(19)国家知识产权局

(12)发明专利

(10)授权公告号 CN 112219189 B
(45)授权公告日 2024.07.23

(21)申请号 201980034671.0
(22)申请日 2019.03.27
(65)同一申请的已公布的文献号
申请公布号 CN 112219189 A
(43)申请公布日 2021.01.12
(30)优先权数据
62/648,907 2018.03.27 US
(85)PCT国际申请进入国家阶段日
2020.11.23
(86)PCT国际申请的申请数据
PCT/US2019/024417 2019.03.27
(87)PCT国际申请的公布数据
WO2019/191320 EN 2019.10.03

(73)专利权人 奈飞公司
地址 美国加利福尼亚州
(72)发明人 维奈·切拉 约瑟夫·林奇
阿贾伊·乌帕德海耶
(74)专利代理机构 北京东方亿思知识产权代理有限责任公司 11258
专利代理师 张敏

(51)Int.Cl.
G06F 9/48(2006.01)
G06F 16/27(2006.01)
(56)对比文件
US 6098078 A,2000.08.01
审查员 赵静

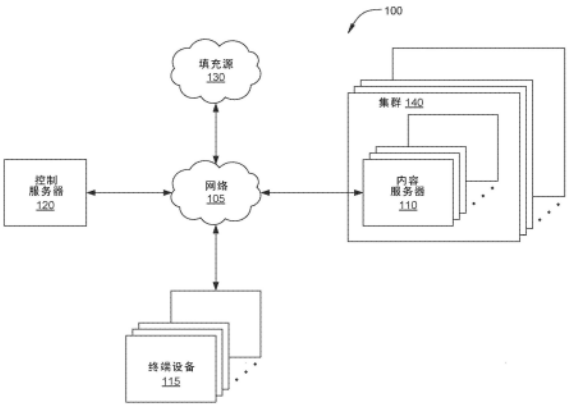
权利要求书3页 说明书19页 附图6页

(54)发明名称

用于调度的反熵修复设计的技术

(57)摘要

本文所公开的本发明的各个实施例提供了用于在分布式数据库网络中的复数个节点上执行分布式反熵修复程序的技术。该分布式数据库网络内的复数个节点中所包括的一节点在复数个节点中所包括的所有其他节点之前确定第一反熵修复程序已结束。该节点确定第二反熵修复程序准备好开始。该节点生成用于执行与第二反熵修复程序相关联的一个或多个操作的调度。该节点将调度写入共享的修复调度数据结构,以在复数个节点中所包括的多个节点上启动第二反熵修复程序。然后复数个节点中所包括的每个节点基于该调度来执行节点修复。



CN 112219189 B

1.一种计算机实现的方法,包括:

由复数个节点中所包括的第一节点在所述复数个节点中所包括的所有其他节点之前确定第一反熵修复程序已结束;

由所述第一节点确定第二反熵修复程序准备好开始;

生成用于执行与所述第二反熵修复程序相关联的一个或多个操作的第一调度,其中,所述第一调度包括在所述第二反熵修复程序期间由所述复数个节点中包括的多个节点执行一个或多个节点修复的顺序;以及

将所述第一调度写入共享的修复调度数据结构,以在所述复数个节点中所包括的所述多个节点上启动所述第二反熵修复程序。

2.根据权利要求1所述的计算机实现的方法,其中确定所述第二反熵修复程序准备好开始包括:确定当前时间在指定用于修复操作的规定时间范围内。

3.根据权利要求1所述的计算机实现的方法,还包括:

由所述复数个节点中所包括的第二节点确定第三反熵修复程序准备好开始;

生成用于执行与所述第三反熵修复程序相关联的一个或多个操作的第二调度;以及

将所述第二调度写入第二共享的修复调度数据结构,以在所述复数个节点中所包括的多个节点上启动所述第三反熵修复程序。

4.根据权利要求3所述的计算机实现的方法,其中:

所述第二反熵修复程序包括完整的反熵修复程序,

所述第三反熵修复程序包括增量式反熵修复程序,并且

确定所述第三反熵修复程序准备好开始包括:确定包括增量式修复的第四反熵修复程序已结束。

5.根据权利要求1所述的计算机实现的方法,还包括:

由所述复数个节点中所包括的第二节点确定所述第二反熵修复程序正在进行中;

确定所述第二节点按所述顺序是下一个要被修复的;以及

修复驻留在所述第二节点上的至少一个不一致数据分区。

6.根据权利要求1所述的计算机实现的方法,还包括:

由所述复数个节点中所包括的第二节点确定所述第二反熵修复程序正在进行中;

确定所述第二节点独立于所述复数个节点中所包括的当前正在执行与所述第二反熵修复程序相关联的节点修复的所有其他节点;以及

修复驻留在所述第二节点上的至少一个不一致数据分区。

7.根据权利要求1所述的计算机实现的方法,还包括:

由所述复数个节点中所包括的第二节点确定所述第二反熵修复程序正在进行中;

确定所述第二节点已经执行了与所述第二反熵修复程序相关联的节点修复;

确定所述复数个节点中所包括的与所述第二节点相互依赖的所有其他节点已经执行了与所述第二反熵修复程序相关联的一个或多个附加的节点修复;以及

由所述第二节点执行与所述第二反熵修复程序相关联的修复后程序。

8.根据权利要求7所述的计算机实现的方法,其中执行所述修复后程序包括:删除在所述第二反熵修复程序完成之后不再需要的分区。

9.根据权利要求7所述的计算机实现的方法,其中执行所述修复后程序包括:对与所述

第二节点相关联的一个或多个分区执行压缩操作以缩短访问所述一个或多个分区时的等待时间。

10.根据权利要求7所述的计算机实现的方法,其中执行所述修复后程序包括:发送消息给监测应用程序,所述消息指示所述第二节点已经执行了与所述第二反熵修复程序相关联的节点修复。

11. 根据权利要求1所述的计算机实现的方法,还包括:

确定与所述第二反熵修复程序相关联的分区的数量超过阈值水平;以及

将与所述第二反熵修复程序相关联的工作划分为复数个子范围。

12. 根据权利要求1所述的计算机实现的方法,还包括:

确定与所述第二反熵修复程序相关联的一个或多个分区的大小超过阈值水平;以及

将与所述第二反熵修复程序相关联的工作划分为复数个子范围。

13. 根据权利要求1所述的计算机实现的方法,还包括:

确定所述第二反熵修复程序的完成时间超过阈值水平;以及

基于所述完成时间,将与所述第二反熵修复程序相关联的工作划分为复数个子范围。

14.一种非暂态计算机可读存储介质,包括指令,所述指令当由一个或多个处理器执行时使所述一个或多个处理器执行以下步骤:

确定第一反熵修复程序已结束;

由复数个节点中所包括的第一节点在所述复数个节点中所包括的所有其他节点之前确定第二反熵修复程序准备好开始;

生成用于执行与所述第二反熵修复程序相关联的操作的调度,其中,所述调度包括在所述第二反熵修复程序期间由所述复数个节点中包括的多个节点执行一个或多个节点修复的顺序;以及

将所述调度写入共享的修复调度数据结构,以在所述复数个节点中所包括的多个节点上启动所述第二反熵修复程序。

15.根据权利要求14所述的一种非暂态计算机可读存储介质,其中确定所述第二反熵修复程序准备好开始包括:确定当前时间在指定用于修复操作的规定时间范围内。

16.根据权利要求14所述的一种非暂态计算机可读存储介质,其中所述复数个节点中所包括的每个节点顺序地执行与所述第二反熵修复程序相关联的节点修复。

17.根据权利要求14所述的一种非暂态计算机可读存储介质,其中:

所述复数个节点中所包括的第一节点子集中所包括的每个节点彼此并行地执行与所述第二反熵修复程序相关联的第一节点修复;

所述复数个节点中所包括的第二节点子集中所包括的每个节点彼此并行地执行与所述第二反熵修复程序相关联的第二节点修复;并且

所述第一节点子集中所包括的节点执行与所述第二反熵修复程序相关联的所述第一节点修复与所述第二节点子集中所包括的节点执行所述第二节点修复按顺序执行。

18.根据权利要求14所述的一种非暂态计算机可读存储介质,其中所述复数个节点中所包括的每个节点彼此并行地执行与所述第二反熵修复程序相关联的节点修复。

19. 一种计算设备,包括:

存储器,所述存储器包括指令;以及

处理器,所述处理器耦合至所述存储器,并且当执行所述指令时被配置为:

确定第一反熵修复程序已结束;

由复数个节点中所包括的第一节点在所述复数个节点中所包括的所有其他节点之前确定第二反熵修复程序准备好开始;

生成与所述第二反熵修复程序相关联的修复调度,其中,所述修复调度包括在所述第二反熵修复程序期间由所述复数个节点中包括的多个节点执行一个或多个节点修复的顺序;以及

将所述修复调度写入数据存储装置以在所述复数个节点中所包括的多个节点上启动所述第二反熵修复程序。

20.根据权利要求19所述的计算设备,其中所述处理器当执行所述指令时还被配置为:在所述复数个节点中所包括的第二节点处指派多个处理器核心以执行与所述第二反熵修复程序相关联的节点修复,其中所述处理器核心的数量最大为所述第二节点内可用的处理器核心的数量的一半。

用于调度的反熵修复设计的技术

[0001] 相关申请的交叉引用

[0002] 本申请要求于2018年3月27日提交的美国专利申请序列号为62/648,907(代理人案卷号NETF0192USL)的申请的权益,该美国专利申请据此以引用方式并入本文。

技术领域

[0003] 本发明整体上涉及分布式计算机系统,并且更具体地涉及用于调度的反熵修复设计的技术。

背景技术

[0004] 在某些分布式数据库网络中,数据集合被复制并存储在整个分布式数据库网络中的多个计算设备中,这多个计算设备在此被称为“节点”。数据集合的多个副本在本文中称为“复制品”。在整个分布式数据库网络中存储复制品提供了防止数据丢失的冗余,由此如果复制品中的一个复制品被破坏,则可以访问其余复制品以检索对应的数据。此外,当复制品存储在地理上不同的节点中时,请求访问特定数据集的用户可以从最接近所述用户的计算设备的节点检索数据。因此,可以减少请求访问副本与检索所述副本之间的等待时间。

[0005] 随着时间的推移,一个复制品中的数据可能变得与其他对应复制品中的数据不一致。作为一个示例,访问一个节点上的特定复制品的用户可修改所述复制品的一部分,然后将所述经修改的复制品存储回同一节点上。因此,经修改的复制品与在整个分布式数据库网络中所分布的其他复制品不一致。为了校正此类不一致,将所述节点中的一个节点指定为反熵修复协调器。然后,指定的反熵修复协调器就不一致性分析复制品子集之间的差异并更新复制品子集,以使复制品子集变得一致。分析和更新复制品的这种过程在本文中称为“反熵修复程序(anti-entropy repair procedure)”,或更简单地称为“修复”。使用常规修复时,用户可以通过分布式数据库外部的修复工具手动将修复调度为离线批处理过程。因此,利用常规反熵修复程序,用户负责为分布式数据库网络内的节点设计和维护反熵修复程序。

[0006] 上述用于修复节点的途径的一个问题是修复解决方案和选项对于用户来说难以正确理解和执行。简单的反熵修复程序对于具有相对较少节点数的分布式数据库网络可为有效的,但是对于具有数万个节点的分布式数据库网络,这种简单的反熵修复程序可能会发生故障。类似地,对存储大小相对较小的复制品的节点可有效的反熵修复程序可能对于存储相对较大复制品的节点会发生故障,反之亦然。因此,对于具有波动数量的节点和/或不同大小的复制品的分布式数据库网络,设计和维持反熵修复程序可能是困难的。

[0007] 上述用于修复节点的途径的另一个问题是,当修复正在进行的同时反熵修复协调器和/或一个或多个其他节点发生故障时,整个修复进程可能会丢失。因此,需要从头开始重新启动反熵修复程序,从而导致丢失在故障之前执行了数小时或数天的修复工作。此外,外部的反熵修复程序可能无法完全了解分布式数据库的状态以及发生故障时的修复进程。因此,反熵修复程序可能难以确定需要哪些特定动作来重新启动修复。上述用于修复节点

的途径的另一个问题是,修复通常需要大量使用磁盘存储装置、中央处理器(CPU)资源和网络带宽,这可能会显著降低分布式数据库网络的性能。

[0008] 如前所述,本领域中需要的是用于修复分布式数据库网络中的节点的更有效的技术。

发明内容

[0009] 本申请的各个实施例阐述了一种用于在分布式数据库网络中的复数个节点上执行分布式反熵修复程序的计算机实现的方法。该方法包括由复数个节点中所包括的第一节点在复数个节点中所包括的所有其他节点之前,确定第一反熵修复程序已结束。该方法还包括由第一节点确定第二反熵修复程序准备好开始。该方法还包括生成用于执行与第二反熵修复程序相关联的一个或多个操作的调度。该方法还包括将该调度写入共享的修复调度数据结构,以在复数个节点中所包括的多个节点上启动第二反熵修复程序。

[0010] 本发明的其他实施例包括但不限于一种计算机可读介质,该计算机可读介质包括用于执行所公开的技术的一个或多个方面的指令;以及一种用于执行所公开的技术的一个或多个方面的计算设备。

[0011] 相对于现有技术,所公开的技术的至少一个技术优点是,反熵修复程序根据分布式数据库网络中的节点数量和复制品的变化的大小自动地缩放。因此,相对于传统技术,反熵修复程序适应于存储在特定节点上的数据的每个子集,以便减少磁盘访问量、CPU资源和所消耗的网络带宽。此外,随着节点数量和复制品大小随时间推移变化,用户无需手动设计和维护反熵修复程序。相对于现有技术,所公开的技术的另一技术优势在于,反熵修复程序分布在各节点上。此外,一旦故障节点恢复执行,则反熵修复程序可以在与故障发生时相对相同的条件下恢复。因此,相对于现有技术途径,当一个或多个其他节点在修复期间发生故障时,很少或没有修复进程丢失。这些技术优点代表相对于现有技术途径的一项或多项技术改进。

附图说明

[0012] 因此,可以详细地理解本发明的上述特征的方式,可以通过参考实施例来对上面简要概述的本发明进行更具体的描述,其中所述实施例中的一些实施例在附图中示出。然而,应当注意的是,附图仅示出了本发明的典型实施例,因此不应视为对本发明范围的限制,因为本发明可允许其他等效实施例。

[0013] 图1示出了根据本发明的各种实施例的用于将内容分发到内容服务器和终端设备的网络基础设施;

[0014] 图2是根据本发明的各种实施例的可结合图1的网络基础设施来实现的内容服务器的更详细图示;

[0015] 图3是根据本发明的各种实施例的可结合图1的网络基础设施来实现的控制服务器的更详细图示;

[0016] 图4是根据本发明的各种实施例的可结合图1的网络基础设施实现的终端设备的更详细的图示;并且

[0017] 图5A-5B阐述了根据本发明的各种实施例的用于在分布式数据库网络中的复数个

内容服务器上执行分布式反熵修复程序的方法步骤的流程图。

具体实施方式

[0018] 在下面的描述中,阐述了许多具体细节以提供对本发明的更透彻理解。然而,对于本领域的技术人员将显而易见的是,可以在没有这些具体细节中的一者或多者的情况下实践本发明的实施例。

[0019] 系统综述

[0020] 图1示出了根据本发明的各种实施例的用于将内容分发到内容服务器110和终端设备115的网络基础设施100。如图所示,网络基础结构100包括集群140、控制服务器120和终端设备115,它们中的每一者都经由通信网络105连接。网络105可以是使得能够实现远程或本地计算机系统与计算设备之间的通信的任何合适的环境,包括但不限于无线和有线LAN以及基于互联网的WAN(广域网)。

[0021] 每个终端设备115包括但不限于计算设备,所述计算设备可以是个人计算机、视频游戏控制台、个人数字助理、移动电话、移动设备,或适用于实现本发明的一个或多个方面的任何其他设备。每个终端设备115经由网络105与一个或多个内容服务器110通信以下载内容,例如文本数据、图形数据、音频数据、视频数据和其他类型的数据。内容服务器110在本文中也称为“高速缓存”、“计算节点”,或更简单地称为“节点”。然后将可下载内容(在本文中也称为“文件”)呈现给一个或多个终端设备115的用户。在各种实施例中,终端设备115可包括计算机系统、机顶盒、移动计算机、智能电话、平板电脑、控制台和手持式视频游戏系统、数字视频记录器(DVR)、DVD播放器、经连接的数字电视、专用媒体流式传输设备(例如,Roku®机顶盒),和/或任何其他技术上可行的计算平台,所述计算平台具有网络连通性并且能够向用户呈现内容,例如文本、图像、视频和/或音频内容。

[0022] 每个集群140包括一个或多个内容服务器110。如本文中进一步描述的,每个集群140能够相对于特定集群140中所包括的内容服务器110独立地执行反熵修复程序。每个内容服务器110包括但不限于可为以下的存储设备:独立的网络附加存储(NAS)系统、存储区域网络(SAN)、存储设备的群集或“场”、分布式存储架构,或适用于实现本发明的一个或多个方面的任何其他设备。另外或可替代地,每个内容服务器110可包括但不限于具有存储子系统的计算设备,所述存储子系统可为独立服务器、服务器的集群或“场”、一个或多个网络家电,或适合于实现本发明的一个或多个方面的任何其他设备。此外,每个内容服务器110可包括但不限于网络服务器和数据库,并且可被配置为与控制服务器120通信以确定由控制服务器120跟踪和管理的各种文件的位置和可用性。

[0023] 更一般地,每个内容服务器110存储被称为表的文件组。每个表继而由分区组成,其中分区是任意大小的数据单位。分区的大小可以较小,例如单个键值对,也可以较大,例如一个千兆字节的数据。每个内容服务器110还可以与填充源130和一个或多个其他内容服务器110通信,以用各种文件的副本“填充”每个内容服务器110。另外,内容服务器110可响应于对从终端设备115接收的文件的请求。然后可以从内容服务器110或经由更广泛的内容分发网络来分发文件。在一些实施例中,内容服务器110使用户能够认证(例如,使用用户名和密码)以便访问存储在内容服务器110上的文件。

[0024] 控制服务器120可包括但不限于计算设备,所述计算设备可为独立服务器、服务器

的集群或“场”、一个或多个网络家电，或适合于实现本发明的一个或多个方面的任何其他设备。尽管在图1中仅示出了单个控制服务器120，但是在各种实施例中，可以实现多个控制服务器120以跟踪和管理文件。

[0025] 在各种实施例中，填充源130可以包括在线存储服务（例如，Amazon®简单存储服务、Google®云存储等），在所述在线存储服务中存储和访问包括数千或数百万个文件的文件目录，以便填充内容服务器110。尽管在图1中仅示出了单个填充源130，但是在各种实施例中，可以实现多个填充源130以服务于对文件的请求。

[0026] 在整个集群中执行分布式反熵修复程序

[0027] 图2是根据本发明的各种实施例的可以结合图1的网络基础设施100实现的内容服务器110的框图。如图所示，内容服务器110包括但不限于处理器204、系统盘206、输入/输出（I/O）设备接口208、网络接口210、互连件212，以及系统存储器214。

[0028] 处理器204被包括以代表单个CPU、多个CPU、具有多个处理核心的单个CPU等。处理器204被配置为检索并执行存储在系统存储器214中的编程指令，例如服务器应用程序217和修复应用程序219。类似地，处理器204被配置为存储应用程序数据（例如，软件库）并从系统存储器214检索应用程序数据。互连件212被配置为促进处理器204、系统盘206、I/O设备接口208、网络接口210和系统存储器214之间的数据（例如编程指令和应用程序数据）的传输。I/O设备接口208被配置为从I/O设备216接收输入数据，并且经由互连件212将输入数据发送到处理器204。例如，I/O设备216可包括一个或多个按钮、键盘、鼠标和/或其他输入设备。I/O设备接口208还被配置为经由互连件212从处理器204接收输出数据，并且将输出数据发送到I/O设备216。

[0029] 系统盘206可包括一个或多个硬盘驱动器、固态存储设备或类似的存储设备。系统盘206被配置为存储非易失性数据，例如文件218（例如，音频文件、视频文件、字幕、应用程序文件、软件库等）。然后，可由一个或多个终端设备115经由网络105来检索文件218，或更具体地，与一个或多个文件218相关联的分区和/或记录。在一些实施例中，网络接口210被配置为遵照以太网标准操作。

[0030] 系统存储器214包括但不限于服务器应用程序217、修复应用程序219和数据存储装置221。数据存储装置221包括服务器应用程序数据存储装置，其中的数据由服务器应用程序217存储和检索。在一些实施例中，数据存储装置221还可包括修复数据存储装置，其中的数据由修复应用程序219存储和检索。另外地或可替代地，修复数据存储装置可驻留在内容服务器110外部的数据存储装置中。在这种情况下，修复应用程序219可经由网络接口210访问外部修复数据存储装置。

[0031] 服务器应用程序217被配置为服务于对从终端设备115和其他内容服务器110接收的一个或多个文件218中的一个或多个分区的请求。当服务器应用程序217接收到对一个或多个文件218内的一个或多个分区的请求时，服务器应用程序217从系统盘206中检索对应的文件218，并将实体化的分区经由网络105传输到终端设备115或内容服务器110。在执行这些操作时，服务器应用程序217将数据存储在数据存储装置221中并从所述数据存储装置中检索数据。

[0032] 修复应用程序219当由处理器204执行时，执行与图1的内容服务器110相关联的一个或多个操作，如本文所进一步描述的。在执行这些操作时，修复应用程序219将数据存储

在数据存储装置221中并从所述数据存储装置中检索数据。

[0033] 在操作中,在处理器204上执行的修复应用程序219通过修复状态更新来管理反熵修复程序的执行。在一个实施例中,修复应用程序219经由包括四种类型的状态表的数据结构来管理和协调反熵修复程序的执行。这四个状态表在本文中被称为repair_process(修复_过程)、repair_sequence(修复_顺序)、repair_status(修复_状态)和repair_hook_status(修复_挂钩_状态)。对于特定的反熵修复程序,集群140中所包括的每个内容服务器110都有一个repair_process状态记录和一个repair_sequence记录。此外,对于在特定内容服务器110上执行的特定反熵修复程序,每个分区表都有一个repair_status记录,并且集群140中所包括的每个内容服务器110都有一个repair_hook_status状态表。在各种实施例中,repair_process、repair_sequence、repair_status和repair_hook_status状态表中的每一者可以任何技术上可行的组合存在于每个群集上或一个主数据存储装置中。在此类实施例中,主repair_status状态表和一个主repair_hook_status状态表可被划分为多个区段(section),其中每个区段对应于集群140中所包括的不同内容服务器110。

[0034] 第一状态表是repair_process状态表,其包括用于在整个集群140的级别上执行反熵修复程序的参数。第二状态表是repair_sequence状态表,其定义了用于在集群140中所包括的内容服务器110上执行反熵修复程序的调度。该调度是节点执行节点修复的顺序或次序的形式。具体地,repair_sequence状态表跟踪集群140中的每个内容服务器110的状态,并指示在当前反熵修复程序期间哪些内容服务器110已执行了节点修复。第三状态表是repair_status状态表,其跟踪群集140中所包括的每个内容服务器110中所包括的特定要素的修复状态。如本文进一步所述,这些特定要素包括在对应的内容服务器110内经历修复的数据的分区和子范围中的每一者。第四状态表是repair_hook_status状态表,其跟踪由内容服务器110执行的与修复后程序相关联的操作的状态。与修复后程序相关联的此类操作在本文中被称为“修复后挂钩(post-repair hook)”。

[0035] 通常,一旦内容服务器110和所有相关的邻近内容服务器110已经执行了与反熵修复程序相关联的节点修复,则修复后挂钩包括由内容服务器110执行的各种维护任务。如本文所用的,给定内容服务器110的邻居是如下集群中所包括的其他内容服务器110的集合:该集合存储与给定内容服务器110共同的一个或多个分区。通常,在共享相同数据的所有其他内容服务器110已经执行节点修复之前,内容服务器110无法执行修复后挂钩。该规则防止相邻内容服务器110的节点修复所需的数据在这些修复完成之前被删除。在特定内容服务器110和所有相关邻居内容服务器110已经执行了节点修复之后,特定内容服务器110可以存储修复期间需要但一旦修复完成就不再需要的一个或多个分区。因此,清除挂钩删除这些不再需要的分区。

[0036] 此外,在特定内容服务器110和所有相关邻居内容服务器110已经执行了节点修复之后,特定内容服务器110可以低效方式存储经修复的数据。更具体地,在节点修复期间,内容服务器110从和向系统盘206的各种存储位置读取、写入和拷贝数据块。因此,一旦节点修复完成,特定分区的各部分可被存储在系统盘206上的随机位置中,和/或所述分区可包括嵌入在所述分区内的具有空闲或未使用的存储空间的各区段。检索以这种方式存储的分区可能是耗时的,因为系统盘206可能需要从彼此远离的存储位置检索数据,并且可能需要跳过具有空闲或未使用的存储空间的各区段。作为以这种低效方式存储数据的结果,相对于

在修复之前做出的请求,在修复完成之后从内容服务器110请求数据的终端设备115可能会经历增加的等待时间。为了减少访问等待时间,处理器204执行压缩过程以便以连续、线性方式将分区存储在系统盘206上,而没有具有空闲或未使用的存储空间的内部区段。具体地,处理器204执行压缩挂钩以便以更有效的方式存储数据,从而减少请求等待时间。

[0037] 此外,用户可定义任何附加的修复后挂钩,使得在特定内容服务器110和所有相关的邻居内容服务器110已经执行了节点修复之后,所述特定内容服务器110执行这些附加的修理后挂钩。一个此类用户定义的修复挂钩可以向监测应用程序(未示出)发送内容服务器110已完成节点修复的消息。以这种方式,监测应用程序跟踪正在执行修复的集群140中的每个内容服务器110的进程,并计算对应的度量,例如在集群140的修复期间耗费的时间。

[0038] 修复应用程序219在分布式数据库网络中的各内容服务器110上执行,其中没有内容服务器110被预先指定为反熵修复协调器。在一些实施例中,集群140中的每个内容服务器110周期性地(例如每两分钟一次)执行修复应用程序219。当执行修复应用程序219时,内容服务器110首先从repair_process状态表和/或repair_sequence状态表中检索数据,以确定反熵修复程序是否已经在进行中。如果来自repair_process状态表和/或repair_sequence状态表的数据指示反熵修复程序正在进行中,则内容服务器110根据repair_sequence状态表中的数据确定该内容服务器110是否按顺序下一个执行节点修复。如果repair_sequence状态表中的数据指示内容服务器110是按顺序的下一个,则内容服务器110修复复制品和相关联的状态表并退出修复程序。如果repair_sequence状态表中的数据指示内容服务器110不是按顺序的下一个,则内容服务器110确定是否允许内容服务器110执行与内容服务器110相关的修复后挂钩。如果repair_sequence状态表中的数据指示内容服务器110和所有邻居内容服务器110已完成节点修复,则允许内容服务器110执行修复后挂钩。如果内容服务器110被允许,则内容服务器110执行修复后挂钩并退出修复程序。否则,内容服务器110在不执行修理后挂钩的情况下退出修复程序。

[0039] 另一方面,如果内容服务器110确定反熵修复程序不是正在进行中,则内容服务器110确定是否应启动新的反熵修复程序。通常,如果先前的反熵修复程序已完成并且对启动新的反熵修复程序没有任何限制,则应启动新反熵修复程序。例如,新反熵修复程序的启动可以被限制为一天中被指定为非高峰小时的时间,其中非高峰小时表示预期内容服务器110上的负载低于特定阈值水平时的规定时间范围。如果当前时间落入被指定用于修复操作的规定时间范围内,则可以启动新反熵修复程序。如果不应启动新反熵修复程序,则内容服务器110退出修复程序。否则,内容服务器110尝试启动新反熵修复程序。如果内容服务器110可获取对repair_procedure状态表中的数据的锁定,则该尝试成功。如果内容服务器110未能成功获取对repair_procedure状态表中的数据的锁定,则另一内容服务器110已获取该锁定并且正在启动新反熵修复程序。然而,如果内容服务器110成功获取了所述锁定,则内容服务器110用集群级别参数填充repair_process状态表。另外,内容服务器110用内容服务器110执行新反熵修复程序的新序列填充repair_sequence状态表。然后,内容服务器110退出修复程序。

[0040] 通常,内容服务器110彼此协调的唯一时间是当内容服务器110尝试启动新反熵修复程序、填充上述repair_process和repair_sequence状态表并生成限定内容服务器110执行与新反熵修复程序有关的节点修复的顺序或次序的新序列时。当内容服务器110尝试启

动新反熵修复程序时,内容服务器110请求对集群级别的repair_process状态表的锁定。以这种方式,防止多个内容服务器110同时写入用于下一个反熵修复程序的新的repair_process状态表和repair_sequence状态表。因此,集群140内的任何内容服务器110都可获取对repair_process状态表的锁定并启动下一个反熵修复程序。

[0041] 当内容服务器110获取所述锁定时,内容服务器110将逐机器的repair_sequence状态表作为批式操作写入。在批式操作写入中,所有写入均成功或所有写入均不执行。在批式操作中,防止内容服务器110将部分序列写入repair_sequence状态表并随后故障,从而导致包括与多个反熵修复程序有关的部分序列的repair_sequence状态表。此外,防止其他内容服务器110在批式操作完成或失败之前访问repair_sequence状态表。因此,其他内容服务器110被确保了新的repair_sequence状态表是完整且一致的。通过以批式操作写入repair_sequence状态表,可以防止其他内容服务器110读取部分写入的repair_sequence状态表并因此以错误顺序执行节点修复。

[0042] 简而言之,任何内容服务器110都可经由两个协调状态启动新反熵修复程序:(1)获取对repair_process状态表的锁定并写入repair_process状态表;以及(2)以批式操作写入repair_sequence状态表。除了这两个协调状态之外,内容服务器110独立地执行节点修复,而无需内容服务器110彼此协调或获取锁定以便继续进行。

[0043] 此外,每个内容服务器110将被调度用于修复的分区的集合划分为子范围。给定的子范围可表示单独分区、多个分区、或分区的一部分。每个内容服务器110自动将分区分割并合并为子范围,以满足完成修复的目标完成时间。作为一个示例,反熵修复程序的目标可为每个子范围在不超过三十分钟内完成修复。基于分区的数目和存储在内容服务器110中的分区的大小,自动调整每个子范围的大小以满足该目标完成时间。通常,用于包括大量小分区的集群140中的内容服务器110的repair_status状态表的最佳集合与用于包括少量大分区的集群140中的内容服务器110的repair_status状态表的最佳集合不同。

[0044] 为了优化repair_status状态表的集合,内容服务器110执行自适应分割范围程序。这将基于分区数量、分区大小和其他度量自动选择子范围。作为一个示例,考虑包括一百万个分区的群集140,其中每个分区是键值对。为了找到跨多个内容服务器110的分区差异,集群140中所包括的每个内容服务器110计算每个部件的密码高速缓存并生成高速缓存树,在此称为“Merkle树”。在Merkle树中,每片叶子都包括对应分区或分区的一部分中的数据的哈希值。在给定两个Merkle树的情况下,其中一个Merkle树用于两个内容服务器110中的每一者上的分区集合,如果两个Merkle树上的对应叶子具有相同的哈希值,则对应的源数据相同。每个内容服务器110跨内容服务器110分析Merkle树,以找到存储在一个Merkle树中的哈希值与存储在另一Merkle树中的哈希值之间的差异。此类差异对应于由哈希值表示的源数据中的差异。然而,为一百万个分区生成单个Merkle树所消耗的内存可能比可用于执行反熵修复程序的内存更多。

[0045] 解决此问题的一种方式是将Merkle树的大小限制为合理数量的哈希值。然而,以这种方式限制Merkle树的大小可导致其中每个哈希值表示大量分区的Merkle树。如果两个对应的哈希值不同,则需要比较所述哈希值所表示的所有分区,以便找到数据不一致。

[0046] 因此,内容服务器110执行自适应分割范围程序,由此将节点修复分割为多个子范围,每个子范围具有不同的Merkle树。每个Merkle树可具有一对一的分辨率,由此每个哈希

值表示单独分区。并发地计算多个Merkle树。自适应分割范围程序基于每个Merkle树的目标完成时间(例如三十分钟)来确定每个Merkle树的数量、大小和分辨率。

[0047] 此自适应分割范围程序基于群集140中的基础分区自动调整Merkle树。例如,第一集群140可具有一百万个分区,其中每个分区是小的。该集群140将快速地修复,因为可快速计算每个哈希值,并且可以快速比较每个分区。第二集群140可具有一百万个分区,其中每个分区是一个千兆字节。该集群140将缓慢地修复,因为计算每个哈希值可耗费很长时间,并且比较每个分区同样可耗费很长时间。通过自适应地选择适当的分割范围,可以实现每个子范围三十分钟的目标完成时间。分割范围的这种自适应选择允许在内容服务器110发生故障之后较快地恢复。

[0048] 在一些实施例中,反熵修复程序可基于与一个或多个先前执行的反熵修复程序有关的历史数据来自动调整修复工作。在此类实施例中,如由每个内容服务器110所报告的,正在启动新反熵修复程序的内容服务器110分析repair_status状态表中的数据。该数据标识每个内容服务器110的每个子范围、该子范围的大小、以及修复该子范围所耗费的时间量。然后可以基于该数据来调适用于新的反熵修复程序的状态表,以便满足目标完成时间。作为一个示例,如果在比目标完成时间更短的时间内执行了在先反熵修复程序,则新反熵修复程序的分割范围可以基于如下目标完成时间:该目标完成时间是在先反熵修复程序的实际完成时间的两倍。另一方面,如果在比目标完成时间更长的时间内执行了在先反熵修复程序,则新反熵修复程序的分割范围可以基于如下目标完成时间:该目标完成时间是小于在先反熵修复程序的实际完成时间的固定百分比。以这种方式,用于后续反熵修复程序的配置适应于分区的性质随时间推移的变化。

[0049] 现在描述一组修复选项。这些修复选项提供用户控制以指导反熵修复程序的配置和执行,包括如何执行自适应分割范围程序。修复选项在下表1中显示:

[0050]

选项	说明	默认
类型	要运行的修复的类型。选择是完全和增量。	完全
工作者	运行修复的修复工作者的数量。在调整并行性之前，调整该参数以提高修复速度。	最大（1，核心数量/2）
并行性	计算 Merkle 树中的并行性程度。选择是顺序的、并行的、dc_parallel。	顺序的

挂钩	一旦内容服务器和所有邻居完成修复后，应在内容服务器上执行的一组修复后挂钩。可用的修复后挂钩包括清除、压缩和附加可插拔选项。	[“清除”]
split_range（分割_范围）	出于子范围的目的，将 repair_status 状态表分割成的分区数目。值可为整数值，<integer>_dry_run 或自适应。如果是<integer>_dry_run，则自适应算法在 repair_status 状态表的配置图中包括 split_range_adaptive 字段，以在启用自适应功能之前进行检查。如果是自适应的，则群集自动确定最佳值。	自适应
interrepair_delay_minutes（修复间_延迟_分钟）	修复运行之间的延迟，以分钟为单位。默认值取决于所调度的修复类型。使用此项以使修复彼此“抵消”，例如，如果已知修复耗费 2 天，并且目标是每 10 天完成一次修复，则将延迟设置为约 3 天可为良好的值。	完全->1440（24 小时）增量->0
process_timeout_seconds（过程_超时_秒）	等待另一个内容服务器进行状态转换（例如，生成序列、开始修复、变得健康等……）的秒数。如果观察到许多取消，请设置此项。	1800（30 分钟）
repair_timeout_seconds（修复_超时_秒）	等待单个子范围修复完成的秒数。如果观察到许多取消，请设置此项。	14400（4 小时）

[0051]

[0052] 表1

[0053] “类型”修复选项指定要执行的反熵修复程序的类型。在完全修复中，内容服务器110对所有分区执行修复。在增量修复中，内容服务器110仅对已改变的数据执行修复。默认修复类型是完全修复。

[0054] “工作者”修复选项指定了在其上执行修复的处理器核心(在本文中也称为“工作者”)的数量。工作者的默认数量是一个核心和可用核心的数量除以2的最大值。此默认数量的工作者在至少一个处理器核心上执行修复,但不超过可用核心数量的一半。

[0055] “并行性”修复选项指定在计算用于执行修复的Merkle树时采用的并行性程度。对于在三个地理上不同的数据中心中的每一者中包括三个内容服务器110的群集140,顺序的使得九个内容服务器110中的每一者顺序地建立Merkle树,其中在任何给定时间仅一个内容服务器110建立Merkle树。并行的使给定数据中心中的三个内容服务器110同时建立Merkle树。数据中心顺序地建立Merkle树,其中在任何给定时间仅一个数据中心中的内容服务器110建立Merkle树。Dc_parallel使所有三个数据中心上的所有九个内容服务器110同时建立Merkle树。默认的并行性是顺序的。尽管顺序的可能是这三个选项中最慢的,但就数据保存而言,顺序的也是最保守的。

[0056] “挂钩”修复选项指定:一旦对应的内容服务器110和邻居内容服务器110已完成节点修复就执行的修复后挂钩的集合。可用的修复后挂钩包括“清除”、“压缩”以及任何其他用户供应的修复后挂钩。默认的挂钩是清除。

[0057] “split_range”修复选项指定出于子范围的目的,将repair_status状态表分割成的分区的数量。如果split_range指定整数‘n’,则执行节点修复,使得每个分区均分割为‘n’个子范围。如果split_range指定后接“_dry_run”的整数‘n’,则执行节点修复,使得每个分区都被分割为‘n’个子范围,并将附加诊断数据存储在repair_status状态表中。如果split_range指定“自适应”,则节点修复执行自适应分割范围过程,如本文进一步所述。默认的split_range是自适应的。

[0058] “interrepair_delay_minutes”修复选项指定在完成一个反熵修复程序与启动后续反熵修复程序之间的延迟的整数分钟数。interrepair_delay_minutes值为“完全”指示在完成在先反熵修复程序之后1440分钟(24小时)发生后续修复的启动。interrepair_delay_minutes的值为“增量”指示在完成在先反熵修复程序之后立即无延迟地发生后续反熵修复程序的启动。

[0059] “process_timeout_seconds”修复选项指定等待另一内容服务器110从一种状态转换为另一种状态的整数秒数。默认的process_timeout_seconds是1800秒(30分钟)。

[0060] “repair_timeout_seconds”修复选项指定等待单个子范围修复完成的整数秒数。默认的repair_timeout_seconds是14,400秒(4小时)。

[0061] 在各种实施例中,在本文中被称为“nodetool”的两个补充命令可用于检查或改变当前执行的修复程序的状态。这两个补充命令包括:用于检查内容服务器110的修复历史的repairstatus命令,以及用于控制集群140的修复的repairctl命令。现在将进一步详细讨论这些nodetool命令中的每一者。

[0062] repairstatus命令的结构如下表2中所示:

[0063] 001 名称

[0064] 002 nodetool repairstatus—打印修复历史信息

[0065] 003 总览

[0066] 004 nodetool...repairstatus

[0067] 005 [--repair-id<repairid>][--node-id<nodeid>]

[0068] 006 [--keyspace<keyspace>][--table<table>]

[0069] 007 选项

[0070] 008 --repair-id<repairid>

[0071] 009 仅显示指定修复标识符的修复状态。

[0072] 010 如果未指定,则显示最新的修复状态。

[0073] 011 --node-id<nodeid>

[0074] 012 要获取修复状态的节点标识符

[0075] 013 --keyspace<keyspace>

[0076] 014 限制到指定的键空间

[0077] 015 --table<table>

[0078] 016 限制到指定的状态表

[0079] 表2

[0080] 001行和002行将名称“nodetool repairstatus”指定为打印修复历史信息的命令。003行至006行说明了repairstatus命令的总览,包括如在005行至006行中显示并且在007行至016行中更全面描述的四个命令选项。008行至010行说明了repair-id命令选项。如果指定了repair-id,则repairstatus命令将返回指定修复标识符的状态。如果未指定repair-id,则repairstatus命令将返回整个修复的状态。011行至012行说明了node-id命令选项。repairstatus命令返回指定节点标识符的状态。013行至014行说明了keyspace命令选项。repairstatus命令返回指定键空间的状态。015行至016行说明了table命令选项。repairstatus命令返回指定状态表的状态。

[0081] 例如,不带任何命令选项的命令“nodetool repairstatus”将返回最新修复状态的全局视图。命令“nodetool repairstatus--repair-id 12”将返回标识符为12的修复的修复状态。命令“nodetool repairstatus--node-id 73ab7e49”将返回标识符为73ab7e49的节点的修复状态。

[0082] repairctl命令的结构在下表3中显示:

[0083] 001 名称

[0084] 002 nodetool repairctl—控制集群上的修复

[0085] 003 总览

[0086] 004 nodetool...repairctl

[0087] 005 [-stop-cluster][-start-cluster]

[0088] 006 [-cancel-running]

[0089] 007 选项

[0090] 008 -stop-cluster

[0091] 009 暂停整个集群上的修复。

[0092] 010 这不会取消活跃执行的修复

[0093] 011 -cancel-execution

[0094] 012 立即中止给定内容服务器上的任何正在执行的修复。

[0095] 013 -start-cluster

[0096] 014 恢复整个群集上的修复。

[0097] 表3

[0098] 001行和002行将名称“nodetool repairctl”指定为控制集群140上的修复的命令。003行至006行说明了repairctl命令的总览,包括如在005行至006行中所显示并且在007行至014行中更全面描述的三个命令选项。008行至010行说明了stop-cluster命令选项。如果存在该选项,则stop-cluster选项使整个集群140上的修复暂停,而不会取消活跃修复的执行。011行至012行说明了cancel-execution命令选项。cancel-execution立即中止给定内容服务器110上任何正在执行的修复。013行至014行说明了start-cluster命令选项。start-cluster命令恢复群集140上的修复。

[0099] 例如,命令“nodetool repairctl--stop-cluster”将暂停群集140上的修复,而不取消任何活跃修复。命令“nodetool repairctl--start-cluster”将恢复暂停的群集140上的修复。命令“nodetool repairctl--cancel-execution”将中止给定内容服务器110上任何修复的执行。作为结果,剩余的内容服务器110将开始修复。该命令选项将使被卡住的内容服务器110停止阻止其他内容服务器110执行节点修复。命令“nodetool repairctl--stop-cluster--cancel-execution”将使群集140上的修复暂停,并中止给定内容服务器110上任何修复的执行。

[0100] 在一些实施例中,反熵修复程序可支持排他性的修复调度。在此类实施例中,修复可被限制为仅在特定时间段期间启动和/或执行。例如,集群140中的每个数据中心可指定“非高峰”小时,所述“非高峰”小时表示被指定用于修复操作的规定时间范围。如果当前时间落入被指定用于修复操作的规定时间范围内,则可以启动新的反熵维修程序。通常,本文所述的技术不会消耗过多的CPU资源、内存或网络带宽。即使这样,一些用户可能还希望将修复限制于非高峰小时,以最小化对其他数据请求和网络流量的影响。在其他实施例中,用户可能能够使修复或一个或多个修复后挂钩暂停指定的时间段。

[0101] 在一些实施例中,反熵修复程序可支持多个修复调度。在此类实施例中,修复可以是不同类型的并且可并发地执行。repair_sequence可包括附加字段,所述附加字段基于修复类型指定要应用于当前修复的修复类型和/或配置选项。作为一个示例,可以支持完全修复和增量修复。在完全修复中,内容服务器110对所有分区执行修复。在增量修复中,内容服务器110仅对已改变的数据执行修复。因此,完全修复可以每月执行一次,而增量修复可以每天执行一次。每次增量修复都会修复由于写入而不同的分区。另外,每次完全修复将修复由于数据校正而不同的分区。在这些实施例中,两个修复序列可以同时但以不同的配置执行。因此,完全修复和增量修复可以同时执行,只要给定内容服务器110在给定时间执行一个节点修复即可。

[0102] 在一些实施例中,大集群140的反熵修复程序可以被执行为高度并发修复,其中两个或更多个内容服务器110可以针对相同的反熵修复程序并发地执行节点修复,在本文中被称为“漂移(drifted)”或“并发”修复。在此类实施例中,不共享任何范围的多个不相交的内容服务器110并行地执行节点修复。在内容服务器110确定内容服务器110是否按顺序为下一个时,内容服务器110可以可替代地确定进行至节点修复不会负面地影响任何相邻的内容服务器110。

[0103] 在一些实施例中,内容服务器110可获取对repair_process状态表的锁定,但是可能无法继续生成repair_sequence状态表来启动下一个反熵修复程序。在此类实施例中,群

集140中的一个或多个其他内容服务器110可监测尝试获取锁定的内容服务器110的进程。如果尝试获取锁定的内容服务器110未能在可配置的超时期间(例如三十分钟)内生成新的repair_sequence状态表,则另一个内容服务器110可以启动下一个反熵修复程序并取消失败的反熵修复程序。

[0104] 在一些实施例中,内容服务器110可以确定内容服务器110按顺序为下一个,但是可能无法执行节点修复。在此类实施例中,内容服务器110可以连续地生成心跳消息到repair_sequence状态表中的对应行并对其监测,以确保内容服务器110在节点修复中取得进展。其他内容服务器110也可监测心跳消息。如果当前尝试进行节点修复的内容服务器110在可配置的超时期间(例如三十分钟)内未更新心跳消息,则在repair_sequence状态表中为下一个的另一内容服务器110可取消卡住的内容服务器110的所有正在进行的修复。然后,该内容服务器110在repair_sequence状态表中将卡住的内容服务器110的状态标记为已取消(CANCELLED),并继续进行节点修复。

[0105] 在一些实施例中,给定的内容服务器110在执行节点修复时可能耗费过多的时间量和CPU资源,或者在执行节点修复时可能产生过量的网络流量。为了解决该问题,内容服务器110执行自适应分割范围功能,以将修复工作自适应地划分为适应于内容服务器110上的分区的大小和数量的工作子范围。调整子范围的大小,使得每个子范围在可配置的超时期间(例如三十分钟)内完成。如果对特定子范围的修复超过超时期间达指定量,则可以取消并重新调度对该子范围的修复。

[0106] 在一些实施例中,当内容服务器110正在执行节点修复时,数据库可以重启。在此类实施例中,一旦数据库重启,内容服务器110就可以恢复节点修复。由于修复的每个子范围的大小被确定为在可配置的超时期间(例如三十分钟)内完成,因此使由于数据库重启而丢失的工作量最小化。一旦数据库重启,在重置之前处于启动(STARTED)状态的内容服务器110转换到repair_sequence状态表中的已取消(CANCELLED)状态,并且在与数据库重置发生时相同的状态表和/或相同的子范围处恢复修复。此过程确保了节点修复完整完成。

[0107] 在一些实施例中,当执行修复后挂钩时,内容服务器110可能被卡住。通常,无需由于修复后挂钩无法完成而推迟修复。因此,可以以要施加至处于REPAIR_HOOK_RUNNING状态下的内容服务器110的积极超时时间段解决该问题。如果内容服务器110花费在执行修复后挂钩上的时间量超过了超时时间段,则内容服务器110可以被取消并重启。

[0108] 在一些实施例中,可以在执行修复期间添加内容服务器110。因为新的内容服务器110不影响当前修复的顺序,所以当前修复继续以现有顺序执行。然后,后续修复将新的内容服务器110添加到下一个repair_sequence状态表。

[0109] 在一些实施例中,内容服务器110可以在修复期间被终止或变得不可用。在此类实施例中,一个或多个内容服务器110可监测集群140中其他内容服务器110的健康。如果给定内容服务器110的子范围在可配置的超时期间(例如三十分钟)内不可用,则将内容服务器110的状态设置为“失败(FAILED)”,然后继续进行修复。

[0110] 图3是根据本发明的各种实施例的可以结合图1的网络基础设施100实现的控制服务器120的框图。如图所示,控制服务器120包括但不限于处理器304、系统盘306、输入/输出(I/O)设备接口308、网络接口310、互连件312和系统存储器314。

[0111] 处理器304被包括以代表单个CPU、多个CPU、具有多个处理核心的单个CPU等。处理

器304被配置为检索并执行存储在系统存储器314中的编程指令,例如作为控制应用程序317。类似地,处理器304被配置为存储应用程序数据(例如,软件库)并从系统存储器314和存储在系统盘306中的数据库318中检索应用程序数据。互连件312被配置为促进处理器304、系统盘306、I/O设备接口308、网络接口310和系统存储器314之间的数据传输。I/O设备接口308被配置为经由互连件312在I/O设备316与处理器304之间传输输入数据和输出数据。系统盘306可包括一个或多个硬盘驱动器、固态存储设备等。系统盘306被配置为存储具有与内容服务器110、(一个或多个)填充源130和文件218相关联的信息的数据库318。

[0112] 系统存储器314包括控制应用程序317,该控制应用程序317被配置为访问存储在数据库318中的信息并处理该信息以确定将在网络基础设施100中所包括的内容服务器110上复制特定文件218的方式。控制应用程序317还可被配置为接收和分析与内容服务器110和/或终端设备115中的一者或多者相关联的性能特征。

[0113] 图4是根据本发明的各种实施例的可以结合图1的网络基础设施100实现的终端设备115的框图。如图所示,终端设备115可包括但不限于处理器410、图形子系统412、I/O设备接口414、大容量存储单元416、网络接口418、互连件422和存储器子系统430。

[0114] 处理器410被包括以代表单个CPU、多个CPU、具有多个处理核心的单个CPU等。在一些实施例中,处理器410被配置为检索并执行存储在存储器子系统430中的编程指令。类似地,处理器410被配置为存储和检索驻留在存储器子系统430中的应用程序数据(例如,软件库)。互连件422被配置为促进处理器410、图形子系统412、I/O设备接口414、大容量存储装置416、网络接口418和存储器子系统430之间的数据(例如编程指令和应用程序数据)传输。

[0115] 在一些实施例中,图形子系统412被配置为生成视频数据的帧并将所述视频数据的帧发送到显示设备450。在一些实施例中,图形子系统412可以与处理器410一起集成到集成电路中。显示设备450可以包括用于生成供显示的图像的任何技术上可行的装置。例如,可以使用液晶显示(liquid crystal display,LCD)技术、阴极射线技术和发光二极管(light-emitting diode,LED)显示技术来制造显示设备450。输入/输出(I/O)设备接口414被配置为从用户I/O设备452接收输入数据,并且经由互连件422将所述输入数据发送到处理器410。例如,用户I/O设备452可包括一个或多个按钮、键盘和鼠标或其他指点设备。I/O设备接口414还包括音频输出单元,该音频输出单元被配置为生成电音频输出信号。用户I/O设备452包括扬声器,该扬声器被配置为响应于电音频输出信号而产生声学输出。在替代实施例中,显示设备450可包括扬声器。电视机是本领域中已知的可显示视频帧并生成声学输出的设备的示例。

[0116] 大容量存储单元416,例如硬盘驱动器或闪存存储驱动器,被配置为存储非易失性数据。网络接口418被配置为经由网络105发送和接收数据分组。在一些实施例中,网络接口418被配置为使用众所周知的以太网标准进行通信。网络接口418经由互连件422耦合到处理器410。

[0117] 在一些实施例中,存储器子系统430包括编程指令和应用程序数据,所述存储器子系统包括操作系统432、用户界面434和播放应用程序436。操作系统432执行系统管理功能,例如管理包括网络接口418、大容量存储单元416、I/O设备接口414和图形子系统412在内的硬件设备。操作系统432还为用户界面434和播放应用程序436提供处理和存储器管理模型。用户界面434(例如窗口和对象隐喻)为用户与终端设备108的交互提供了机制。本领域技术

人员将认识本领域中众所周知并且适合于结合到终端设备108中的各种操作系统和用户界面。

[0118] 在一些实施例中,播放应用程序436被配置为经由网络接口418从内容服务器105请求和接收内容。此外,播放应用程序436被配置为经由显示设备450和/或用户I/O设备452来解释内容并呈现内容。

[0119] 图5A至图5B阐述了根据本发明的各个实施例的用于在分布式数据库网络中的复数个内容服务器110上执行分布式反熵修复程序的方法步骤的流程图。尽管结合图1至图4的系统描述了方法步骤,但是本领域普通技术人员应理解,被配置为以任何次序执行方法步骤的任何系统都在本发明的范围内。

[0120] 如图所示,方法500开始于步骤502处,在该步骤处在内容服务器110上执行的修复应用程序219确定是否存在当前正在进行的反熵修复程序。更具体地,修复应用程序219读取集群范围内的repair_process状态表,以确定最新的反熵修复程序是显示完成还是进行中的状态。

[0121] 如果反熵修复程序当前正在进行中,则方法500行进至步骤504,在该步骤处修复应用程序219确定内容服务器110是否如由repair_sequence状态表所指示按顺序为下一个执行修复的内容服务器。如果repair_sequence状态表中显示的所有先前内容服务器110的状态为失败(FAILED)或完成(FINISHED),则内容服务器110按顺序为下一个。在此类情况下,方法500行进至步骤506,在该步骤处修复应用程序219对存储在内容服务器110上的分区执行修复。更具体地,修复应用程序219对存储在内容服务器110上的每个分区执行修复。如果分区已被分割成多个子范围,则修复应用程序219一次执行一个子范围的修复。在修复应用程序219完成对存储在内容服务器110上的所有分区和子范围的修复之后,则方法500终止。

[0122] 然而,如果在步骤504处,内容服务器110不是按顺序的下一个,则方法行进至步骤508,在该步骤处修复应用程序219确定当前内容服务器110和当前内容服务器110的所有邻居内容服务器110已经完成了相应的修复。如果当前内容服务器110或当前内容服务器110的至少一个邻居内容服务器110尚未完成修复,则方法终止。然而,如果当前内容服务器110和所有邻居内容服务器110已经完成了相应的修复,则方法500行进至步骤510,在该步骤处修复应用程序219执行修复后挂钩。如本文中进一步描述的,此类修复后挂钩执行清除和/或压缩操作以及其他维护任务,例如向监测应用程序发送当前内容服务器110已完成修复的通知。然后,方法500终止。

[0123] 返回到步骤502,如果反熵修复程序当前不是正在进行,则方法500行进至步骤512,在该步骤处修复应用程序219确定新反熵修复程序是否准备好开始。具体地,修复应用程序219确定在启动新反熵修复程序方面没有附加限制,例如两次连续的反熵修复程序之间的最小时间间隔或一天中将反熵修复程序限制于非高峰小时的时间。如果新反熵修复程序尚未准备好开始,则方法500终止。然而,如果新反熵修复程序准备好开始,则该方法行进至步骤514,在该步骤处修复应用程序219尝试获取对repair_process状态表的锁定。如果修复应用程序219未能成功获取对repair_process状态表的锁定,则方法500终止。然而,如果修复应用程序219成功获取了对repair_process状态表的锁定,则方法500行进至步骤518,在该步骤处修复应用程序219生成下一个反熵修复程序的repair_sequence状态表。修

复应用程序219以批式操作存储repair_sequence状态表。然后,方法500终止。

[0124] 另外,如本文中进一步描述的,集群140中的每个内容服务器110周期性地(例如每两分钟一次)执行方法500的步骤。

[0125] 总之,反熵修复程序在分布式数据库网络中的各节点上执行,其中没有节点被预先指定为反熵修复协调器。每个节点周期性地执行技术,其中所述节点首先确定反熵修复程序是否正在进行中。如果反熵修复程序正在进行中,则所述节点确定所述节点是否按顺序下一个执行修复。如果所述节点为按顺序的下一个,则所述节点修复复制品和相关联的状态表并退出。如果所述节点不是按顺序的下一个,则所述节点确定所述节点是否被允许执行与所述节点有关的修复后程序。如果所述节点被允许,则所述节点执行修复后程序并退出。否则,所述节点将退出,而不执行修复后程序。

[0126] 另一方面,如果所述节点确定反熵修复程序不是正在进行中,则所述节点确定是否应启动新反熵修复程序。如果不应启动新反熵修复,则所述节点将退出。否则,所述节点尝试启动新反熵修复程序,如果成功,则生成新序列,所述新序列定义节点执行与新反熵修复程序有关的节点修复的顺序次序。然后所述节点退出。

[0127] 相对于现有技术,所公开的技术的至少一个技术优点是,反熵修复程序自动地缩放到分布式数据库网络中的节点数量和复制品的变化大小。因此,相对于传统技术,反熵修复程序适应于存储在特定节点上的数据的每个子集,以便减少磁盘访问量、CPU资源和所消耗的网络带宽。此外,随着节点数量和复制品大小随时间推移变化,用户无需手动设计和维护反熵修复程序。

[0128] 相对于现有技术,所公开的技术的另一技术优势在于,反熵修复程序分布在各节点上。此外,一旦故障节点恢复执行,则反熵修复程序可以在与故障发生时相对相同的条件下恢复。因此,相对于现有技术途径,当一个或多个其他节点在修复期间发生故障时,很少或没有修复进程发生损失。这些技术优点代表相对于现有技术途径的一项或多项技术改进。

[0129] 1.在一些实施例中,一种计算机实现的方法包括:由复数个节点中所包括的第一节点在所述复数个节点中所包括的所有其他节点之前确定第一反熵修复程序已结束;以及由所述第一节点确定第二反熵修复程序准备好开始;生成用于执行与所述第二反熵修复程序相关的一个或多个操作的调度;以及将所述调度写入共享修复调度数据结构以在所述复数个节点中所包括的多个节点上启动所述第二反熵修复程序。

[0130] 2.根据条款1所述的计算机实现的方法,其中确定所述第二反熵修复程序准备好开始包括:确定当前时间在指定用于修复操作的规定时间范围内。

[0131] 3.根据条款1或条款2所述的计算机实现的方法,所述计算机实现的方法还包括:由所述复数个节点中所包括的第二节点确定第三反熵修复程序准备好开始;生成用于执行与所述第三反熵修复程序相关联的一个或多个操作的调度;以及将所述调度写入第二共享修复调度数据结构以在所述复数个节点中所包括的多个节点上启动所述第三反熵修复程序。

[0132] 4.根据条款1-3中任一项所述的计算机实现的方法,其中:所述第二反熵修复程序包括完全反熵修复程序,所述第三反熵修复程序包括增量式反熵修复程序,并且确定所述第三反熵修复程序准备好开始包括:确定包括增量式修复的第四反熵修复程序已结束。

[0133] 5.根据条款1-4中任一项所述的计算机实现的方法,所述计算机实现的方法还包括:由所述复数个节点中所包括的第二节点确定所述第二反熵修复程序正在进行中;确定所述第二个节点按顺序是下一个要被修复的;以及修复常驻于所述第二节点上的至少一个不一致数据分区。

[0134] 6.根据条款1-5中任一项所述的计算机实现的方法,所述计算机实现的方法还包括:由所述复数个节点中所包括的第二节点确定所述第二反熵修复程序正在进行中;确定所述第二节点独立于所述复数个节点中所包括的当前正在执行与所述第二反熵修复程序相关联的修复的所有其他节点;以及修复常驻于所述第二节点上的至少一个不一致数据分区。

[0135] 7.根据条款1至6中任一项所述的计算机实现的方法,所述计算机实现的方法还包括:由所述复数个节点中所包括的第二节点确定所述第二反熵修复程序正在进行中;确定所述第二节点已执行了与所述第二反熵修复程序相关联的修复;确定所述复数个节点中所包括的与所述第二节点相互依赖的所有其他节点已执行了与所述第二反熵修复程序相关联的修复;以及由所述第二节点执行与所述第二反熵修复程序相关联的修复后程序。

[0136] 8.根据条款1-7中任一项所述的计算机实现的方法,其中执行所述修复后程序包括:删除在所述第二反熵修复程序完成之后不再需要的分区。

[0137] 9.根据条款1至8中任一项所述的计算机实现的方法,其中执行所述修复后程序包括:对与所述第二节点相关联的一个或多个分区执行压缩操作以缩短访问所述一个或多个分区时的等待时间。

[0138] 10.根据条款1至9中任一项所述的计算机实现的方法,其中执行所述修复后程序包括发送消息给监测应用程序,所述消息指示所述第二节点已经执行了与所述第二反熵修复程序相关联的修复。

[0139] 11.根据条款1-10中任一项所述的计算机实现的方法,所述计算机实现的方法还包括:确定与所述第二反熵修复程序相关联的分区的数量超过阈值水平;以及将与所述第二反熵修复程序相关联的工作划分为复数个子范围。

[0140] 12.根据条款1-11中任一项所述的计算机实现的方法,所述计算机实现的方法还包括:确定与所述第二反熵修复程序相关联的一个或多个分区的大小超过阈值水平;以及将与所述第二反熵修复程序相关联的工作划分为复数个子范围。

[0141] 13.根据条款1-12中任一项所述的计算机实现的方法,所述计算机实现的方法还包括:确定所述第二反熵修复程序的完成时间超过阈值水平;以及基于所述完成时间将与所述第二反熵修复程序相关联的工作划分为复数个子范围。

[0142] 14.在一些实施例中,一个或多个非暂态计算机可读存储介质包括指令,所述指令当由一个或多个处理器执行时使所述一个或多个处理器执行以下步骤:确定第一反熵修复程序已结束;由复数个节点中所包括的第一节点在所述复数个节点中所包括的所有其他节点之前确定第二反熵修复程序准备好开始;生成用于执行与所述第二反熵修复程序相关的操作的调度;以及将所述调度写入共享修复调度数据结构以在所述复数个节点中所包括的多个节点上启动所述第二反熵修复程序。

[0143] 15.根据条款14所述的一种或多种非暂态计算机可读存储介质,其中确定所述第二反熵修复程序准备好开始包括:确定当前时间在指定用于修复操作的规定时间范围内。

[0144] 16.根据条款14或条款15所述的一种或多种非暂态计算机可读存储介质,其中所述复数个节点中所包括的每个节点顺序地执行与所述第二反熵修复程序相关联的修复。

[0145] 17.根据条款14-16中任一项所述的一种或多种非暂态计算机可读存储介质,其中:所述复数个节点中所包括的第一节点子集中所包括的各个节点彼此并行地执行与所述第二反熵修复程序相关联的修复;所述复数个节点中所包括的第二节点子集中的各个节点彼此并行地执行与所述第二反熵修复程序相关联的修复;并且所述第一节点子集中所包括的节点与所述第二节点子集中所包括的节点顺序地执行与所述第二反熵修复程序相关联的修复。

[0146] 18.根据条款14-17中任一项所述的一种或多种非暂态计算机可读存储介质,其中所述复数个节点中所包括的每个节点彼此并行地执行与所述第二反熵修复程序相关联的修复。

[0147] 19.在一些实施例中,一种计算设备包括:存储器,所述存储器包括指令;以及处理器,所述处理器耦合至所述存储器,并且当执行指令时被配置为:确定第一反熵修复程序已结束;由所述复数个节点中所包括的第一节点在所述复数个节点中所包括的所有其他节点之前确定第二反熵修复程序准备好开始;生成与所述第二反熵修复程序相关联的修复调度;以及将所述修复调度写入数据存储装置以在所述复数个节点中所包括的多个节点上启动第二反熵修复程序。

[0148] 20.根据条款19所述的计算设备,其中所述处理器当执行所述指令时还被配置为:在所述复数个节点中所包括的第二节点处指派多个处理核心以执行与所述第二反熵修复程序相关联的修复,其中所述处理核心的数量最大为所述第二节点内可用的处理器核心的数量的一半。

[0149] 以任何方式在任何权利要求中记载的任何权利要求要素和/或在本申请中描述的任何要素的任何和所有组合都落入本发明的预期保护范围内。

[0150] 已经出于说明的目的给出了各种实施例的描述,但是这些描述并不旨在是穷举性的或限于所公开的实施例。在不脱离所描述的实施例的范围和精神的情况下,许多修改和变型对于本领域普通技术人员将是显而易见的。

[0151] 本发明实施例的各方面可以体现为系统、方法或计算机程序产品。因此,本公开的各方面可以采取全硬件实施例、全软件实施例(包括固件、驻留软件、微代码等)或组合软件方面和硬件方面的实施例的形式,这些软件方面和硬件方面通常可以在本文中统称为“模块”或“系统”。此外,本公开的各方面可以采取计算机程序产品的形式,所述计算机程序产品体现为其上体现有计算机可读程序代码的(一个或多个)计算机可读介质。

[0152] 可以利用(一个或多个)计算机可读介质的任何组合。所述计算机可读介质可为计算机可读信号介质或计算机可读存储介质。计算机可读存储介质可以是例如但不限于电、磁、光学、电磁、红外或半导体系统、装置或设备,或前述的任何合适的组合。计算机可读存储介质的更具体示例(非穷举列表)将包括以下:具有一根或多根导线的电连接、便携式计算机软磁盘、硬盘、随机存取存储器(RAM)、只读存储器(ROM)、可擦除可编程只读存储器(EPROM或闪存)、光纤、便携式光盘只读存储器(CD-ROM)、光学存储设备、磁性存储设备,或前述的任何合适的组合。在本文的上下文中,计算机可读存储介质可以是任何有形介质,所述有形介质可包含或存储供指令执行系统、装置或设备使用或与其结合使用的程序。

[0153] 上面参考根据本公开的实施例的方法、装置(系统)和计算机程序产品的流程图图示和/或框图描述了本公开的各方面。应当理解的是,流程图图示和/或框图的每个框以及流程图图示和/或框图中的各框的组合可以由计算机程序指令来实现。可以将这些计算机程序指令提供给通用计算机、专用计算机或其他可编程数据处理装置的处理器以产生机器,使得经由计算机或其他可编程数据处理装置的处理器执行的指令允许实现在流程图和/或框图的一个或多个框中指定的功能/动作。此类处理器可为但不限于通用处理器、专用处理器(special-purpose processor)、应用程序专用处理器(application-specific processor)或现场可编程处理器。

[0154] 附图中的流程图和框图示出了根据本公开的各种实施例的系统、方法和计算机程序产品的可能实现的架构、功能性和操作。就此而言,流程图或框图中的每个框可代表代码的模块、区段或部分,所述模块、区段或部分包括用于实现一种或多种指定的逻辑功能的一个或多个可执行指令。还应注意的是,在一些替代实施例中,框中指出的功能可以不按图中指出的次序发生。例如,取决于所涉及的功能性,连续示出的两个框实际上可以基本上同时执行,或者所述框有时可以以相反的次序执行。还应注意的是,框图和/或流程图图示的每个框以及框图和/或流程图图示中各框的组合可以由执行指定功能或动作的基于专用硬件的系统或专用硬件和计算机指令的组合来实现。

[0155] 虽然前述内容是针对本公开的实施例,但是可以在不脱离本公开的基本范围的情况下设计出本公开的其他和进一步的实施例,并且本公开的所述范围由随后的权利要求来确定。

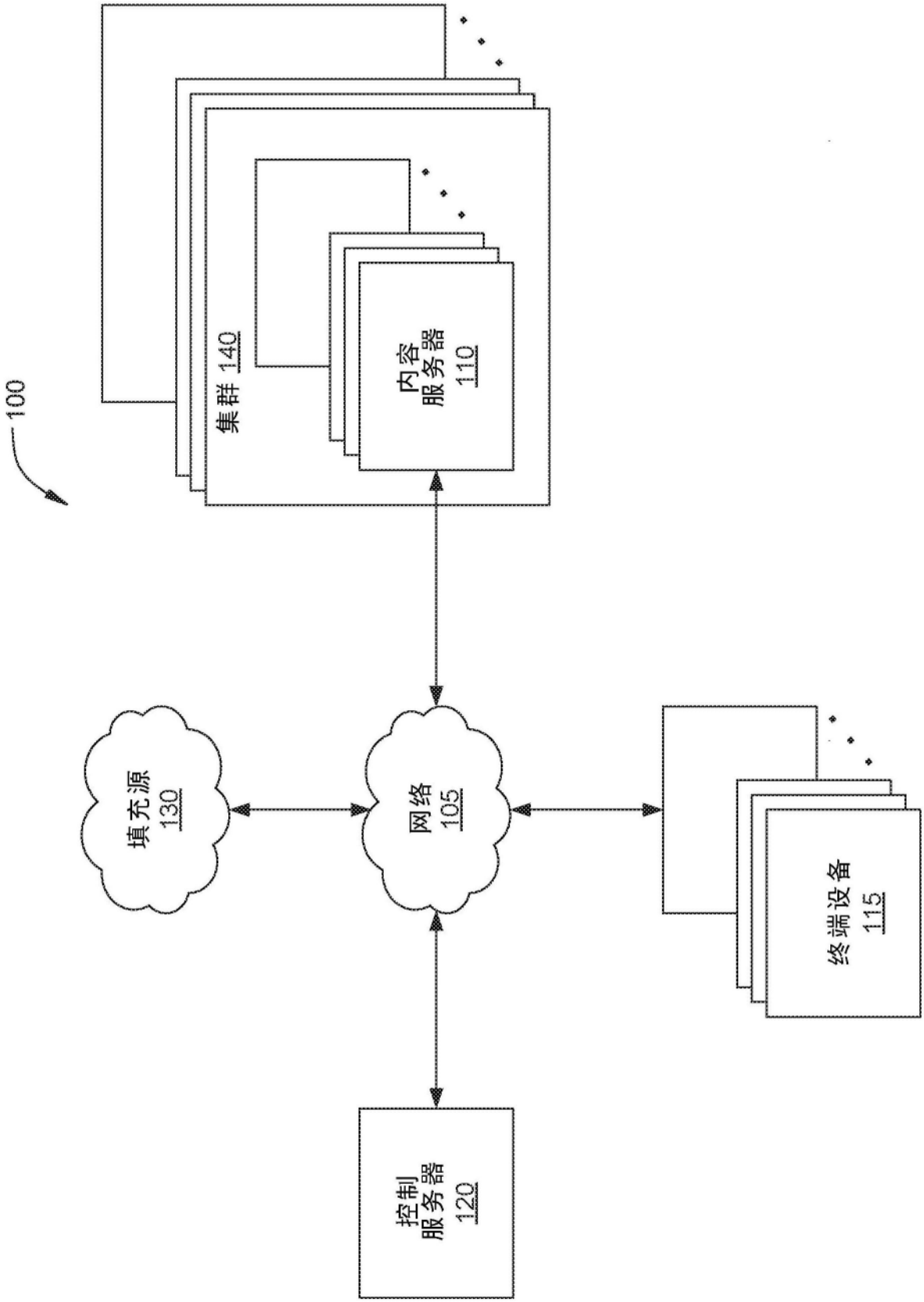


图1

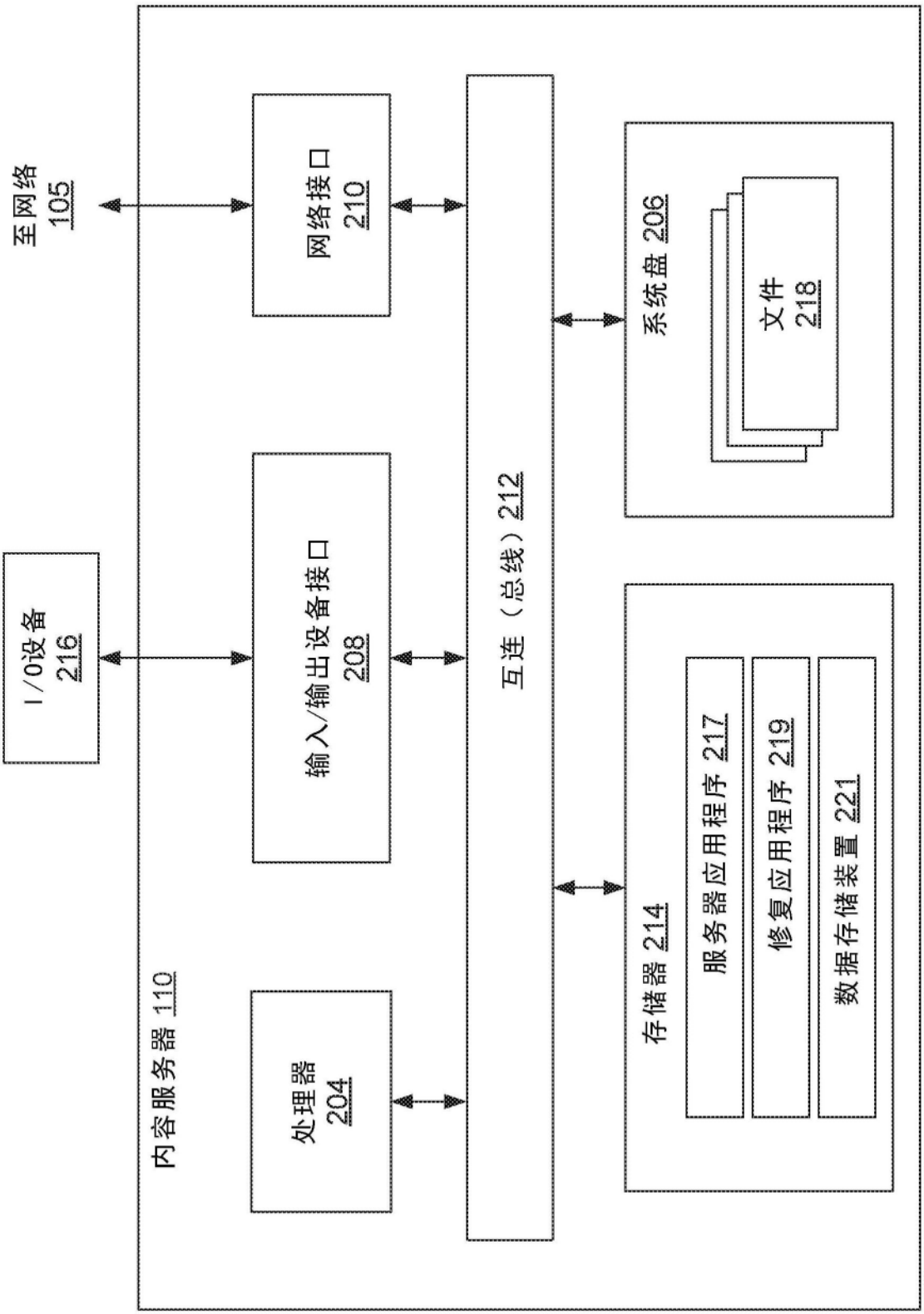

至网络
105
网络接口
210
系统盘 206
文件
218
I/O设备
216
输入/输出设备接口
208
互连（总线）212
内容服务器 110
处理器
204
存储器 214
服务器应用程序 217
修复应用程序 219
数据存储装置 221

图2

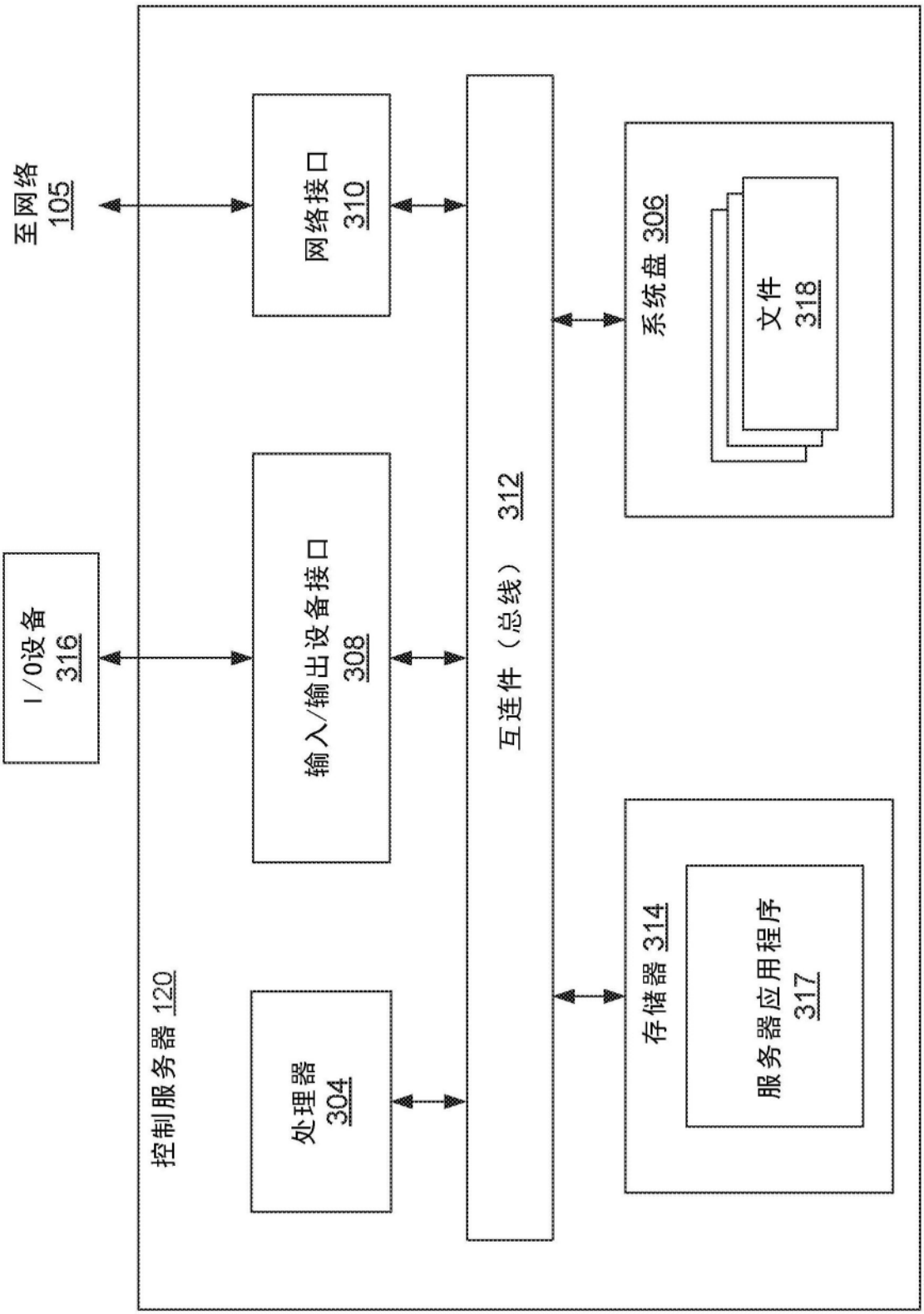

至网络
105
网络接口
310
互连件（总线） 312
系统盘 306
文件
318
I/O设备
316
输入/输出设备接口
308
控制服务器 120
处理器
304
存储器 314
服务器应用程序
317

图3

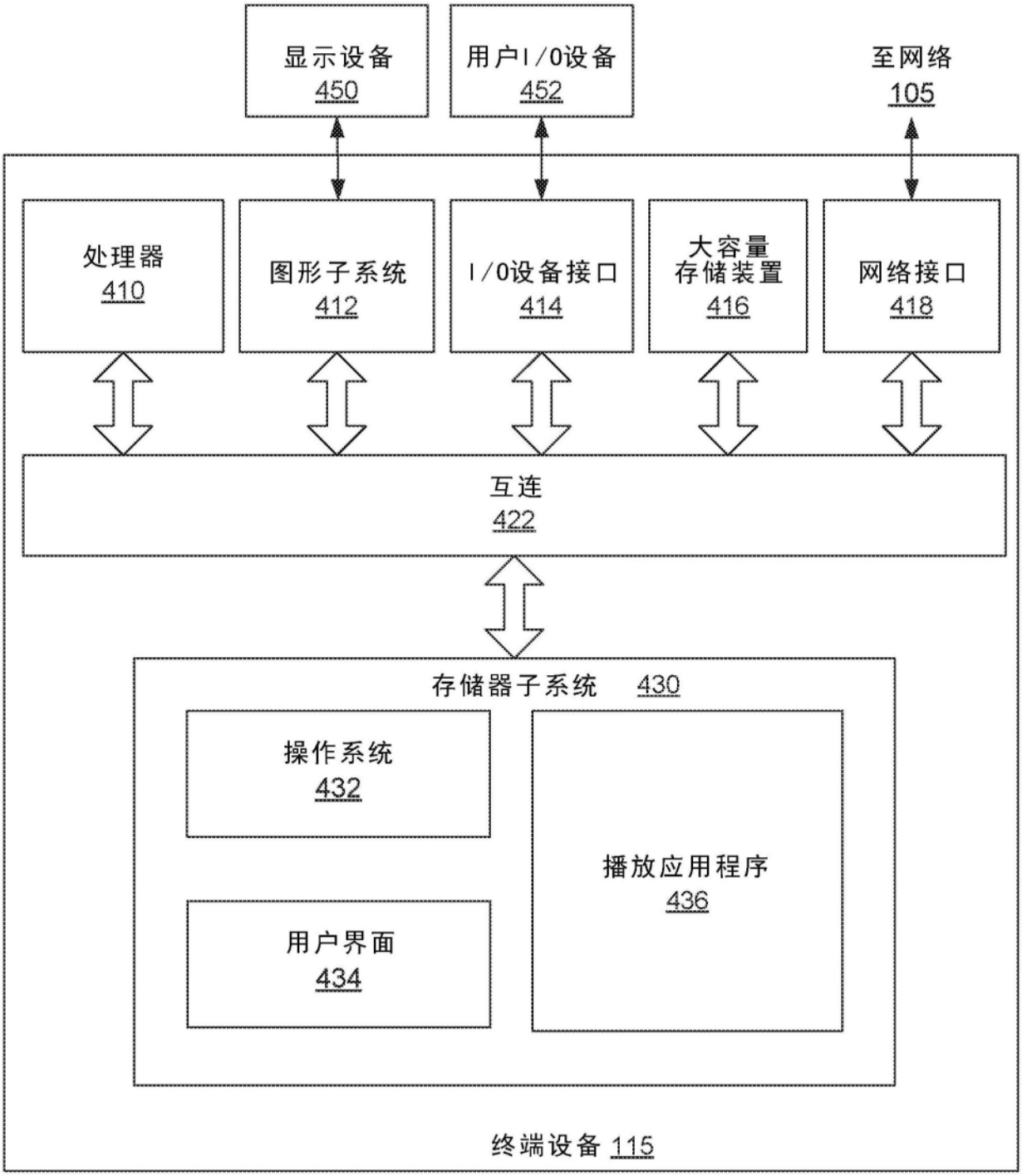

显示设备
450
用户I/O设备
452
至网络
105
处理器
410
图形子系统
412
I/O设备接口
414
大容量
存储装置
416
网络接口
418
互连
422
存储器子系统 430
操作系统
432
播放应用程序
436
用户界面
434
终端设备 115

图4

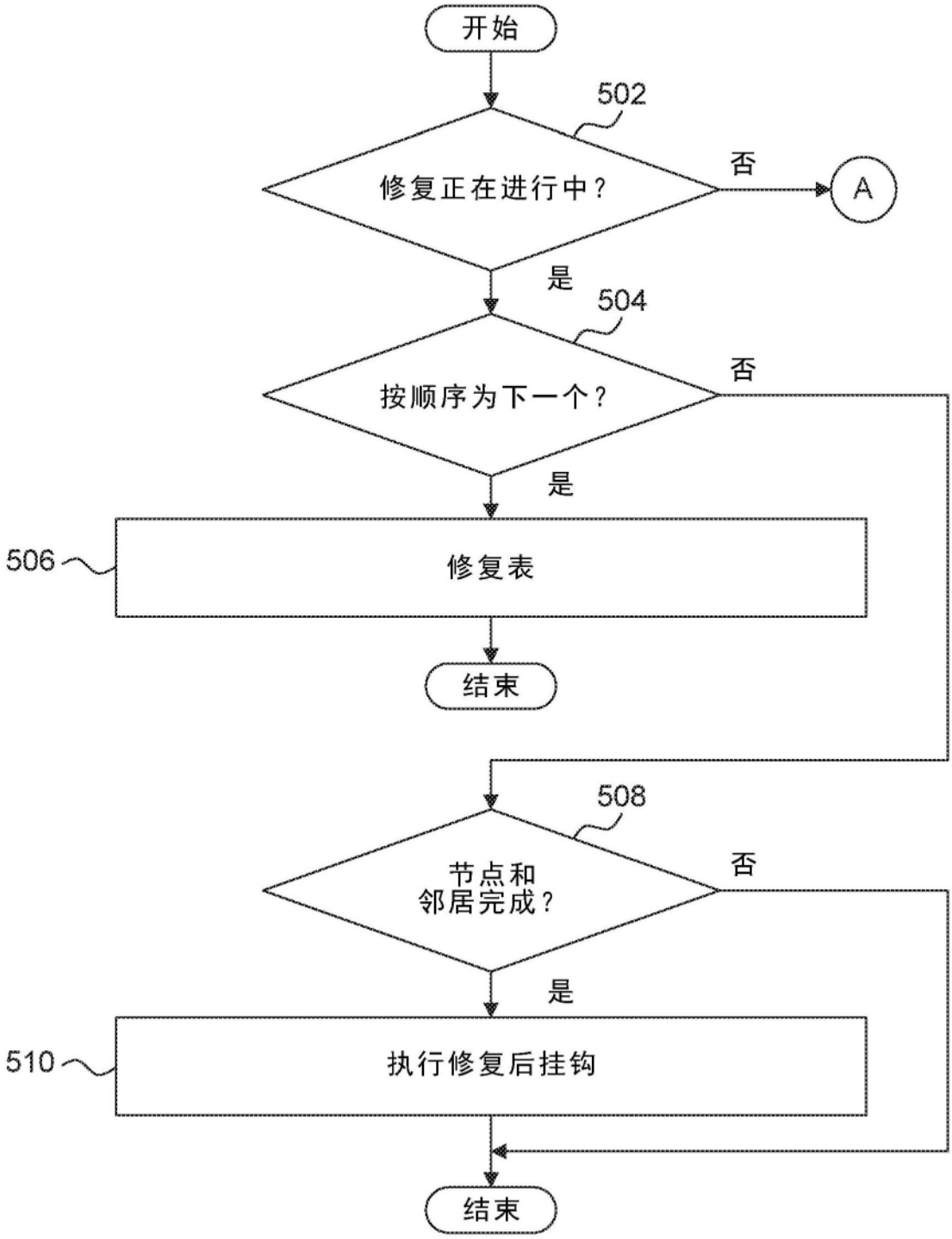

开始
502
修复正在进行中？
否
A
是
504
按顺序为下一个？
否
是
506
修复表
结束
508
节点和
邻居完成？
否
是
510
执行修复后挂钩
结束

图5A

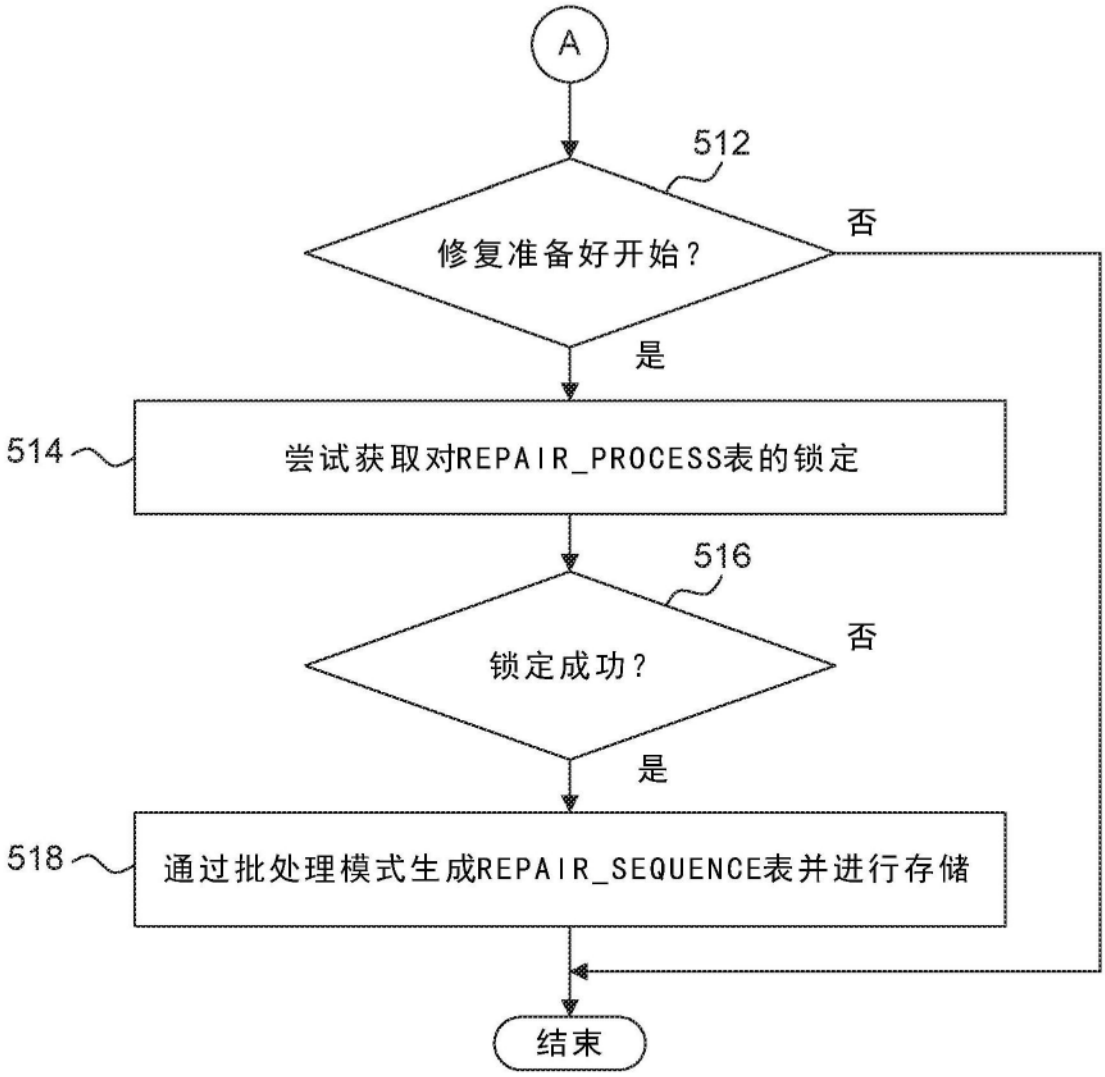

A
512
修复准备好开始？
否
是
514
尝试获取对REPAIR_PROCESS表的锁定
516
锁定成功？
否
是
518
通过批处理模式生成REPAIR_SEQUENCE表并进行存储
结束

图5B