文献
US5857203A 1999.1.5
US6108759A 2000.8.22
US6088694A 2000.7.11
审查员 姚梦琦

[74] 专利代理机构 中国专利代理(香港)有限公司
代理人 刘 杰 王 勇

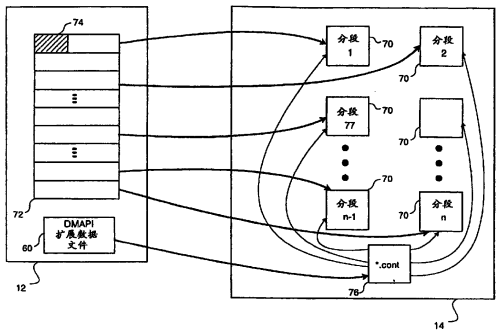
权利要求书 1 页 说明书 18 页 附图 8 页

[54] 发明名称

大文件的有效管理

[57] 摘要

一种对象管理系统，包括一个配置成保存一个对象(72)的计算机系统，其中所述对象可以是一个文件。 计算机系统被配置成将对象(72)划分成分段(70)，选择分段，并且把选择的分段提供给一个存储设备(14)，以便加以保存。 在这里基于是否修改过分段来确定哪些分段需要迁移到设备，由此选择所述分段。 系统则是通过检查一个与对象相关联的数据对象来对此加以判定的。 所述计算机系统可以被配置成从存储设备中检索一个选择的分段。



1．一种包含一个计算机系统的对象管理系统，其中计算机系统包括：

一个主存储设备，被配置成保存一个对象，以及

一个处理器，耦合到存储器并且被配置成将所述对象划分成分段，选择所述分段的一个子集以便迁移到辅助存储设备，所述选择是至少部分地基于确定包括所述子集的分段已经被修改而进行的，并且把选择的分段的子集提供给所述辅助存储设备，以便作为单独的保存对象来加以保存。

2．如权利要求 1 所述的系统，还包括一个与对象相关联的数据对象，其中数据对象包含关于是否修改过分段的信息。

3．如权利要求 1 所述的系统，其中所述对象是文件。

4．如权利要求 1 所述的系统，其中计算机系统还被配置成从存储设备中检索一个选择的分段。

5．如权利要求 4 所述的系统，其中计算机系统还被配置成接收一个数据存取请求，并且使用该数据存取请求来选择要从存储设备中检索的分段。

6．如权利要求 5 所述的系统，还包括一个与对象相关联的数据对象，其中所述数据对象包含关于分段是否存在于计算机系统中的信息。

7．如权利要求 6 所述的系统，其中如果数据对象指示分段不在计算机系统中，则计算机系统选择要从存储设备中检索的分段。

8．一种用于管理系统中的对象的方法，所述系统包括一个主存储设备和一个被配置成保存一个对象的计算机系统，所述方法包括：将对象划分成分段，选择所述分段的一个子集以便迁移到辅助存储设备，所述选择是至少部分地基于确定包括所述子集的分段已经被修改而进行的，以及将选择的分段的子集提供给一个辅助存储设备，以便加以保存。

9．如权利要求 8 所述的方法，还包括接收一个数据存取请求，并且其中选择要从存储设备中检索的分段包括使用所述数据存取请求来确定要选择哪个分段。

大文件的有效管理

相关申请的交叉引用

本申请要求针对 2001 年 9 月 26 日提交的、名称是“SYSTEMS & METHOD FOR MANAGING TABLESPACES AND DATABASES”的共同未决美国专利申请 60/324,578(律师案卷号为 OTG-001PROV)而享有优先权，该申请在此引入用于各种用途。

技术领域

本发明一般涉及一种用于在对象管理系统中有效管理大文件的方法、产品和设备。更为特殊的是，本发明涉及的是在对象管理系统中对大文件进行有效的存储和检索，其中对象管理系统可以包括具有以下不同特性的存储器的各种组合，所述特性是例如速度、成本、大小、可用性、可存取性以及远程性。

背景技术

本发明涉及的是在对象管理系统中对大文件进行管理。现有系统旨在处理那些规模极大的表空间，其中这种现有系统可以是与 Oracle 数据库结合使用的系统。一个表空间可以由多个数据文件组成，这些数据文件可以由操作系统存取和操作。“数据文件”是指一个可以拷贝、移动或在其他方面作为连续信息对待的文件，不管所述数据是否在物理上以连续方式保存在存储介质上。

本地存储容量也许不足以保存与一个或多个数据库相关联的整个表空间。通过购买充足的快速存取存储介质（例如硬盘或光学介质）来提供保存整个数据库的足够容量是非常浪费的，在没有必要以相对较快的速度存取所有数据的时候尤其如此。已经实施这样一种常规数据库系统，其中可以将数据“迁移”到价格较低的介质，并且可以在需要的时候才从介质中检索数据。然而，现有系统和方法并未有效管理那些将要迁移、迁移之后以及检索之后的数据。这些系统要忍受等待时间并具有很高的带宽要求，此外还需要很长的备份时间并具有高昂的成本，但却未必能够产生“时间点（point in time）”快照。这些问题不仅限于数据库和表空间。而是，这些问题同样存在于保存诸

如多媒体之类的大文件的系统。

如这里所公开的那样，存在一种更为有效的方法，该方法减少了等待时间和带宽要求，降低了成本，并且只需要较小的备份窗口，由此更有效地管理数据。尽管这里的公开是在数据库和表空间的环境中描述问题和发明的，但是本发明也可应用于任何使用迁移的数据管理系统，尤其是在所述系统管理诸如音频或视频这类大数据文件的时候。

发明内容

因此，简要地说，本发明提供了一种用于在对象管理系统中有效管理大文件的方法、产品和设备。在本发明的一个实施例中，对象管理系统包括一个配置成保存对象的计算机系统，其中所述对象可以是一个文件。所述计算机系统被配置成将对象分割成分段并且选择分段，此外所述计算机系统还把选择的分段提供给一个存储设备，以便加以保存。

其中，根据是否修改过分段来确定要将哪些分段迁移到存储设备，由此对分段进行选择。而系统则是通过检查一个与对象相关联的数据对象来对此加以确定的。

计算机系统可以配置成从存储设备中检索一个选择的分段。所述系统接收一个数据存取请求并且使用数据存取请求来选择一个分段，此外还使用了数据对象来确定选择的分段是否存在于计算机系统中，如果选择的分段不在计算机系统上，则从存储设备中检索这个选择的分段。

对本领域技术人员来说，本发明的优点和进一步细节将从以下结合附图所进行的详细描述中变得清楚。

附图说明

本发明将通过以下结合附图所进行的详细描述而变得易于理解，其中相同的参考数字表示的是相同的结构单元，并且其中：

图 1 是具有一个前端和一个后端的对象管理系统的一个示意图；

图 2 是附加了大容量存储设备的计算机系统的示意图；

图 3 是 i 节点（inode）与保存了 DMAPI 信息的文件之间的逻辑关系的框图；

图 4 是 i 节点与保存了 DMAPI 信息的扩展块之间的逻辑关系的框

图；

图 5 是文件、DMAPI 扩展数据文件、容器文件以及后端保存的分段的框图。

图 6 是显示对象管理系统中使用的守护程序的方框图；

图 7 是一个显示分段的版本控制（versioning）的方框图；

图 8 是对迁移处理进行描述的流程图；

图 9 是描述更新文件状态来确保一致性的流程图；

图 10 是描述清除处理的流程图；

图 11 是描述事件处理和重新提交（restaging）数据的流程图；

图 12 是显示重新提交分段的方框图；

图 13 是一个描述有效搜索迁移或清除候选者以及获取候选者路径名的流程图；

图 14 是显示了若干种搜索迁移或清除候选者的方法的方框图；

图 15 是描述了使用容器文件而使来自一台计算机的文件能在另一台计算机上使用的流程图；以及

图 16 是图 15 中的计算机的框图。

具体实施方式

在这里提供了关于实施例的详细描述。虽然本发明是结合实施例来描述的，但是应该理解，本发明不局限于任何一个实施例。相反，本发明的范围只受附加权利要求限制，并且本发明包括多种替换、修改和等价物。出于例证目的，在以下描述中阐述了很多具体细节，以便提供关于本发明的全面理解。但是本发明可以依照权利要求而在没有若干或全部这些具体细节的情况下实施。为了清楚起见，在这里并未详细描述那些与本发明有关但在本技术领域已知的技术材料，以免不必要地造成本发明不清楚。

应该了解的是，本发明可以借助多种方式实施，其中包括作为一个进程、一个仪器、一个系统、一个设备、一种方法或一种计算机可读介质，例如计算机可读的存储介质或是计算机网络，其中程序指令是通过光/电通信链路发送的。

概述

如图 1 所示，在一个实施例中，对象管理系统 10 包括一个主要或本地计算机系统 12 以及一个存储系统 14，它可以是一个辅助或远程计

算机系统。主计算机系统 12 称为前端 12，它向用户（没有显示）提供主应用程序和数据存储服务。在正常操作中，主计算机系统 12 在其 CPU 20 上运行应用程序并在其本地文件系统 16 上提供本地数据存取，所述文件系统通常包括高速设备，例如各自处于一个 RAID 阵列中的硬盘驱动器 18 或是其他结构。存储系统 14 也称为后端 14，它可以是一个与各种大容量存储设备相连的通用 CPU 22，其中所述大容量存储设备可以组织到文件系统 26 中，也可以是一个专用存储装置。在后端 14 中使用的大容量存储设备可以是硬盘驱动器 24，也可以是更廉价、更慢或专用于归档目的的其他设备。例如，可以使用光盘 28、磁带驱动器 30 以及磁光驱动器。并且可以将后端 14 用作那些较少存取的数据的廉价近线（near-line）存储器，也可将其用于备份前端 12 的数据。

后端 14 可以与前端 12 处于相同位置，也可以位于远端。数据传送是经由 NFS、AFS、FTP 或其他方法来完成的。前端 12 与后端 14 可以使用一对一、多对一或多对多的关系连接。通常，前端 12 上的文件系统 16 与后端 14 是一对一的关系，但是在前端 12 上可能存在几个文件系统 16，其中每一个系统都映射到一个不同后端 14。例如，在将各个文件系统 16 用于归属公司中不同群组的一个不同应用程序的时候，可以使用这种结构。因此，会计部门的数据可以转向一个后端 14 系统，而涉及工资单的数据则转向另一个后端 14。这些后端系统 14 可以位于不同的远端位置。

在操作中，迁移守护进程将来自前端 12 的数据迁移（拷贝）到后端 14；也就是将来自前端 12 的数据拷贝到后端 14。这些数据划分成写入后端 14 的分段。一旦迁移了一个文件，则在前端 12 将其标记为已迁移。这个操作是通过设定一个指示文件已经迁移的比特或者其他标志来完成的。清除守护进程寻找那些数据不再为前端所需要的已迁移文件，并且清除那些不再需要的数据。事件守护进程捕获数据存取事件，如果在前端 12 上需要数据，那么它会将数据从后端 14 重新提交到前端 12。事件守护进程还被配置成移动所需要的文件部分，从而仅仅重新提交文件的一部分。这样的优点是减少正在传送的数据量，以便减少带宽需要并缩短等待时间。

迁移守护进程周期性地检查那些需要迁移的文件（确切地说，应该是文件的一些部分）。如果它发现一个标记为脏（dirty）的重新提

交的文件（也就是说，在所述文件最后一次迁移到后端 14 以来已经对其执行过写入），那么它将会把该文件迁移到后端 14。迁移守护进程则配置成只写入那些经过修改的分段。而未修改的分段则不必写入，由此减少了所要传送的数据量。在后端 14，对经过修改的分段进行版本控制，并且将其写入大容量存储设备。这些分段可以改写当前存在的分段，但是保存较早版本则允许创建时间点快照，以便在任何给定时间都能保存系统状态。涉及分段位置的信息和其他元数据写入到位于后端 14 的快速存储器上的容器文件。在这种结构中，备份是依靠系统操作来创建的，并且不需要进行单独的备份，因为容器文件包含了将数据还原到前端 12 所需要的全部信息。

除了设定分段大小和其他结构以及环境信息之外，这些操作是在不需要在应用程序用户部分上的特别介入的情况下以透明方式执行的。

此外还能使用这种系统而使来自一个计算机系统的数据可以应用于另一个计算机系统，同时不需要拷贝所有数据。为了实现这个目的，其中将第二计算机系统配置成在后端 14 上使用第一计算机的容器文件，以便在其本地文件系统上创建自己的文件。当第二计算机系统将自己的数据写入后端 14 时，第二计算机系统会在设置过程中或者按照需要来创建自己的容器文件。在第二计算机的操作过程中，如果来自后端 14 的文件的必要分段不在前端 12 上，那么数据存取会导致重新提交这些分段。如果任何一个计算机系统修改了数据，那么它会在迁移数据的时候将自己的分段写入后端 14 并且更新自己的容器文件。这样一来，每一个计算机系统都具有了自己的文件版本。

此外还可以对系统进行配置，以便通过将文件名、i 节点编号以及扩展属性存入单个文件来搜索迁移和清除候选者。系统可以搜索这个文件来快速识别迁移或清除候选者，而不用检查 i 节点以及可能还有名称空间中各个文件的扩展块。使用 i 节点编号所进行的反向查找被用于确定路径名。这种方法允许系统仅仅查找那些用于迁移的 i 节点和清除候选者，从而减少需要的时间。

所述系统可用于管理那些由数据库使用的大的表空间，但在大文件管理方面也具有应用性，在只对文件一部分进行数据存取的情况下更是如此。

详细说明

在一个实施例中，主要和辅助计算机系统各自包含了一个在通用计算机上执行的软件程序，所述通用计算机可以是一个运行 UNIX 的工作站，但是也可以使用其他计算机系统，例如基于 Intel 公司 Pentium 处理器并运行视窗或 Linux 操作系统的 PC。此外还可以使用实现了计算机系统的其他装置，例如将指令烧入诸如专用集成电路（ASIC）或现场可编程门阵列（FPGA）这类芯片的专用硬连线系统。辅助计算机系统则用作一个存储系统，它可以是以凭借网络附加存储（NAS）或存储区域网络（SAN）结构中连接的专用存储设备的形式来实现，例如 EMC、Network Appliance、StorageTek 和 ADIC 制造的存储设备。如图 2 所示，计算机系统可以具有任意数量的下列部件：中央处理器 41、存储器 42、显示器 44、键盘 46、大容量存储设备 48、网络接口 50 以及其他输入/输出设备 52。根据惯例，存储器 42 和大容量存储设备 48 可用来保存程序指令和数据。计算机系统 40 还可以具有一个以上的中央处理单元 41，例如基于多处理器 Pentium 的系统或是 Sun SPARC 工作站。大容量存储设备 48 可以包括使用了 RAID、光纤通道或其他接口的一个或多个硬盘驱动器、光驱动器、磁带驱动器、磁光驱动器、DAT、CD 驱动器、DVD 驱动器或用于存储数据的其他设备。大容量存储设备 18 可以组成为文件系统，其中可以使用一组以上的大容量存储设备 18。

文件系统是对象存储的一种类型，它可以保存文件（在这种情况下，对象 ID 即为它的路径名）或对象（在这种情况下，对象是由不同于文件系统名的某个 ID 来引用的）。一个对象存储具有以下组成部分：一种识别对象的方法，允许操作对象的接口（API、文件系统原语等等）以及用于对象的存储器。文件可以视为是一种对象。这里公开的原理同等地适用于对象和对象存储以及文件和文件系统。就对象存储而言，用于描述在后端何处发现对象的扩展属性是一个对象 ID 而不是文件名。

主计算机系统则充当前端 12，它通常具有以硬盘驱动器 18 为形式的高速存储器，以便快速存取数据。存储系统则充当后端 14，它具有某些高速存储器以及比较慢但却更经济的设备，例如磁带备份 30 和光驱动器 28。其他归档设备可以附属于存储系统 14。并且使用后端 14

来保存那些很少存取的数据，由此减少了对于昂贵的快速存储空间的需要。

后端 14 可以作为主计算机系统 12 的一部分来实施，以使后端 14 的存储设备附属于主计算机系统 12。尽管所公开的实施例描述了分别用于前端 12 和后端 14 的独立的主要和辅助计算机系统，但是独立的辅助计算机系统并不是实施本发明所必需的。

这里描述的主计算机系统符合如下的系统管理规范：由开放组织（先前的 X/Open）出版，ISBN 号为 UK ISBN 1-85912-190-X 并且文档编号为 C429 的 Data Storage Management（XDSM）API，其公开在此引入作为参考。该文档可以从http://www.opengroup.org在线获取，其中阐述了用于那些依从数据管理 API（DMAPI）的应用程序的规范。如果还有其他操作系统支持这里描述的数据管理功能，那么也可以使用这些操作系统。

在 UNIX 或 LINUX 系统中，操作系统使用 i 节点来保存标准的文件系统信息，例如大小、所有权、日期和权限。如 XDSM 规范所规定的那样，i 节点还用于保存一个与 DMAPI 扩展数据的链接，其中包含了扩展属性/元数据以及区域信息。在一个实施例中，DMAPI 扩展数据优选保存在一个名为.DMATTR 的 DMAPI 扩展数据文件 60 中。DMAPI 扩展数据文件 60 可以是前端 12 或后端 14 保存的单个文件，也可以跨越前端 12 或者后端 14 保存的若干文件。在前端 12 中的存储器提供了更快速存取，如同在存储器中进行高速缓存。图 3 显示了文件系统 16 中的 i 节点 62 与 DMAPI 扩展数据文件 60 之间的逻辑关系。应该理解的是，尽管在这里将这些单元显示为集中在一起，但这未必反映了其在磁盘上的物理位置。如图 3 所示，DMAPI 扩展数据文件 60 可以作为扩展属性和区域的表格来实现，并且是以 i 节点的编号作为索引的。如图 4 所示，另一种布局是使每一个 i 节点 62 都指向自己的扩展块 64。通过在.DMATTR 文件 60 中保存 DMAPI 信息，可以在不必读取其 i 节点 62 和 DMAPI 扩展块 64 的情况下读取文件的 DMAPI 信息，这在核对多个文件的时候非常有利。此外该信息也可作为文件系统 16 的一部分而加以保存，这与 i 节点信息的保存方式相似。

扩展属性被用于记录诸如是否迁移文件和是否清除文件这类信息，以及以目录路径形式的与后端 14 上的容器文件 76 的链接、对象

ID 等等。区域信息包含了关于文件分段的信息，例如本地是否存在分段中的数据，是否数据为脏并且需要迁移，此外区域信息还包含了相应的本地文件中的数据位置。在这里可以使用标志并且可以用偏移和长度信息来指示本地文件中的数据位置。区域信息可以包括一个字段，它规定了当在所映射的文件区域中存取或改变数据时产生的事件集合；例如 dm_region_read，dm_region_write，dm_region_truncate。此外还可以设定所述标志，以便指示数据是固有的还是脏的，以及是否需要重新迁移。如果分段是被动态地定大小的，那么也可以保持分段大小信息。在存取数据的时候，由于使用了区域信息来记录数据位置，因此可以增加映射一个文件的区域数目。

如图 5 所示，在将文件 72 迁移到后端 14 的时候，所述文件划分成多个分段 70。分段 70 的大小可以是固定的，也可以是动态变化的。固定分段的分段大小可以由文件系统中保存的结构或环境变量来限定。然而也可以根据文件系统特征、文件大小以及数据存取模式来选择分段大小。分段大小不应该太大，因为较大的分段大小会产生更多的等待时间和较高的带宽使用，并且在后端 14 中将会占用更多空间来进行版本控制。另一方面，太小的分段大小会因为存在更多需要处理的分段而增加内务处理的需要。对大型数据库而言，可以使用大小为 100 兆字节（MB）的分段大小。因此，举例来说，一个 2GB 的文件划分成 20 个用于后端 14 的 100MB 分段。在一种实施方式中，所述系统被用于一个 Oracle 数据库，它具有 9840 个用于后端 14 的磁带和 16 个大小为 128MB 的分段。某些影响到分段大小的因素包括：需要数据的应用程序的存取模式、后端 14 与前端 12 之间的数据传送速度、在后端 14 中用于数据传送的归档介质速度以及后端 14 介质等待时间。具有较慢定位时间的后端介质（例如磁带 30）有利于较大的分段大小。

通过对文件 72 进行分割，所述系统能在较小的部分 70 中移动数据。如果一个应用请求从 2GB 的文件中只存取一小部分数据，那么，即使这部分数据不在本地文件系统上，也只需要对包含该数据的某个 100MB 分段进行检索，进而对所述数据进行存取。这样将会减少等待时间，因为移动数据需要的时间要小得多，此外还因为移动较少量数据而减少了带宽使用。同样，如果随后仅仅修改一小部分数据文件 72，那么只有包含了经过修改的数据部分的分段 70 才需要迁移到后端

14。

如图 6 所示，其中对主计算机系统进行配置，以便在操作中包含三个守护进程，即事件守护进程 *dxdbmigd* 80、迁移守护进程 *dxdbmigwatch* 82 以及清除守护进程 *dxdbpurgewatch* 84。但是也可以使用任意数量的守护进程，其中只包括一个线程化的守护进程，此外还可以使用守护进程之外的其他方法。例如，视窗系统可以使用系统代理和服务。事件守护进程 80 捕获的是访问文件数据的时间。如果所请求的数据不在前端 12 的本地文件系统 16 上，那么事件守护进程 80 会把来自后端 14 的数据拷贝到本地文件系统 16，然后进行数据存取。迁移守护进程 82 寻找那些其数据需要从前端 12 迁移到后端 14 的文件。一旦它识别了其数据需要迁移的文件，则它会将数据从前端 12 拷贝到后端 14。清除守护进程 84 寻找其数据不再为本地文件系统 16 所需要的文件，并且清除不需要的数据。由于这里描述的有效搜索迁移和清除候选者允许以很低费用来查找迁移和清除候选者，从而可以将迁移和清除守护进程 82 和 84 配置成以例如 5 分钟的时间间隔来周期性地运行。

参考图 8 中的步骤 100，迁移守护进程 82（或者线程化的单个守护进程）使用.DMATTR 文件 60 中保存的属性信息来确定是否需要迁移文件，但是如这里所述，所述进程也可以使用扩展块 64 中保存的属性信息来确定是否需要迁移文件。如果以前没有迁移过文件 72，那么该文件将会划分成若干个分段 70，这些分段可以具有固定或动态确定的大小。分段大小可以基于如下因素而被动态确定，例如数据存取频率，每次存取时读取的连续数据的量以及其他存取模式信息。迁移守护进程 82 还可以派生出处理不同任务的其他进程。并且在迁移之前的处理、数据迁移期间或是检查文件是否在迁移中发生变化的时候（如果文件变化了，则舍弃当前迁移并且尝试下一次迁移），可以使用信号量、文件锁和其他方法来保护文件。

在步骤 102，迁移进程以独占形式锁定文件，在步骤 104，该进程读取属性和区域信息，并且在步骤 106 中确定是否需要迁移文件。如果不需要迁移，则在步骤 108 中撤销锁定。否则在步骤 110 中使用区域信息、文件大小和分段大小来确定需要迁移的文件部分。在步骤 112，如果以前迁移过文件，那么存在一个容器文件 76，其中包含了描

述先前迁移的信息，并且通过读取这个文件来确定文件的新版本号。如果从未迁移过文件，则将版本号假设为 1。只有新数据或是在上次迁移之后发生过变化的数据才会迁移到后端 14。在步骤 114，数据是依照分段大小来分割的，其中分段大小既可以是固定的也可以是动态的。在步骤 116 中将会撤销锁定，以使其他使用了所述文件的进程能在迁移的时候继续进行。

在步骤 120，在并未改变前端 12 上的文件存取时间的情况下，需要迁移的分段 70 将被拷贝到后端 14。如果迁移过程中出错，则在步骤 124 中将会在执行下一次迁移时重新尝试进行迁移。如果迁移成功，则在步骤 126 中再次锁定文件，此外还会在步骤 128 中再次读取文件属性和区域信息，并且使用这些信息来判定是否在数据移动中改变了文件。如果改变了文件，则在步骤 132 中撤销锁定，由于数据可能不一致，因此在步骤 124，所述迁移失败并且稍后将会再次尝试。如果在数据传送中并未改变文件，则在步骤 134 中通过更新文件信息来显示成功迁移了文件。并且还会更新扩展属性（例如 DMAPI/XDSM）和区域信息，而包括路径名、大小、所有者、权限以及其他文件属性在内的文件相关信息则写入到一个与数据文件相关联的名为*.cont 文件的容器文件 76。此外，写入容器文件 76 的信息还包含了与后端 14 保存的分段有关的信息，其中包括版本控制信息。

文件状态必须按照指定顺序更新，以便确保一致性。在图 9 中，在步骤 140，通过设定属性来显示并未清除文件，此外还设定了文件分段大小（如果必要）和后端 14 的文件位置，并且在步骤 142 中将信息直写到文件系统 16 中。由此确保能在发生崩溃的情况下在文件中存在足够信息，从而可以通过前滚到新状态或是后退到先前状态来存取文件。在步骤 144，新版本的*.cont 文件 76 是以一种确保*.cont 文件 76 包含文件旧状态和新状态的方式写入的。在步骤 146，区域信息直写入 DMAPI/XDSM 接口，由此显示所有数据文件都已成功迁移到后端 14，然后在步骤 148 中对.DMATTR 文件 60 进行同步（将没有完成的信息写入磁盘）。如果失败了，则旧的区域信息仍然有效，因此可以在以后的迁移尝试中继续进行未来的迁移。在步骤 150 中对 DMAPI/XDSM 属性进行更新，以便将文件标记为正在迁移。这其中包括对元数据进行设定，以便显示数据在后端 14 中的存储位置和迁移时间，并且将文

件标记成正在迁移。而元数据则以这样一种方式写入，即其中任何时间的故障都会使文件处于一致状态。

这样能够防止发生故障时引发的数据损坏，并且防止文件处于一种允许存取那些有可能无效的文件数据的状态。如果必要的话，区域信息中保存的状态、文件元数据以及后端版本文件（*.cont 文件）足以在清除之后还原所述文件，但也可以在本地管理的磁盘丢失或无意移动了文件的情况下恢复所述文件。

各个数据文件 72 可以具有一个相关的*.cont 文件 76，但也可以将用于多个数据文件的*.cont 信息存入单个容器文件，所述文件可以用数据文件名或诸如对象 ID 这样的其他标识符作为索引。如果先前迁移过文件，则对.DMATTR 文件 60 中的区域信息进行检查，以便确定哪些分段是脏的；也就是确定在上次迁移以后修改过哪些分段。经过修改的分段拷贝至后端 14 并且可以对其进行版本控制，使之不会改变分段的现有拷贝。如图 7 所示，*.cont 文件 76 记录了所述版本。

举例来说，如果*.cont 文件 76 指示后端 14 已经存在一个分段版本 54，则在没有改变或改写版本 54 的情况下将这个经过修改的分段作为版本 55 来写入，并且通过更新*.cont 文件 76 来反映这个操作。实际上，*.cont 文件记录了文件增量。由于所述文件存在于任何时间点，因此它具有恢复相关文件所需要的信息，并且由于*.cont 文件 76 在任何特定日期和时间都有效保持了快照，因此可能确定数据文件 72 看起来像什么。在结束迁移之后（例如将迁移的数据成功写入磁带的时候），关于新迁移分段的信息保存在容器文件 76 中。此外还会通过更新.DMATTR 文件 60 来指示已经迁移了所述文件，如果第一次迁移文件，则会更新与后端 14 中的容器文件的链接。如果迁移的是经过修改的分段，则对区域信息进行更新，从而显示所述分段不再是脏的，并且由此不再需要迁移所述分段。因此，在前端 12 的文件与其在后端 14 上的分段之间存在链接，该链接保存在.DMATTR 文件 60 和*.cont 文件 76 中。

为了描述迁移，如果在前端 12 上存在一个以前没有迁移过的称作 ABC 的数据文件，则在后端 14 为其选择一个唯一路径名，假设为 123。ABC 文件划分成分段大小由对象管理系统结构确定的分段。这些分段拷贝至后端 14，以便作为指示唯一路径名、分段编号和版本号的各个文

件，例如 123.partition1.version1，123.partition2.version1，直到 123.partitionN.version1。然后则写入 a123.cont 文件，该文件描述的是迁移中已经完成什么，其中包括本地文件系统 16 上的文件名、分段名以及分段版本。附加信息是在.DMATTR 文件 60 中写入本地文件系统 16 或文件系统中的其他位置的。尽管该信息也可位于其他位置，但是将其置于本地文件系统 16 上将会加速存取。此外还可以在前端 12 的存储器中高速缓存.DMATTR 文件 60，但这需要频繁同步高速缓存的文件以及磁盘上的文件，以便在发生崩溃的时候保持一致。所写入的属性包括一个已将文件 ABC 迁移至文件 123 的指示，指示迁移时间和文件所迁移位置的时戳，以及一个已经成功将文件迁移至后端 14 的指示。

参考图 10，在步骤 200，清除守护进程 84（或线程化的守护进程）使用.DMATTR 文件 60 中保存的属性信息来识别那些已经迁移（不再需要迁移）但却并未清除的文件。清除守护进程 84 可以在确定本地文件系统 16 需要自由空间之后再进行这个操作，并且可以在有足够自由空间可用的时候停止清除。实际上，在步骤 202，清除候选者是根据清除策略而被排序的，其中所述策略可以由用户进行配置。并且可以将 LRU（最近最少使用）用作清除策略，然而也可以基于文件系统 16 中保存的数据特征、使用文件系统 16 上的数据的应用以及数据存取模式来选择其他策略。这些策略可以在文件级或分段级上使用。清除守护进程 84 可以使用扩展属性中的信息来确定文件中的分段的最后存取时间和存取频率等等。基于清除策略，清除守护进程 84 将分段标识为将要清除。例如，清除守护进程 84 可以使用一个 LRU 算法来对文件排序，然后在被选择进行清除的文件范围内选择具有早于某个日期的最后访问日期的分段。

在这里可以使用文件锁、信号量或其它方法来防止数据丢失或损坏。在步骤 204，在清除进程编辑了一个清除候选者列表，并且在选择了一个用于清除的文件之后，所述文件将被锁定。在步骤 206 中将会读取属性和区域信息，并且在步骤 208 中对其进行检查，以便了解在编辑了文件之后是否存取过文件；即它是否仍是一个清除候选者。如果不是的话，则在步骤 210 中不清除文件并且撤销锁定。如果文件仍是一个清除候选者，则在步骤 212 中设定文件区域信息，以便显示已

经清除了整个文件。在步骤 214，清除守护进程 84 从文件中消除对应于选定分段的本地数据。在一个依从 DMAPI 的系统中，清除守护进程 84 使用 dm_punch_hole()来消除本地数据。根据操作系统性能，可以从一个相对文件末端的偏移那里截断本地文件 72，这对大多数操作系统而言都是很常见的，或者如 ATX 所规定的那样，也可以在文件中间凿孔。将要消除的数据有可能位于文件开端，在这种情况下将会留下一个存根 74（如图 5 所示）。

在消除了本地数据之后，在步骤 216 中将会更新属性和区域信息，以便反映数据的清除。区域信息表示实际清除的数据不会在本地文件系统 16 上出现，如果清除了整个文件（除了存根 74），那么所述属性表示已经清除了文件。如果残留了文件的任何部分，则所述文件继续成为一个清除候选者。在更新了文件元数据之后，在步骤 218 中将会消除文件锁定。由此在系统崩溃或是发生其他故障的情况下保护文件。由于并未将其标记为已清除，因此仍将文件视为一个清除候选者，但由于在清除操作前将其标记为已清除，因此在存取文件时将会导致重新提交文件的所有部分。某些数据可能仍是本地数据，但如果在清除操作中出错，则未必能够断定清除了哪些数据。因此，清除进程将所有分段标记成已清除，在结束清除之后，所述进程更新区域信息，以便指示仍旧存在哪些分段。如果元数据更新失败，则文件仍旧显现为清除候选者，尽管它的某些或所有数据可能已从本地文件中清除。所描述的序列被用于防止数据损坏，但只要通过同步区域信息、文件数据删除与文件元数据更新来确保一致性，那么其他方案也是可行的。由此避免文件处于一种可以存取过期数据的不一致状态之中。

文件 72 通常在由应用扫描的文件开端包含了报头和其他频繁使用的信息，并且在清除文件的时候，通过将存根 74 保留在适当位置，可以加快数据存取时间。用户可以基于如下信息来定义存根 74 的长度，例如文件 72 开端是否存在频繁存取的信息以及对文件 72 开端的多少数据进行过存取。举例来说，Oracle 数据库可能需要至少 128KB 的存根大小，这是因为 Oracle 在例如启动时间和 Oracle 存取各个 Oracle 数据文件的时候会频繁访问数据文件中的这个数据。如果存根数据不是固有的，那么 Oracle 将会停止，直到从后端 14 恢复所述数据，这意味着多磁带装配（或者诸如 CD 之类的其他存储介质的装配）。此外

还可以对存根 74 进行版本控制，并且将其作为文件或对象保存在后端 14。

在图 11 中，在步骤 300，事件守护进程 80 或线程化的守护进程捕获数据存取事件并且记录数据存取活动。当发出数据存取请求的时候，在步骤 302，所述处理取决于数据存取究竟是读取还是修改该文件（例如写或者截断）的存取。在步骤 304，如果数据存取是一个读取，则对.DMATTR 文件 60 中的区域信息进行检查，以便确定所请求的数据是否存在于本地文件系统 16 之上；也就是说，是否区域信息显示的是将对应于正在存取的文件部分的分段标记为存在。在步骤 306，如果存在所请求的数据，那么在步骤 308，存取请求传递至文件系统，并且在步骤 324 中通过更新文件属性和区域信息来反映所述存取。如果不存在所请求的数据，那么在步骤 310，事件守护进程 80 或是另一个从事件守护进程 80 那里接收事件的进程将会锁定文件并且检查区域和属性信息，以便在步骤 312 中确定是否需要进行处理。在步骤 314，如果因为先前的事件处理而不需要进行处理，则撤销锁定并为事件产生一个响应，所述响应将会唤醒这个等待结束 I/O 请求的进程。如果需要处理，则在步骤 316 中将文件标记为可清除，并且对元数据进行同步以及确定完成事件处理所需要的文件数据，此外还会确定后端 14 的文件位置，读取*.cont 文件并撤销文件锁定。

在数据移动过程中并未锁定文件，由此允许处理那些本地文件系统 16 固有的数据。在步骤 318，从后端 14 读取必要分段并将其写入前端 12 的本地文件系统 16。此外所述文件仍然是以独占方式锁定的；在步骤 320 中以一种一致方式来更新区域和属性信息，并在步骤 322 中撤销锁定。在步骤 314 中则把一个响应发送到正在等待的进程，以便唤醒所述进程，由此完成它的读取请求。除非再次清除数据，否则针对这个数据的未来存取是不会产生恢复事件的。

根据数据访问模式，也可以使用某些预先的分段检索；也就是说，可以记录关于数据存取模式的信息，并且如果确定对于一个特定分段的存取频繁地引起了对于另一个分段的存取，那么也可以预先检索所述分段。

在步骤 324，当事件守护进程 80 处理了数据存取事件之后，可以对.DMATTR 文件 60 进行更新，以便指示对应于所存取文件部分的一个

或多个分段的存取时间，例如清除策略在分段级使用数据存取信息的时间。所述系统还会更新文件属性，以便指示文件存取时间。

如果存取请求是写入，则在步骤 326 中通过修改区域来更新存取时间，并且将对应于数据的分段标记为脏（由此需要对其进行迁移）。在这种情况下，在步骤 328 中以独占形式锁定文件并对事件进行检查，以便确定是否有必要进行处理。如果没必要处理，则在步骤 330 中撤销锁定并对事件做出响应，终止处理。如果有必要处理，则在步骤 332 中通过更新元数据来指示所述文件是一个清除候选者，并且由于要改变数据，因此还会指示所述文件是一个迁移候选者。此外还会读取文件元数据、区域信息以及*.cont 文件，以便确定文件状态。在步骤 336 中将会确定产生事件的文件区域，如果区域信息指示所述数据不在本地存在，则在步骤 338 撤销锁定并且以一种类似于上述读取过程的方式来移动数据。如果数据是固有的，则不需要从后端 14 移动数据。在步骤 340，在重新提交了数据之后，文件将会再次锁定，并且将会更新文件元数据和区域信息，以便指示需要迁移文件中的某些数据以及改变了文件中的哪些部分。在步骤 342 中则会撤销独占性锁定并且会将一个响应发送到正在等待的进程。

为了描述事件处理，除了长度由结构限定的存根 74 之外，对 N 个分段都被迁移和清除的文件 ABC 来说，针对分段 77 中的数据的数据存取请求是由事件守护进程 80 捕获的。在图 12 中描述了这种情况。所述守护进程判定分段 77 中的数据不在本地文件系统 16 中存在，并且通过检查.DMATTR 文件 60 来确定后端 14 上的相应容器文件是 123.cont。在这里将会发布一个关于相应后端文件 123 的分段 77 的请求。123.cont 可以指示版本 55 是文件 123 的分段 77 的最新版本，由此将 123.partition77.version55 检索到前端 12。在将分段还原到磁盘之后，属性和区域信息将会得到更新。

本系统通过迁移和检索文件段而不是整个文件而避免了大量和耗时的文件传送。举例来说，数据库文件趋向于很大并且以千兆字节来度量。而在迁移和检索过程中来回移动整个文件是不切实际的，在诸如 Oracle 这样的数据库应用可能只访问很少一部分表格（例如行）的时候尤其如此。举例来说，Oracle 不会立刻扫描整个文件。而是，它会部分地扫描一个数据库文件，然后继续扫描另一个文件，依此类推，

直到最终回到第一个文件并扫描更多的数据。而使用全文件检索则会导致系统失效或是更长的检索时间。

为了实现这里描述的文件分割和分段管理，所述系统还被配置成有效搜索迁移、清除候选者并获取其路径名，但这并不是必需的。如图 14 所示，其中存在一种在名称空间中搜索文件名并为每个文件查找 i 节点和扩展属性的方法。在某些结构中，i 节点包括一个指向包含了扩展属性的扩展块的指针，由此需要一个第二查找以及相应的磁盘存取。在图 13 和 14 所描述的一种更有效的方法中，在步骤 350，系统关于文件名、i 节点编号以及扩展属性来搜索 DMAPI 扩展数据文件 60（.DMATTR 文件）。此外还可以将所述系统配置成使用一个不同的文件或是若干个文件。这样一来，在步骤 352，系统可以快速确定哪些文件是迁移或清除候选者，而不必在名称空间中为各个文件查找 i 节点以及可能还有扩展块。为了产生用于一个候选者的路径名，在步骤 354 中使用所述候选者的 i 节点编号来查找其路径名。这可以利用一个在 i 节点和路径名列表中的反向查找来实施。所述列表可以作为表格而被保存在同一个文件.DMATTR 中，也可以保存在单独的文件或文件组中。利用这种结构，系统仅仅是查找那些标识为迁移和清除候选者的文件的 i 节点，而不必检查关于包含了已迁移和清除的那些文件在内的所有文件的信息。由此显著减少了迁移和清除所需要的执行时间以及系统负载。例如，在确定需要迁移哪些文件的过程中，使用这里描述的有效搜索的系统可以在不到一分钟的时间里对带有一百万个文件的文件系统进行检查，以便找出迁移和清除候选者。与先前使用名称空间进行搜索所花费的 20 分钟相比，对一个在一百万个文件中只有一个迁移候选者的 Solaris 设备来说，它只要花费十秒钟时间就能找到这个文件。这种方法可以与 NFS 类型的文件系统、XFS、UFS、Veritas 以及其他相似的文件系统结合使用，其使用 Unix 风格的操作系统，其中所述 Unix 风格的操作系统可以是例如 Linux 和 Solaris，但是也可以将所述方法扩展到其他操作系统和文件系统。

如图 15 和 16 所述，使用这里描述的容器文件 76 能使来自一个计算机系统（设备 A）的数据可用于另一个计算机系统（设备 B），而不需要首先对全部数据进行拷贝（将一个文件系统的内容复制到另一个系统通常是一个非常耗时的过程）。在步骤 400，可以将设备 B 配置成

使用处于后端 14 上的设备 A 的容器文件。每个容器文件 76 都包含了可以由设备 B 用来在其本地文件系统创建新文件的文件属性信息，例如大小、所有者、权限和路径。当设备 B 在步骤 402 中读取了*.cont 文件之后，在步骤 404 中将会创建新的文件，并且在步骤 406 中把新文件的大小设定为*.cont 文件 76 中指定的大小，以及在步骤 408 中释放所分配的空间（就好像已经清除了文件那样），由此在设备 B 的文件系统上创建一个存根文件。在步骤 410 中创建了一个.DMATTR 文件 60 或是其它扩展数据块或文件，并且还设定了属性和区域。在步骤 412 中则将设备 B 独有的容器（*.cont）文件写入后端 14，但是这也可以根据需要而在设备 B 修改数据并将其迁移到后端 14 的时候执行。在步骤 414，由于设备 B 的运作，数据存取请求通常会产生一个所请求数据不在设备 B 的本地文件系统上的判定，并且会把需要的分段从后端 14 拷贝到设备 B 的本地文件系统。而在设备 B 上则使用已经描述过的同一方式来更新文件属性和区域信息。在步骤 416，如果设备 B 修改了数据，则将那些经过修改的分段（显示为图 16 中的设备 B 的已修改数据）写入后端 14 并将关于变化的信息存入设备 B 的容器文件（显示为*.cont）。在步骤 418，设备 A 继续写入其自己的分段并且将关于其变化的信息保存在自己的容器文件 76 中。每一个计算机系统都把自己的已修改分段写入后端 14。每一个计算机系统使用的都是自己的容器文件，由此具有自己的数据版本。

前述公开和实施例论证了本发明在增加计算机系统中的对象管理效率方面的效用，但是很明显，本发明对于许多其他应用也是有益的。本发明在数据库、视频、音频以及任何一个只存取文件一部分并且只有这部分文件相关而不需要访问文件所有数据的应用中都具有特殊价值。

为了清楚起见，这里的进程和方法是结合特定的流来描述的，但是应该理解，在不脱离本发明精神的情况下，其他序列也是可行的，并且可以并行执行某些序列。此外还可以细分或组合步骤。如这里所公开的那样，依照本发明编写的软件可以用计算机可读介质的形式保存，例如内存或 CD-ROM，但也可以在网络上传送并由处理器执行。

这里引用的所有引证文献意图作为参考而被引入。尽管上文已经依照特定实施例而对本发明进行了描述，但是可以了解，对本领域技

术人员来说，很明显，针对本发明的变化和修改是显而易见的，并且这些变化和修改可以在附加权利要求的范围和等价物中得到实施。此外还可以通过例如并行使用多台计算机或者均分负载方案并且还可以将任务分发到多台计算机，从而使用一台以上的计算机，以使它们能够作为一个整体来执行对象管理系统的功能，也就是用它们来替代单独的计算机。上文描述的各种功能可以在单个计算机上由单个进程或进程组执行，也可以分发到几台计算机。这些进程可以调用其他进程来处理某些任务。所公开的原则不但适用于对象和对象存储，而且还适用于文件和文件系统。此外，所给出的实施例应视为是说明性而不是限制性的，并且并未将本发明局限于这里给出的细节。因此，在这里意图将本公开和所附权利要求理解为覆盖了落入本发明真正精神和范围的所有这类变化和修改。

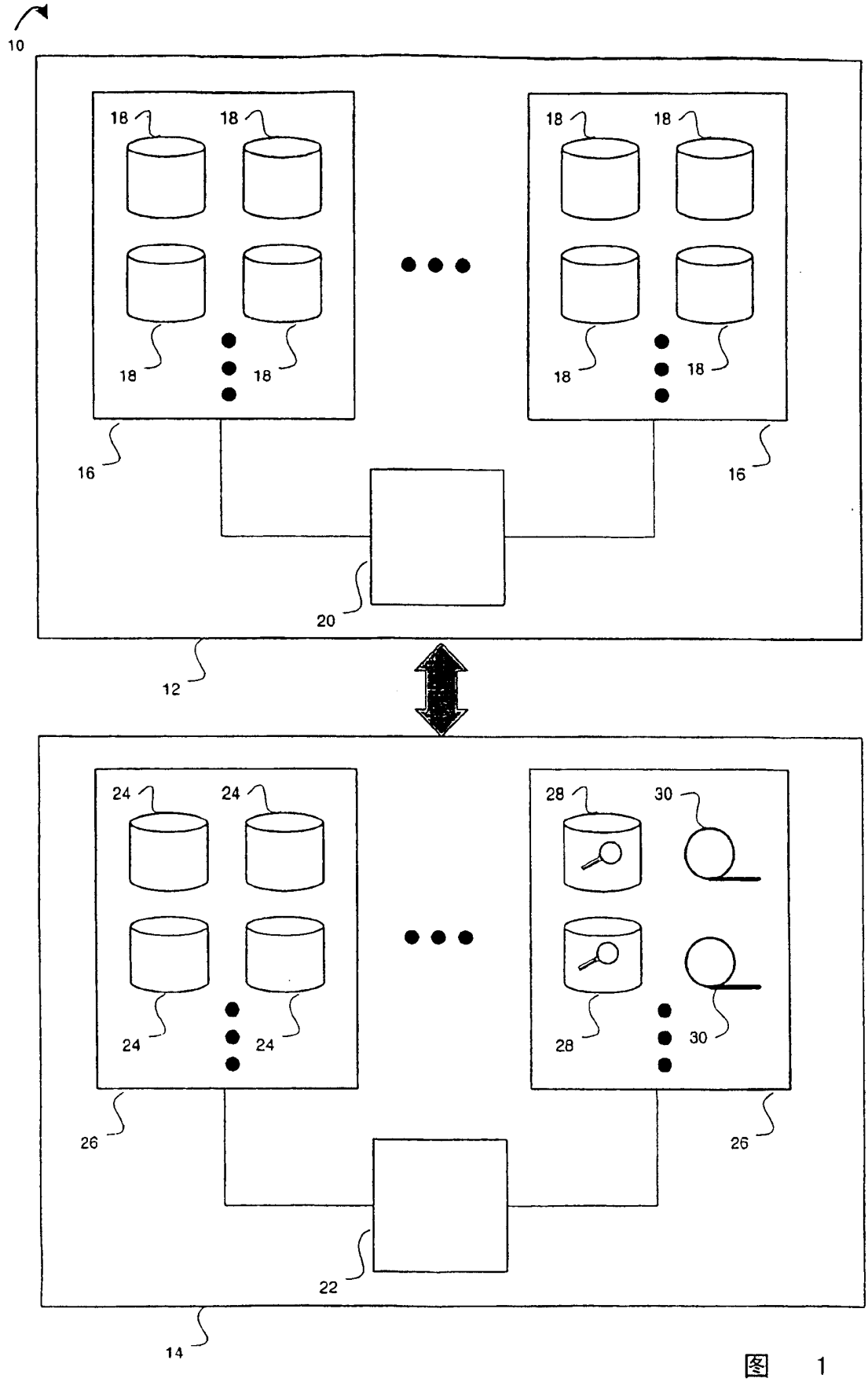

10
18
18
18
18
18
18
18
18
16
16
20
12
24
24
28
30
24
24
28
30
26
26
22
14

图 1

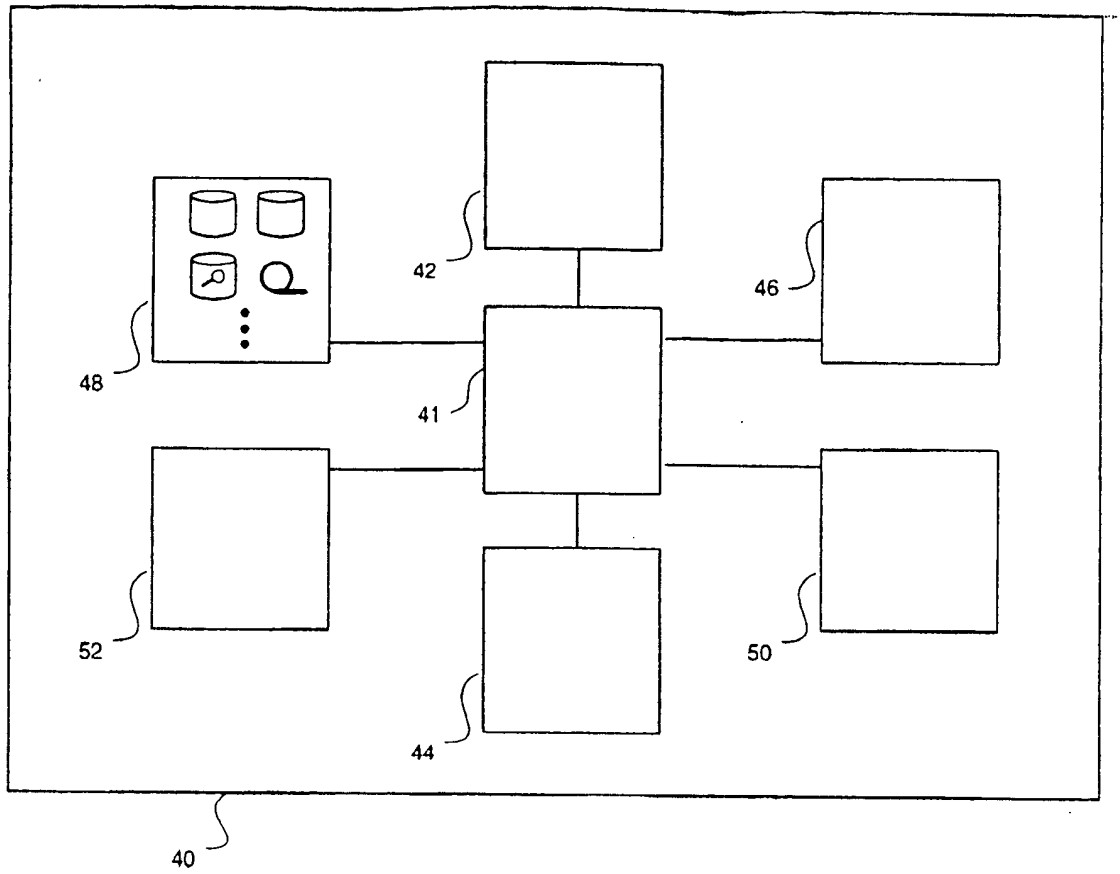

42
46
48
41
52
50
44
40

图 2

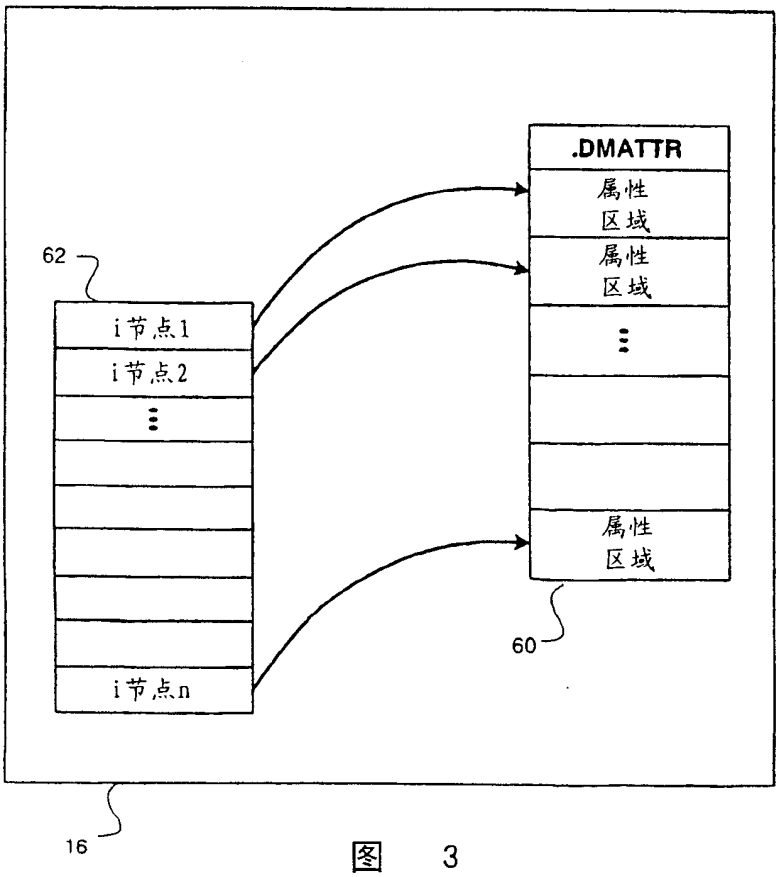

.DMATTR
属性区域
属性区域
属性区域
62
i节点1
i节点2
i节点n
60
16

图 3

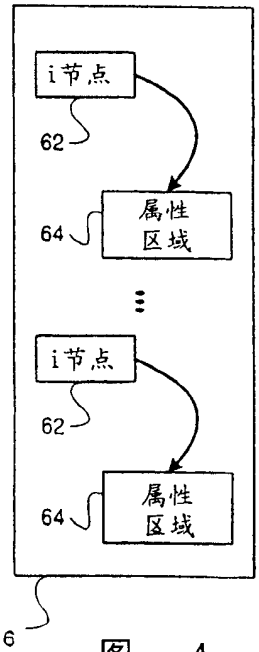

i节点
62
64
属性区域
i节点
62
64
属性区域
16

图 4

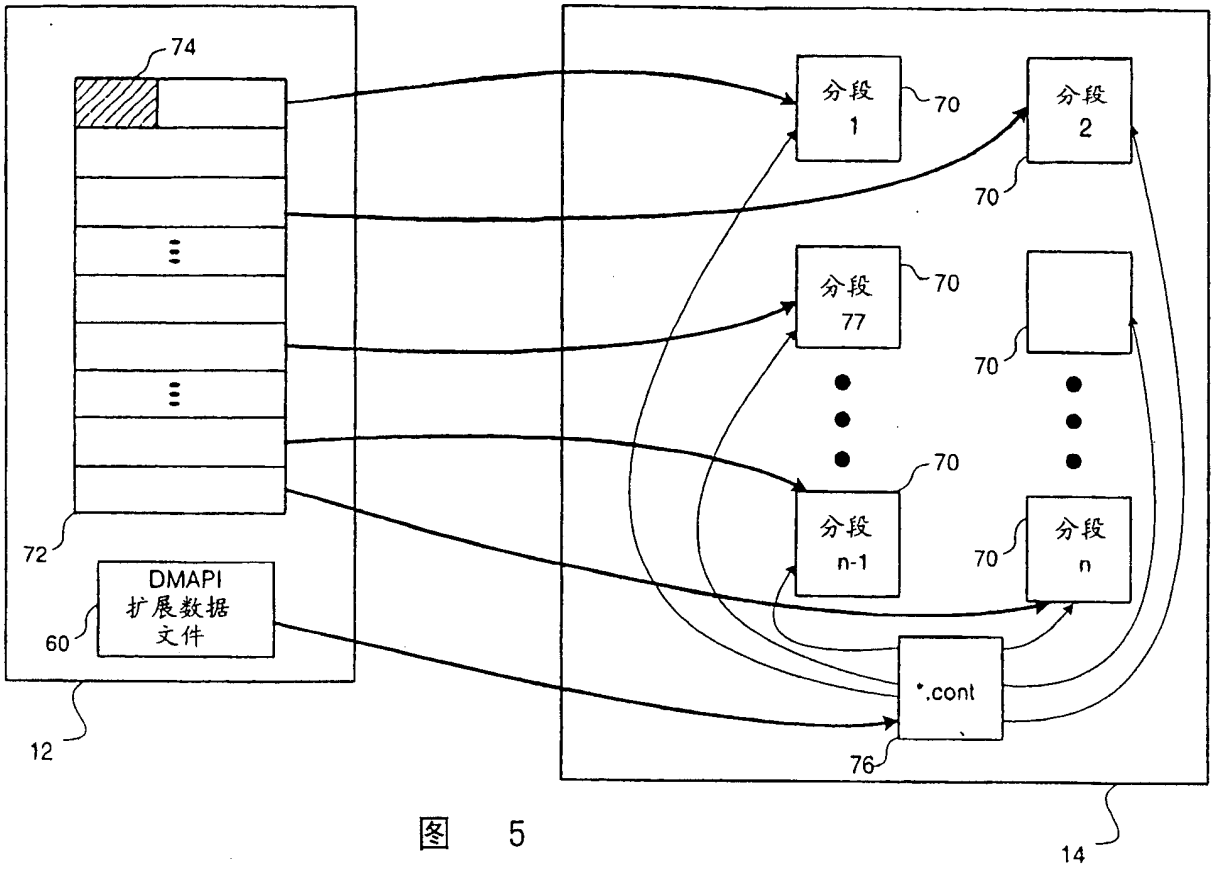

74
72
DMAPI
扩展数据
文件
60
12
分段
1
70
分段
2
70
分段
77
70
70
分段
n-1
70
分段
n
70
*.cont
76
14

图 5

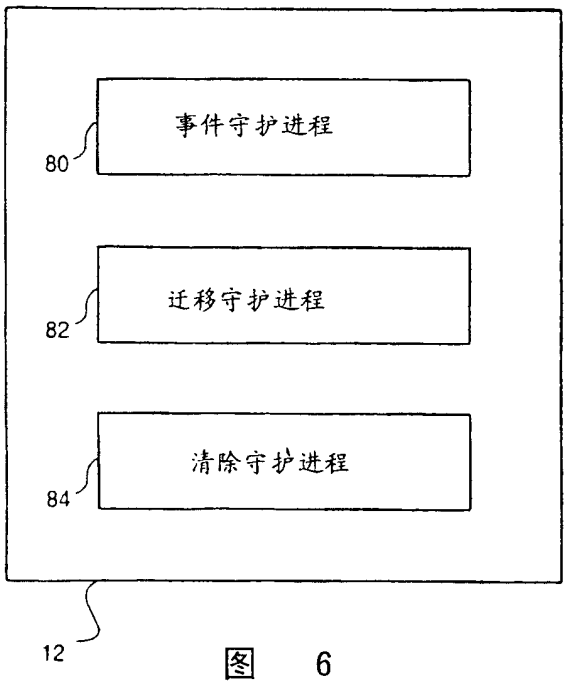

事件守护进程
80
迁移守护进程
82
清除守护进程
84
12

图 6

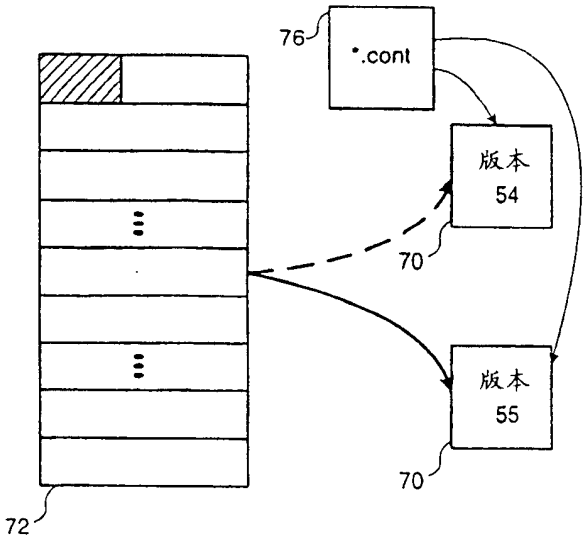

76
*.cont
版本
54
70
版本
55
70
72

图 7

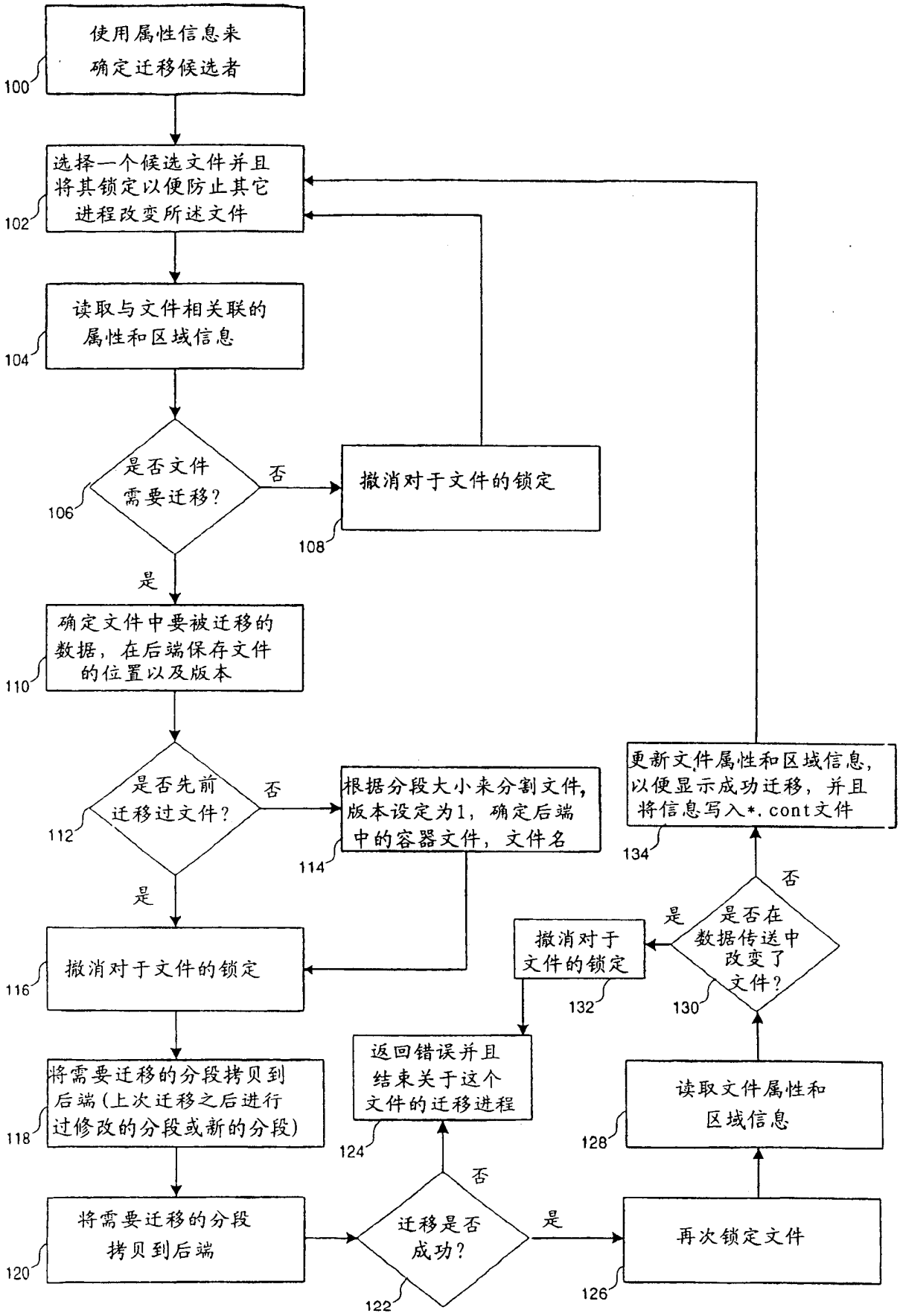

使用属性信息来确定迁移候选者
100
选择一个候选文件并且将其锁定以便防止其它进程改变所述文件
102
读取与文件相关联的属性和区域信息
104
是否文件需要迁移?
106
否
撤消对于文件的锁定
108
是
确定文件中要被迁移的数据，在后端保存文件的位置以及版本
110
是否先前迁移过文件?
112
否
根据分段大小来分割文件，版本设定为1，确定后端中的容器文件，文件名
114
是
撤消对于文件的锁定
116
将需要迁移的分段拷贝到后端(上次迁移之后进行过修改的分段或新的分段)
118
将需要迁移的分段拷贝到后端
120
迁移是否成功?
122
否
返回错误并且结束关于这个文件的迁移进程
124
是
再次锁定文件
126
读取文件属性和区域信息
128
是否在数据传送中改变了文件?
130
是
撤消对于文件的锁定
132
否
更新文件属性和区域信息，以便显示成功迁移，并且将信息写入*.cont文件
134

图 8

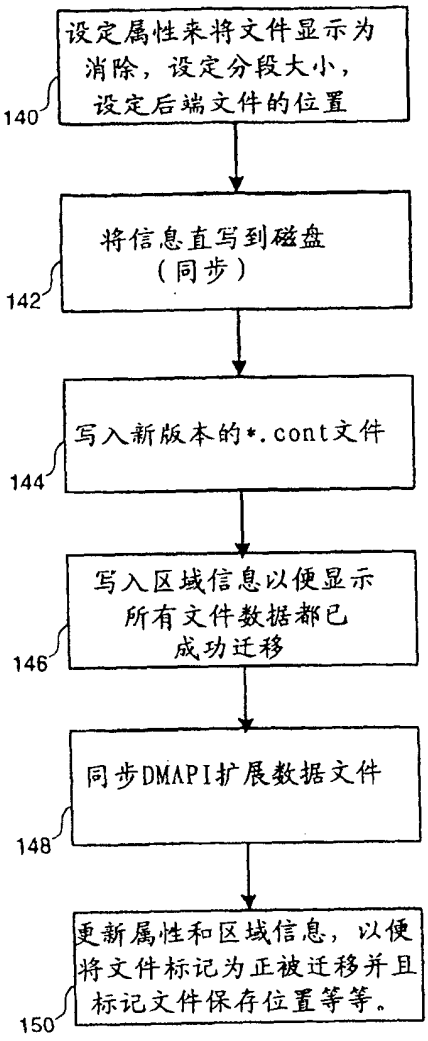

设定属性来将文件显示为消除，设定分段大小，设定后端文件的位置
140
将信息直写到磁盘（同步）
142
写入新版本的*.cont文件
144
写入区域信息以便显示所有文件数据都已成功迁移
146
同步DMAPI扩展数据文件
148
更新属性和区域信息，以便将文件标记为正被迁移并且标记文件保存位置等等。
150

图 9

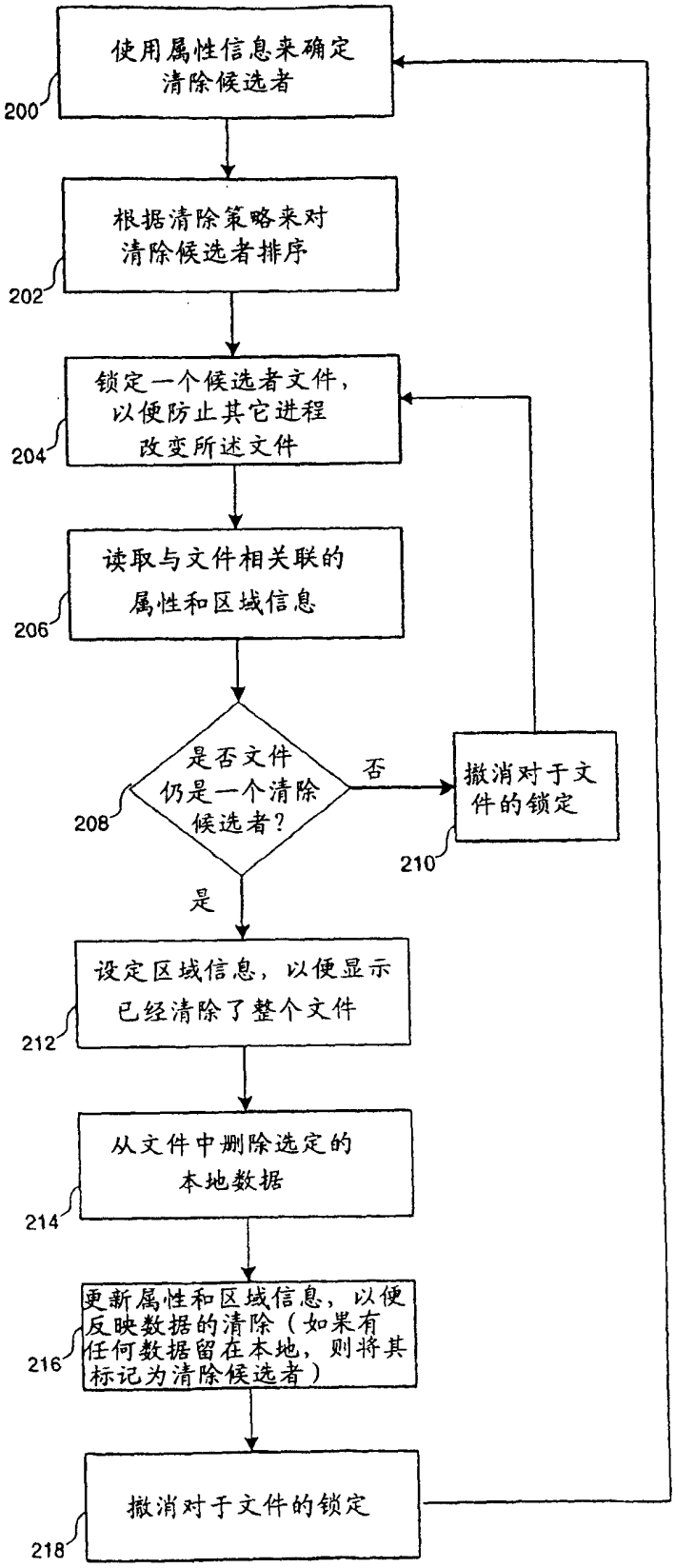

使用属性信息来确定清除候选者
200
根据清除策略来对清除候选者排序
202
锁定一个候选者文件，以便防止其它进程改变所述文件
204
读取与文件相关联的属性和区域信息
206
是否文件仍是一个清除候选者？
208
否
撤消对于文件的锁定
210
是
设定区域信息，以便显示已经清除了整个文件
212
从文件中删除选定的本地数据
214
更新属性和区域信息，以便反映数据的清除（如果有任何数据留在本地，则将其标记为清除候选者）
216
撤消对于文件的锁定
218

图 10

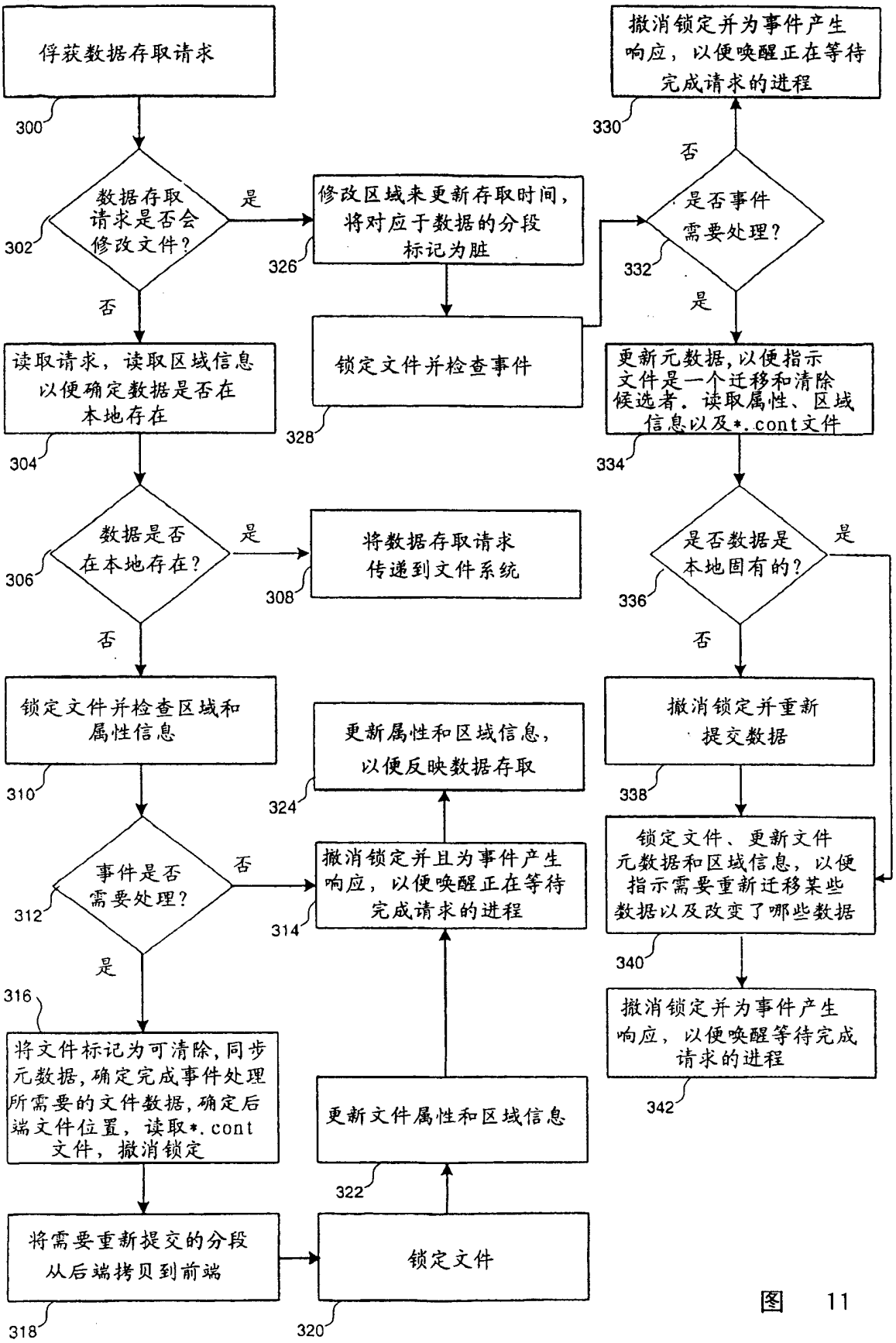

俘获数据存取请求
300
数据存取请求是否会修改文件?
302
是
否
修改区域来更新存取时间，将对应于数据的分段标记为脏
326
读取请求，读取区域信息以便确定数据是否在本地存在
304
锁定文件并检查事件
328
数据是否在本地存在?
306
是
否
将数据存取请求传递到文件系统
308
锁定文件并检查区域和属性信息
310
更新属性和区域信息，以便反映数据存取
324
事件是否需要处理?
312
否
是
撤消锁定并且为事件产生响应，以便唤醒正在等待完成请求的进程
314
将文件标记为可清除，同步元数据，确定完成事件处理所需要的文件数据，确定后端文件位置，读取*.cont文件，撤消锁定
316
更新文件属性和区域信息
322
将需要重新提交的分段从后端拷贝到前端
318
锁定文件
320
撤消锁定并为事件产生响应，以便唤醒正在等待完成请求的进程
330
否
是否事件需要处理?
332
是
更新元数据，以便指示文件是一个迁移和清除候选者．读取属性、区域信息以及*.cont文件
334
是否数据是本地固有的?
336
是
否
撤消锁定并重新提交数据
338
锁定文件、更新文件元数据和区域信息，以便指示需要重新迁移某些数据以及改变了哪些数据
340
撤消锁定并为事件产生响应，以便唤醒等待完成请求的进程
342

图 11

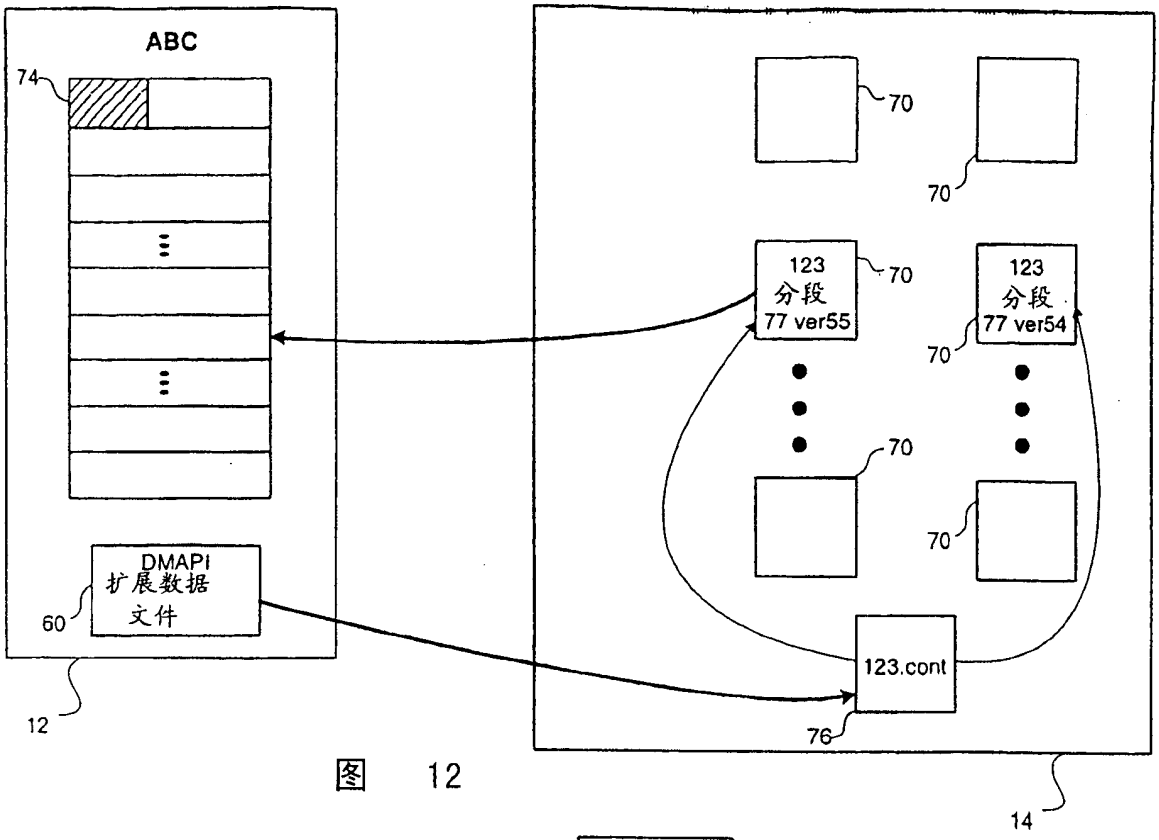

ABC
74
DMAPI
扩展数据
文件
60
12
70
123
分段
77 ver55
123
分段
77 ver54
123.cont
76
14

图 12

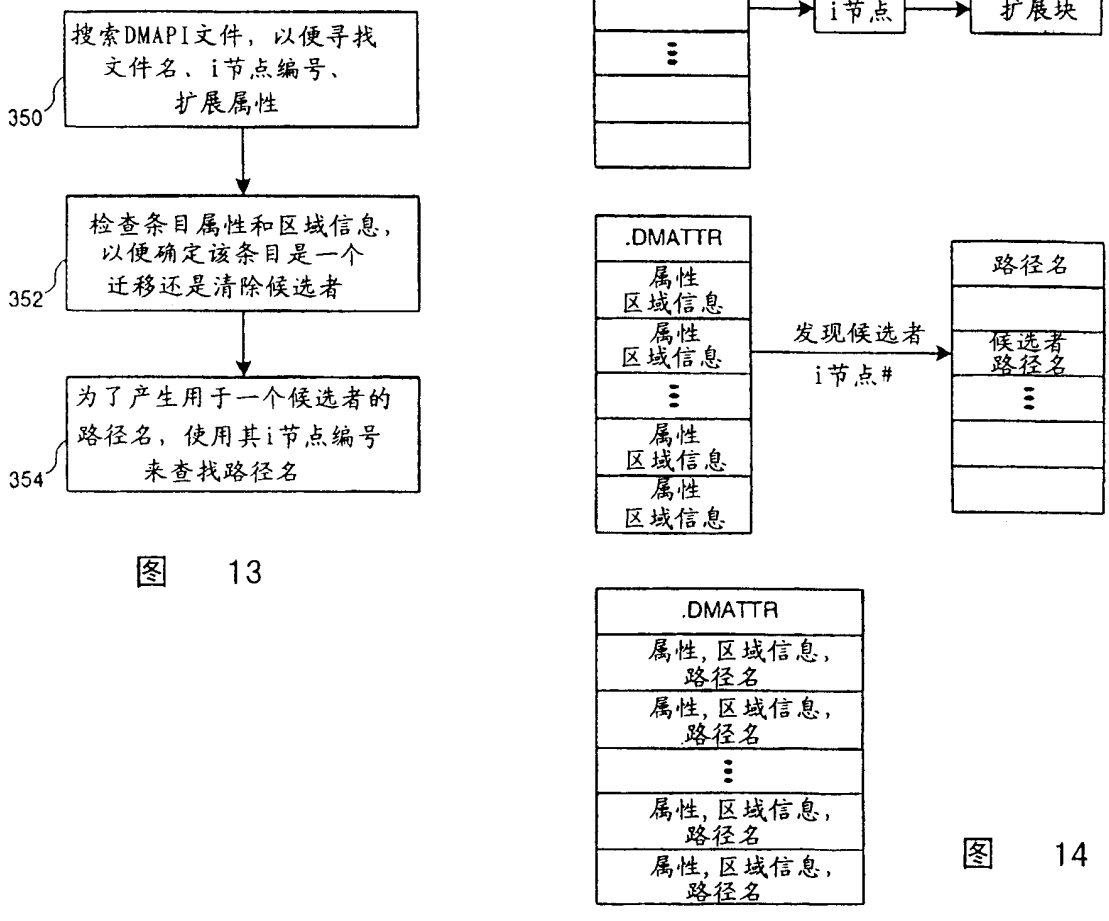

搜索DMAPI文件，以便寻找
文件名、i节点编号、
扩展属性
350
检查条目属性和区域信息，
以便确定该条目是一个
迁移还是清除候选者
352
为了产生用于一个候选者的
路径名，使用其i节点编号
来查找路径名
354
名称空间
i节点
扩展块
.DMATTR
属性
区域信息
属性
区域信息
属性
区域信息
属性
区域信息
发现候选者
i节点#
路径名
候选者
路径名
.DMATTR
属性，区域信息，
路径名
属性，区域信息，
路径名
属性，区域信息，
路径名
属性，区域信息，
路径名

图 13

图 14

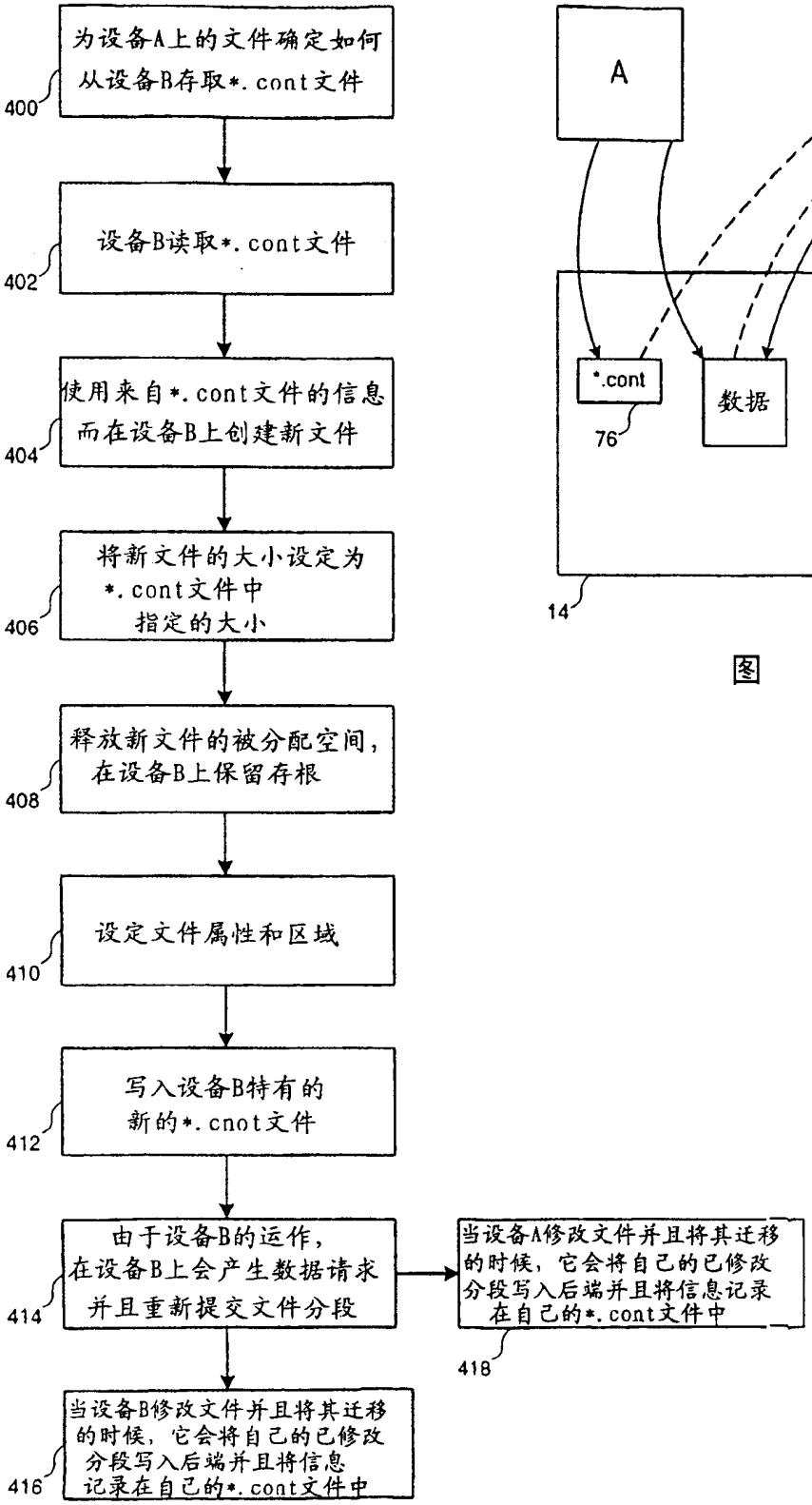

为设备A上的文件确定如何从设备B存取*.cont文件
400
设备B读取*.cont文件
402
使用来自*.cont文件的信息而在设备B上创建新文件
404
将新文件的大小设定为*.cont文件中指定的大小
406
释放新文件的被分配空间，在设备B上保留存根
408
设定文件属性和区域
410
写入设备B特有的新的*.cnot文件
412
由于设备B的运作，在设备B上会产生数据请求并且重新提交文件分段
414
当设备B修改文件并且将其迁移的时候，它会将自己的已修改分段写入后端并且将信息记录在自己的*.cont文件中
416
当设备A修改文件并且将其迁移的时候，它会将自己的已修改分段写入后端并且将信息记录在自己的*.cont文件中
418

图 15

A
B
*.cont
76
数据
*.cont'
76
B'的已修改数据
14

图 16