(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 107408128 B
(45)授权公告日 2020.12.08

(21)申请号 201680014244.2
(22)申请日 2016.04.20
(65)同一申请的已公布的文献号
申请公布号 CN 107408128 A
(43)申请公布日 2017.11.28
(30)优先权数据
62/150,188 2015.04.20 US
62/150,191 2015.04.20 US
62/198,958 2015.07.30 US
(85)PCT国际申请进入国家阶段日
2017.09.07
(86)PCT国际申请的申请数据
PCT/US2016/028420 2016.04.20
(87)PCT国际申请的公布数据
WO2016/172195 EN 2016.10.27
(73)专利权人 甲骨文国际公司
地址 美国加利福尼亚
(72)发明人 V·赫格德 J·德拉瓦瑞恩 D·萨博 M·迪尔曼 L·诺瓦克 胡维 A·施瓦鲁德拉艾哈 周彤 I·塔拉努维
(74)专利代理机构 中国贸促会专利商标事务所有限公司 11038
代理人 边海梅
(51)Int.Cl.
G06F 16/22(2019.01)
G06F 16/2455(2019.01)
(56)对比文件
US 2014324910 A1,2014.10.30
CN 103365929 A,2013.10.23
CN 101777047 A,2010.07.14
CN 103605727 A,2014.02.26
CN 1567301 A,2005.01.19
US 2015026213 A1,2015.01.22
审查员 谢晶

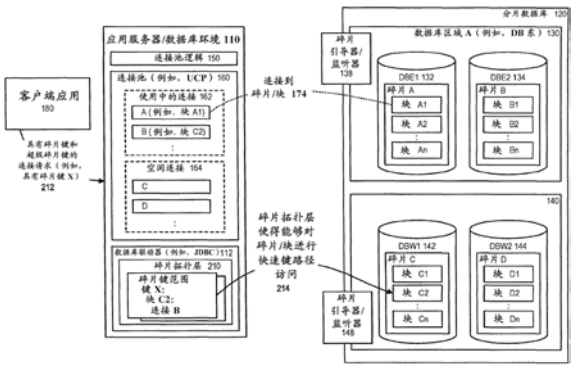
权利要求书5页 说明书23页 附图12页

(54)发明名称

用于使用高速缓存和碎片拓扑提供对分片数据库的访问的系统和方法

(57)摘要

根据实施例，系统使得能够访问分片数据库。连接到分片数据库的碎片感知的客户端应用可以使用连接池(例如，通用连接池(UCP))来存储或访问共享池内到分片数据库的不同碎片或块的连接。当创建新连接时，可以在数据库驱动器层处构建碎片拓扑层，该碎片拓扑层获知碎片的位置的碎片键范围并将其高速缓存，并使得来自客户端应用的后续连接请求能够使用对适当碎片或块的快速键路径访问。连接池和数据库驱动器可以被配置为允许客户端应用在连接检出期间或者在稍后的时间处提供碎片键；辨识由客户端应用指定的碎片键；以及启用由客户端应用到特定碎片或块的连接。



CN 107408128 B

1.一种用于提供对分片数据库的访问的系统,包括:

计算机,所述计算机包括处理器;

应用服务器或数据库环境,所述应用服务器或数据库环境提供对具有存储和呈现数据的多个碎片的数据库的访问,并且其中所述数据库与以下各项相关联:

数据库驱动器,以及

连接池,所述连接池创建和维护用于与所述数据库一起使用的连接的池,

其中所述连接池和所述数据库驱动器一起操作,以使用所述连接来提供由客户端应用对存储在所述数据库处的数据的访问;其中所述数据库驱动器将所述数据库内碎片的位置的碎片键范围高速缓存作为碎片拓扑,用于在处理连接请求中使用;以及

其中所述数据库驱动器被配置为使得客户端应用能够作为连接请求的一部分来访问所述数据库,包括使用所述碎片拓扑来确定所述数据库内适当碎片的位置以及提供由所述客户端应用对所述数据库的所述适当碎片的访问,包括:

从客户端应用接收碎片键信息;

使用所述碎片键信息以通过其碎片键识别连接,并且提供对所述数据库的适当碎片的访问,以供所述客户端应用使用;以及

当接收到针对相同碎片键的后续请求时,允许对所述连接的重用。

2.如权利要求1所述的系统,其中碎片拓扑层被配置为使得后续连接请求能够绕过碎片引导器或监听器部件,并且代替地使用对适当碎片或块的快速键路径访问,其中所述碎片引导器或监听器部件操作以提供由软件客户端应用对数据库碎片的访问。

3.如权利要求1或2所述的系统,其中所述连接池和所述数据库驱动器被配置为使得客户端应用能够在以下各项中的至少一者期间提供碎片键信息:到所述数据库的连接的检出,或在稍后的时间点处;以及

其中所述系统被配置为使用所述碎片键信息来提供由所述客户端应用对所述数据库的适当碎片的直接访问,以用于由所述客户端应用使用。

4.如权利要求3所述的系统,其中所述数据库驱动器和所述连接池被配置为辨识由所述客户端应用指定的碎片键,并且使得所述客户端应用能够连接到与该客户端应用相关联的特定碎片和块。

5.如权利要求1-2或4中任何一项所述的系统,其中所述连接池被配置为通过连接的碎片键来识别该连接,并且允许当从客户端应用接收到对相同碎片键的请求时重用连接。

6.如权利要求1-2或4中任何一项所述的系统,其中,如果所述连接池中没有到特定碎片或块的连接,那么尝试将到另一个块的现有可用连接重新目的化并且重用该连接。

7.一种用于提供对分片数据库的访问的系统,包括:

计算机,所述计算机包括处理器;

应用服务器或数据库环境,所述应用服务器或数据库环境提供对具有存储和呈现数据的多个碎片的数据库的访问,并且其中所述数据库与以下各项相关联:

数据库驱动器,以及

连接池,所述连接池创建和维护用于与所述数据库一起使用的连接的池,

其中所述连接池和所述数据库驱动器一起操作,以使用所述连接来提供由客户端应用对存储在所述数据库处的数据的访问;

其中所述连接池和所述数据库驱动器被配置为使得客户端应用能够在以下各项中的至少一者期间提供碎片键信息:到所述数据库的连接的检出,或在稍后的时间点处;以及

其中所述碎片键信息然后被所述系统使用以提供由所述客户端应用对所述数据库的适当碎片的直接访问,以用于由所述客户端应用使用,包括:

从客户端应用接收碎片键信息;

使用所述碎片键信息以通过其碎片键识别连接,并且提供对所述数据库的适当碎片的访问,以供所述客户端应用使用;以及

当接收到针对相同碎片键的后续请求时,允许对所述连接的重用。

8.如权利要求7所述的系统,其中所述数据库驱动器和所述连接池被配置为辨识由所述客户端应用指定的碎片键,并且使得所述客户端应用能够连接到与该客户端应用相关联的特定碎片和块。

9.如权利要求7-8中任何一项所述的系统,其中所述连接池被配置为通过连接的碎片键来识别该连接,并且允许当从客户端应用接收到对相同碎片键的请求时重用连接。

10.如权利要求7-8中任何一项所述的系统,其中,如果所述连接池中没有到特定碎片或块的连接,那么尝试将到另一个块的现有可用连接重新目的化并且重用该连接。

11.一种提供对分片数据库的访问的方法,包括:

由计算机提供对具有存储和呈现数据的多个碎片的数据库的访问,并且其中所述数据库与以下各项相关联

数据库驱动器,以及

连接池,所述连接池创建和维护用于与所述数据库一起使用的连接的池,

其中所述连接池和所述数据库驱动器一起操作,以使用所述连接来提供由客户端应用对存储在所述数据库处的数据的访问;其中所述数据库驱动器将所述数据库内的碎片的位置的碎片键范围高速缓存作为碎片拓扑,用于在处理连接请求中使用;以及

其中所述数据库驱动器被配置为使得客户端应用能够作为连接请求的一部分来访问所述数据库,包括使用所述碎片拓扑来确定所述数据库内适当碎片的位置以及提供由所述客户端应用对所述数据库的所述适当碎片的访问,包括:

从客户端应用接收碎片键信息;

使用所述碎片键信息以通过其碎片键识别连接,并且提供对所述数据库的适当碎片的访问,以供所述客户端应用使用;以及

当接收到针对相同碎片键的后续请求时,允许对所述连接的重用。

12.如权利要求11所述的方法,其中碎片拓扑层使得后续连接请求能够绕过碎片引导器或监听器部件,并且代替地使用对适当碎片或块的快速键路径访问,其中所述碎片引导器或监听器部件操作以提供由软件客户端应用对数据库碎片的访问。

13.如权利要求11或12所述的方法,其中所述连接池和所述数据库驱动器被配置为使得客户端应用能够在以下各项中的至少一者期间提供碎片键信息:到所述数据库的连接的检出,或在稍后的时间点处;以及

其中所述碎片键信息然后用于提供由所述客户端应用对所述数据库的适当碎片的直接访问,以用于由所述客户端应用使用。

14.如权利要求13所述的方法,其中所述数据库驱动器和所述连接池被配置为辨识由

所述客户端应用指定的碎片键，并且使得所述客户端应用能够连接到与该客户端应用相关联的特定碎片和块。

15.如权利要求11-12或14中任何一项所述的方法，其中所述连接池被配置为通过连接的碎片键来识别该连接，并且允许当从客户端应用接收到对相同碎片键的请求时重用连接。

16.如权利要求11-12或14中任何一项所述的方法，其中，如果所述连接池中没有到特定碎片或块的连接，那么尝试将到另一个块的现有可用连接重新目的化并且重用该连接。

17.一种提供对分片数据库的访问的方法，包括：

由计算机提供对具有存储和呈现数据的多个碎片的数据库的访问，并且其中所述数据库与以下各项相关联

数据库驱动器，以及

连接池，所述连接池创建和维护用于与所述数据库一起使用的连接的池，

其中所述连接池和所述数据库驱动器一起操作，以使用所述连接来提供由客户端应用对存储在所述数据库处的数据的访问；

其中连接池和数据库驱动器被配置为使得客户端应用能够在以下各项中的至少一者期间提供碎片键信息：到所述数据库的连接的检出，或在稍后的时间点处；以及

其中所述碎片键信息然后用于提供由所述客户端应用对所述数据库的适当碎片的直接访问，以用于由所述客户端应用使用，包括：

从客户端应用接收碎片键信息；

使用所述碎片键信息以通过其碎片键识别连接，并且提供对所述数据库的适当碎片的访问，以供所述客户端应用使用；以及

当接收到针对相同碎片键的后续请求时，允许对所述连接的重用。

18.如权利要求17所述的方法，其中所述数据库驱动器和所述连接池被配置为辨识由所述客户端应用指定的碎片键，并且使得所述客户端应用能够连接到与该客户端应用相关联的特定碎片和块。

19.如权利要求17-18中任何一项所述的方法，其中所述连接池被配置为通过连接的碎片键来识别该连接，并且允许当从客户端应用接收到对相同碎片键的请求时重用连接。

20.如权利要求17-18中任何一项所述的方法，其中，如果所述连接池中没有到特定碎片或块的连接，那么尝试将到另一个块的现有可用连接重新目的化并且重用该连接。

21.一种非暂态计算机可读存储介质，包括存储在其上的指令，所述指令在由一个或多个计算机读取和执行时，使所述一个或多个计算机执行如权利要求11-20中任何一项所述的方法。

22.一种非暂态计算机可读存储介质，包括存储在其上的指令，所述指令在由一个或多个计算机读取和执行时，使所述一个或多个计算机执行步骤，所述步骤包括：

提供具有存储和呈现数据的多个碎片的数据库，并且其中所述数据库与以下各项相关联：

数据库驱动器，以及

连接池，所述连接池创建和维护用于与所述数据库一起使用的连接的池，

其中所述连接池和所述数据库驱动器一起操作，以使用所述连接来提供由客户端应用

对存储在所述数据库处的数据的访问;其中所述数据库驱动器将所述数据库内的碎片的位置的碎片键范围高速缓存作为碎片拓扑,用于在处理连接请求中使用;以及

其中所述数据库驱动器被配置为使得客户端应用能够作为连接请求的一部分来访问所述数据库,包括使用所述碎片拓扑来确定所述数据库内适当碎片的位置以及提供由所述客户端应用对所述数据库的所述适当碎片的访问,包括:

从客户端应用接收碎片键信息;

使用所述碎片键信息以通过其碎片键识别连接,并且提供对所述数据库的适当碎片的访问,以供所述客户端应用使用;以及

当接收到针对相同碎片键的后续请求时,允许对所述连接的重用。

23.如权利要求22所述的非暂态计算机可读存储介质,其中碎片拓扑层使得后续连接请求能够绕过碎片引导器或监听器部件,并且代替地使用对适当碎片或块的快速键路径访问,其中所述碎片引导器或监听器部件操作以提供由软件客户端应用对数据库碎片的访问。

24.如权利要求22或23所述的非暂态计算机可读存储介质,其中所述连接池和所述数据库驱动器被配置为使得客户端应用能够在以下各项中的至少一者期间提供碎片键信息:到所述数据库的连接的检出,或在稍后的时间点处;以及

其中所述碎片键信息然后用于提供由所述客户端应用对所述数据库的适当碎片的直接访问,以用于由所述客户端应用使用。

25.如权利要求24所述的非暂态计算机可读存储介质,其中所述数据库驱动器和所述连接池被配置为辨识由所述客户端应用指定的碎片键,并且使得所述客户端应用能够连接到与该客户端应用相关联的特定碎片和块。

26.如权利要求22-23或25中任何一项所述的非暂态计算机可读存储介质,其中所述连接池被配置为通过连接的碎片键来识别该连接,并且允许当从客户端应用接收到对相同碎片键的请求时重用连接。

27.如权利要求22-23或25中任何一项所述的非暂态计算机可读存储介质,其中,如果所述连接池中没有到特定碎片或块的连接,那么尝试将到另一个块的现有可用连接重新目的化并且重用该连接。

28.一种非暂态计算机可读存储介质,包括存储在其上的指令,所述指令在由一个或多个计算机读取和执行时,使所述一个或多个计算机执行步骤,所述步骤包括:

提供具有存储和呈现数据的多个碎片的数据库,并且其中所述数据库与以下各项相关联:

数据库驱动器,以及

连接池,所述连接池创建和维护用于与所述数据库一起使用的连接的池,

其中所述连接池和所述数据库驱动器一起操作,以使用所述连接来提供由客户端应用对存储在所述数据库处的数据的访问;

其中所述连接池和所述数据库驱动器被配置为使得客户端应用能够在以下各项中的至少一者期间提供碎片键信息:到所述数据库的连接的检出,或在稍后的时间点处;以及

其中所述碎片键信息然后用于提供由所述客户端应用对所述数据库的适当碎片的直接访问,以用于由所述客户端应用使用,包括:

从客户端应用接收碎片键信息；

使用所述碎片键信息以通过其碎片键识别连接，并且提供对所述数据库的适当碎片的访问，以供所述客户端应用使用；以及

当接收到针对相同碎片键的后续请求时，允许对所述连接的重用。

29.如权利要求28所述的非暂态计算机可读存储介质，其中所述数据库驱动器和所述连接池被配置为辨识由所述客户端应用指定的碎片键，并且使得所述客户端应用能够连接到与该客户端应用相关联的特定碎片和块。

30.如权利要求28-29中任何一项所述的非暂态计算机可读存储介质，其中所述连接池被配置为通过连接的碎片键来识别该连接，并且允许当从客户端应用接收到对相同碎片键的请求时重用连接。

31.如权利要求28-29中任何一项所述的非暂态计算机可读存储介质，其中，如果所述连接池中没有到特定碎片或块的连接，那么尝试将到另一个块的现有可用连接重新目的化并且重用该连接。

32.一种包括用于执行如权利要求11-20中任一项所述的方法的单元的装置。

用于使用高速缓存和碎片拓扑提供对分片数据库的访问的系统和方法

[0001] 版权声明

[0002] 本专利文档的公开内容的一部分包含受版权保护的素材。版权拥有者不反对任何人对专利文档或专利公开内容按照在专利商标局的专利文件或记录中出现的那样进行传真复制，但是除此之外在任何情况下都保留所有版权。

[0003] 优先权要求：

[0004] 本申请要求于2015年4月20日提交的、申请号为62/150,191、标题为“SYSTEM AND METHOD FOR PROVIDING DIRECT ACCESS TO A SHARDED DATABASE”；于2015年7月30日提交的、申请号为62/198,958、标题为“SYSTEM AND METHOD FOR PROVIDING DIRECT ACCESS TO A SHARDED DATABASE”；以及于2015年4月20日提交的、申请号为62/150,188、标题为“SYSTEM AND METHOD FOR PROVIDING ACCESS TO A SHARDED DATABASE USING A CACHE AND A SHARD TOPOLOGY”的美国临时专利申请的优先权，以上申请中的每一个通过引用被结合于此。

技术领域

[0005] 本发明的实施例一般而言涉及应用服务器和数据库，并且具体而言涉及用于提供对分片数据库的访问的系统和方法。

背景技术

[0006] 现代面向web的软件应用在可扩展性方面面临越来越大的挑战，包括需要处置极大量的数据。例如，在移动聊天系统内，处理消息所需的数据库表在尺寸上已经显著增加，使得单个表的容量会成为特定应用的可扩展性的限制因素。解决这种类型问题的常见方法是使用分片，其中数据被呈现为多个较小的数据库或碎片(shard)。存在一些在其中可以使用本发明的实施例的环境类型的示例。

发明内容

[0007] 根据实施例，本文所描述的是用于提供对分片数据库的直接访问的系统和方法。碎片引导器或监听器操作以提供由软件客户端应用对数据库碎片的访问。连接池(例如通用连接池，UCP)和数据库驱动器(例如，Java数据库连接(JDBC)部件)可以被配置为允许客户端应用在连接检出(checkout)期间或在稍后的时间提供碎片键(shard key)；辨识客户端应用指定的碎片键；以及启用客户端应用到特定碎片或块(chunk)的连接。该方法使得能够高效地重用连接资源，并且能够更快地访问适当的碎片。

[0008] 根据实施例，系统使得能够使用高速缓存和碎片拓扑来访问分片数据库。连接到分片数据库的碎片感知(shard-aware)的客户端应用可以使用连接池(例如，UCP)来存储或访问共享池内的到分片数据库的不同碎片或块的连接。当创建新的连接时，可以在数据库驱动器层处构建碎片拓扑层，该碎片拓扑层获知碎片位置的碎片键范围并将其高速缓存。

碎片拓扑层使得来自客户端应用的后续连接请求能够使用对适当碎片或块的快速键路径访问。

[0009] 根据实施例，如果连接池中没有到特定碎片或块的可用连接，那么可以尝试将到另一个块的现有可用连接重新目的化(repurpose)，并重用该连接。

[0010] 下面进一步详细描述以上实施例以及附加实施例。

附图说明

[0011] 图1示出根据实施例的用于使得能够直接访问分片数据库的系统。

[0012] 图2进一步示出根据实施例的用于使得能够直接访问分片数据库的系统。

[0013] 图3进一步示出根据实施例的用于使得能够直接访问分片数据库的系统。

[0014] 图4示出根据实施例的用于使得能够直接访问分片数据库的过程。

[0015] 图5示出根据实施例的用于使得能够使用高速缓存和碎片拓扑来访问分片数据库的系统。

[0016] 图6进一步示出根据实施例的用于使得能够使用高速缓存和碎片拓扑来访问分片数据库的系统。

[0017] 图7进一步示出根据实施例的用于使得能够使用高速缓存和碎片拓扑来访问分片数据库的系统。

[0018] 图8示出根据实施例的、描述当连接池用于创建和维护到分片数据库的连接的池时的流程的顺序图。

[0019] 图9示出根据实施例的、描述当应用在没有连接池的情况下使用数据库驱动器来获取到分片数据库的连接时的流程的顺序图。

[0020] 图10示出根据实施例的用于使得能够使用高速缓存和碎片拓扑来访问分片数据库的过程。

[0021] 图11示出根据实施例的数据库拓扑类设计。

[0022] 图12示出根据实施例的服务拓扑类设计。

具体实施方式

[0023] 如上所述，现代面向web的软件应用在可扩展性方面面临越来越大的挑战，包括需要处置极大量的数据，使得单个表的容量可能成为特定应用的可扩展性的限制因素。解决这种类型的问题的常见方法是使用分片，其中数据被呈现为多个较小的数据库或碎片。为了提供对这种环境的支持，根据各种实施例，本文描述的是用于提供对分片数据库的访问的系统和方法。

[0024] 分片数据库

[0025] 根据实施例，分片是数据库缩放技术，其使用跨多个独立物理数据库的数据的横向划分。存储在每个物理数据库中的数据部分被称为碎片。从软件客户端应用的角度来看，所有物理数据库的集合表现为单个逻辑数据库。

[0026] 根据实施例，可以使用碎片键(SHARD_KEY)将数据库表划分为例如一个或多个列，这一个或多个列确定在特定碎片内每一行被存储在哪里。碎片键可以在连接串或描述中被提供为连接数据(CONNECT_DATA)的属性。

[0027] 作为使用碎片组键(SHARDGROUP_KEY)提供某种形式的用户控制的数据划分的附加共享级别,碎片分组可以可选地用于跨数据库组(DBGROUPS)分布数据,例如:

[0028] (DESCRIPTION＝(...)(CONNECT_DATA＝(SERVICE_NAME＝ORCL(SHARD_KEY＝...)(SHARDGROUP_KEY＝...)))

[0029] 在数据库中,碎片键的示例可以包括VARCHAR2、CHAR、DATE、NUMBER或TIMESTAMP。用户负责给出符合数据库中指定的国家语言支持格式的碎片键。根据实施例,分片数据库还可以接受不具有碎片键或碎片组键的连接。

[0030] 根据实施例,为了减少重新分片对系统性能和数据可用性的影响,每个碎片可以被细分成更小的片或块。每个块充当可以从一个碎片移动到另一个碎片的重新分片单元。通过向碎片键映射添加间接级别(level of indirection),块也简化了路由。

[0031] 例如,每个块可以自动地与碎片键值的范围相关联。用户提供的碎片键可以映射到特定的块,并且该块可以映射到特定的碎片。如果数据库操作尝试对特定碎片上不存在的块进行操作,那么将会引发错误。当使用碎片组时,每个碎片组是具有特定的碎片组标识符值的那些块的集合。

[0032] 根据实施例,碎片感知的客户端应用可以与分片数据库配置一起工作,包括连接到在其中基于一个或多个分片方法划分数据的一个或多个数据库碎片的能力。每次需要数据库操作时,客户端应用可以确定它需要连接到的碎片。

[0033] 根据实施例,分片方法可以用于将碎片键值映射到各个碎片。可以支持不同的分片方法,例如:基于散列的分片,其中,给每个块分配散列值范围,使得在建立数据库连接时,系统将散列函数应用到碎片键的给定值并计算对应的散列值,该散列值然后基于该值所属的范围被映射到块;基于范围的分片,其中,给各个碎片直接分配碎片键值范围;以及基于列表的分片,其中,每个碎片与碎片键值列表相关联。

[0034] 根据实施例,数据库也可以与一个或多个超级碎片(supershard)相关联。超级碎片与数据库的关联允许将附加的约束放入存储在数据库中的记录中。例如,在特定数据库表内,如果位置被识别为碎片组标识符,那么可以使用分片方法来确保存储特定客户的记录的数据中心是最靠近由该客户指定的位置的数据中心。

[0035] 根据实施例,重新分片是跨分片数据库的碎片重新分布数据的过程。在一些情况下需要重新分片(例如,当碎片被添加到分片数据库或从分片数据库中移除时),以消除跨碎片的数据或工作负载分布的偏差(skew);或者满足应用要求(例如,某些数据必须被存储在一起)。

[0036] 根据实施例,可以使用全局数据服务(GDS)部件来提供与多数据库环境一起使用的可扩展性、可用性和可管理性框架。GDS可以与一个或多个全局服务管理器(GSM)监听器一起操作,以将多数据库配置作为单个逻辑数据库呈现给客户端,包括故障转移支持、负载平衡和数据库服务的集中式管理。例如,可以基于可用性、负载、网络延迟、复制滞后(lag)或其它参数将客户端请求路由到适当的数据库。GDS池提供了一组提供全局服务的复制数据库,使得例如GDS池中的数据库可以位于跨不同区域的多个数据中心。分片GDS池包含分片数据库的碎片连同其复制品(replica)。从数据库客户端的角度来看,分片GDS池表现为单个分片数据库。

[0037] 1.直接访问分片数据库

[0038] 根据实施例，本文所描述的是用于提供对分片数据库的直接访问的系统和方法。碎片引导器或监听器部件操作以提供由软件客户端应用对数据库碎片的访问。连接池(例如通用连接池，UCP)和数据库驱动器(例如，Java数据库连接(JDBC)部件)可以被配置为允许客户端应用在连接检出期间或稍后的时间提供碎片键；辨识客户端应用指定的碎片键；以及启用客户端应用到特定碎片或块的连接。该方法使得能够高效地重用连接资源，并且能够更快地访问适当的碎片。

[0039] 图1示出根据实施例的用于使得能够直接访问分片数据库的系统。

[0040] 如图1所示，根据应用服务器或数据库环境110的实施例(应用服务器或数据库环境110包括物理计算机资源(例如，处理器/CPU、存储器、网络)111和数据库驱动器(例如，JDBC部件)112)，连接到分片数据库120的碎片感知的客户端应用可以使用具有相关联的连接池逻辑150的连接池部件160(例如，UCP)以存储或访问共享池内的到分片数据库的不同碎片或块的连接。

[0041] 在图1所示的示例性环境中，分片数据库可以包括第一数据库区域A(这里指示为DB东(DBE))130，第一数据库区域A包括具有存储为块A1，A2，...An的碎片A的分片数据库实例“DBE 1”132；以及具有存储为块B1，B2，...Bn的碎片B的“DBE 2”134。

[0042] 如图1还示出的，第二数据库区域B(这里指示为DB西(DBW))140包括具有存储为块C1，C2，...Cn的碎片C的分片数据库实例“DBW 1”142；以及具有存储为块D1，D2，...Dn的碎片D的“DBW 2”144。

[0043] 根据实施例，每个数据库区域或分片数据库实例组可以与碎片引导器或监听器部件(例如，GSM监听器或另一种类型的监听器)相关联，该引导器或监听器部件操作以提供由软件客户端应用对数据库碎片的访问。例如，碎片引导器或监听器138可以与第一数据库区域A相关联，并且另一个碎片引导器或监听器148可以与第二数据库区域B相关联。

[0044] 根据实施例，客户端应用可以在连接请求期间向连接池(例如，UCP)提供一个或多个碎片键；并且基于这一个或多个碎片键，连接池可以将连接请求路由到正确或适当的碎片。

[0045] 根据实施例，连接池维护多个使用中的连接162和空闲连接164。连接池可以通过其碎片键来识别到特定碎片或块的连接，并且允许当从客户端接收到对相同碎片键的请求时重用连接。

[0046] 例如，如图1所示，可以使用到块A1的连接来连接174到该块。如果池中没有可用的到特定碎片或块的连接，那么系统可以尝试重新目的化到另一个碎片或块的现有可用连接，并重用该连接。可以使得跨数据库中的碎片和块的数据分布对用户是透明的，这也使块的重新分片对用户的影响最小化。

[0047] 图2进一步示出根据实施例的用于使得能够直接访问分片数据库的系统。

[0048] 如图2所示，当碎片感知的客户端应用180向连接池(例如，UCP)提供与连接请求相关联的一个或多个碎片键182时；然后如果连接池或数据库驱动器已经具有该碎片键的映射，那么连接请求可以被直接转发到适当的碎片和块184，在本示例中为块C2。

[0049] 图3进一步示出根据实施例的用于使得能够直接访问分片数据库的系统。

[0050] 如图3所示，当碎片感知的客户端应用不提供与连接请求相关联的碎片键时，或者如果连接池(例如，UCP)或数据库驱动器(例如，JDBC)不具有所提供的碎片键的映射；那么

连接请求可以被转发到适当的碎片引导器或监听器(例如,GDS/GSM监听器)186,在本示例中包括与第二数据库区域B相关联的碎片引导器或监听器。

[0051] 图4示出根据实施例的用于使得能够直接访问分片数据库的过程。

[0052] 如图4所示,在步骤192处,提供了具有多个碎片并且与一个或多个数据库驱动器以及一个或多个连接池相关联的数据库,它们一起提供由客户端应用对存储在数据库中的数据的访问。

[0053] 在步骤194处,数据库驱动器或连接池中的一个或多个被配置为使得客户端应用能够在到数据库的连接的检出期间或者在稍后的时间点提供碎片键信息,该信息然后用于提供由客户端应用对数据库的适当碎片的访问。

[0054] 在步骤196处,数据库驱动器和/或连接池辨识由客户端应用指定的碎片键,并使得客户端应用能够连接到与该客户端应用相关联的特定碎片和块。

[0055] 在步骤198处,连接池可以通过连接的碎片键来识别连接,并且允许当从客户端应用接收到对相同碎片键的请求时重用连接。

[0056] 构建用于连接请求的碎片键

[0057] 根据实施例,碎片感知的客户端应用可以例如使用ShardKey或使得能够利用不同数据类型准备复合碎片键的类似接口和构建器来识别和构建碎片键以及可选地识别和构建碎片组(该碎片键以及碎片组是获取到分片数据库的连接所需的):

[0058] subkey(object subkey,java.sql.SQLTYPE subkeyDataType)

[0059] 根据实施例,可以在ShardKey构建器上对subkey(...)方法进行多次调用以构建复合碎片键,其中每个subkey可以是不同的数据类型。数据类型可以使用枚举器oracle.jdbc.OracleType定义,例如是串(string)和日期(date)复合碎片键:

[0060] ShardKey shardKey＝datasource

[0061] .createShardKeyBuilder()

[0062] .subkey(<string>,oracle.jdbc.OracleType.VARcHAR2)

[0063] .subkey(<date>,oracle.jdbc.OracleType.DATE)

[0064] .build();

[0065] 根据实施例,可以支持一组选择的数据类型作为键,并在构建器中提供对应的验证以阻止未支持的数据类型。示例性数据类型可以包括OracleType.VARCHAR2;OracleType.CHAR;OracleType.NVARCHAR;OracleType.NCHAR;OracleType.NUMBER;OracleType.FLOAT;OracleType.DATE;OracleType.TIMESTAMP;OracleType.TIMESTAMP WITH LOCAL TIME ZONE;以及OracleType.RAW。

[0066] 利用碎片键值更新连接串

[0067] 根据实施例,当使用监听器为包括一个或多个碎片键的连接请求创建连接时,ShardKey接口将碎片键转换为对应的BASE64编码串,使得连接数据将具有与数据库驱动器相关的以下两个字段:

[0068] SHARD_KEY_B64用于碎片键的base64编码二进制表示

[0069] GROUP_KEY_B64用于组键的base64编码二进制表示

[0070] 用于base64编码值的字段(*_B64)可以具有以下格式:

[0071] ...(CONNECT_DATA＝(SHARD_KEY_B64＝[version][type][int literal][int

literal]...,[base64 binary],[base64 binary],[base64 binary],...))...

[0072] 根据实施例,连接串以“版本(version)”号=1的头部(header)开始;后面是“类型(type)”,“类型”的值可以如表1所示的那样定义。

[0073]

0	该串不包含散列值。	字符值以AL32UTF8(对于varchar)编码和AL16UTF16(对于nvarchar)编码进行编码。
1	该串包含散列值。	

[0074]

2	该串不包含散列值。	字符值以(可以特定于每一列进行编码的)数据库编码进行编码。
3	该串包含散列值。	
4	该串只包含散列值。	

[0075] 表1

[0076] 根据实施例,在串类型之后,提供了(作为十进制整数文字的)空格分隔的值“类型标识符”,如表2所示。

[0077]

1	VARCHAR,NVARCHAR,[CHAR,NCHAR=96]
2	NUMBER
12	DATE
23	RAW
180	TIMESTAMP
231	TIMESTAMP WITH LOCAL TIME ZONE

[0078] 表2

[0079] 根据实施例,头部通过逗号终止,并且后面是用于复合碎片键的每个部分的base64编码值的逗号分隔的列表,其中数据类型可以如下编码:

[0080] NUMBER、FLOAT、DATE、TIMESTAMP、TIMESTAMP WITH LOCAL TIMEZONE:对于这些数据类型,在B64编码之前可以使用对应的Oracle表示来将它们转换为对应的字节数组。

[0081] RAW,VARCHAR,CHAR:对于这些数据类型的来自用户的输入串,客户端可以在进行B64编码之前使用AL32UTF8字符编码。

[0082] NVARCHAR,NCHAR:对于这些数据类型,推荐AL16UTF16编码。

[0083] 例如,键(“US”,“94002”)可以被编码为:

[0084] ・・・(CONNECT_DATA=(SHARD_KEY_B64=112,VVM=,OTQwMDI=))・・・

[0085] 利用已知碎片键从连接池进行连接检出

[0086] 根据实施例,当从连接池借用连接时,碎片感知的客户端应用可以使用如上所述的连接构建器来提供碎片键和碎片组键,其可以例如利用OracleDataSource和PoolDataSource来提供:

[0087] Connection conn=dataSource

[0088] .createConnectionBuilder()

[0089] .shardkey(<shard_key>)//of type ShardKey

[0090] .shardGroupKey(<shard_group_key>)

[0091] .build()

[0092] 根据实施例,以上API确保从池中借用或创建的连接被连接到正确的碎片和块,包括:

[0093] (1)该连接上的所有操作将被限制在由连接检出期间供应的键指定的碎片和块;否则将向应用抛回错误或异常。竞争条件也将导致连接使用期间的异常。

[0094] (2)当使用明确指定请求的碎片键的getConnection API时,针对该数据源指定的URL不应该已经包含碎片键;否则将向用户返回异常。

[0095] 在现有连接或已检出的连接上设置碎片键

[0096] 根据实施例,setShardKey(设置碎片键)或类似接口允许在连接上设置碎片键。当应用在连接检出时无法提供一个或多个碎片键时,连接池和数据库驱动器(例如,UCP/JDBC)也可以支持在连接级别处(例如,在OracleConnection类中)使用API:

[0097] connection.setShardKey(<shard_key>,<shard_group_key>);

[0098] 根据实施例,碎片键和碎片组键是ShardKey类型。当以上API在连接上被调用时,以下之一是可能的:

[0099] (1)setShardKey中供应的一个或多个碎片键与在其上原始地创建了连接的碎片和块相匹配。在这种情况下,不需要进一步的动作,并且连接可以原样使用。

[0100] (2)由setShardKey API指定的碎片键映射到不同的“块”但是在在其上原始地创建了连接的同一个碎片上。在这种情况下,需要将连接切换到正确的块。

[0101] (3)由setShardKey指定的碎片键恰好与在其上创建了连接的碎片不同。因此,在连接可以被进一步使用之前,需要将底层物理连接切换到正确的碎片。

[0102] 根据实施例,如果在直接使用数据库驱动器(例如,JDBC)而没有池时遇到后一种情况,那么向应用返回指示不正确的碎片键的错误。连接仍然是可用的,并且可以用于在setShardKey被调用之前该连接所连接到的原始块上执行操作。如果setShardKey成功,那么一旦碎片键被设置,该连接上的所有操作就将被限制到由在setShardKey期间供应的键指定的碎片和块。并且,应用应该确保在调用setShardKey时连接上没有正在进行的事务。如果不是这样,那么向应用抛出异常,指示setShardKey无法完成。

[0103] 根据实施例,如果情况(3)中的连接切换是迫切的,或者在情况(2)中需要进行服务切换,那么连接上所有打开的结果集、语句、大对象、流等必须被关闭。

[0104] 在没有提供碎片键的情况下的连接检出

[0105] 根据实施例,当应用在从连接池或数据库驱动器(例如,UCP/JDBC)检出时没有提供碎片键时,在使用连接执行单碎片查询之前,查询被引导到协调器碎片,该协调器碎片促进这种查询的执行并将结果返回给应用。

[0106] 根据实施例,一旦通过监听器创建了连接,连接池(例如,UCP)就将尝试提取与块对应的实际“块名称”以及在其上创建了该连接的“实例/碎片名称”,并使用该信息来识别池中的连接。

[0107] 在连接创建之后,块名称和实例/碎片名称信息将存在于上下文(SYS_CONTEXT)

中,并且如果在任何点处在该连接上有“切换块”操作,该信息就会被更新。块信息也可以作为池中的连接对象的一部分,并且可以在检出时查找要返回给用户的最佳候选连接期间被使用,以及还用于响应于系统状态或通知事件(例如,来自Oracle快速应用通知(Fast Application Notification,FAN)或运行时连接负载平衡(Runtime Connection Load Balancing,RLB)环境)的连接处理。

[0108] 连接池中的连接借用期间的连接选择

[0109] 根据实施例,对于来自连接池(例如,UCP)的每个连接检出请求,连接池可以使用与池中的每个连接一起存储的块信息来以以下方式之一确定返回给用户的最佳可能的匹配连接:

[0110] (1)给定连接请求中的碎片键,连接池可以尝试在目前为止由池发现的碎片拓扑中查找该键的匹配的块名称。如果未找到匹配的块名称,那么连接请求被转发到监听器以便创建新连接。

[0111] (2)否则,块名称可以被用于使用连接选择算法来获取在其上存在该块的实例的列表,并且连接被选择为到这些实例中的一个。这将确保连接被创建到与用户请求的碎片键对应的块位于其上的实例。

[0112] (3)一旦选择了连接,就在连接上做出块切换捎带(piggyback),从而指定要在当前会话中使用的块名称和碎片键。如果未找到匹配的连接,那么通过将请求转发给监听器来创建新的连接。

[0113] 故障转移/重新分片事件处置

[0114] 根据实施例,连接池(例如,UCP)可以使用诸如例如以下的订阅串来按服务订阅故障切换事件:

[0115] ("eventType=database/event/service/<service_name>")

[0116] 对于分片支持,OracleDBTopology和连接池(例如,UCP)可以订阅接收与块移动或分割对应的事件类型。块级别事件订阅串可以是例如:

[0117] ("eventType=database/event/service/chunk")

[0118] 块名称可以是该事件主体的一部分。当第一连接被置为到特定块时,从sys_context读取对应的块名称,并且块事件订阅可以被置为接收所有与块相关的事件。根据实施例,块订阅不是基于服务的。

[0119] 根据实施例,在服务器侧的任何重新分片努力或块的分割将导致具有事件主体中指定的块名称的对应块关闭通知事件(chunk down notification event),例如:

[0120] VERSION=1.0 event_type=CHUNK chunk=<chunk name>

[0121] instance=<instance name>host=<host>database=<db name>

[0122] db_domain=<db domain name>status=<UP|DOWN>

[0123] timestamp=<timestamp>timezone=<timezone>

[0124] 根据实施例,连接池(例如,UCP)可以更新其碎片拓扑以与最近的块事件同步:

[0125] (1)连接池将从块放置信息(即,块到其上存在该块的实例的列表的映射)中移除对应的“实例”信息。

[0126] (2)块的碎片键信息不从拓扑中移除。

[0127] (3)当块在已知实例上进入工作状态(come up)时,该块的实例信息被添加到块放

置表中,并且如果该块的碎片键信息已经改变,则利用该块的碎片键信息更新碎片键拓扑高速缓存。

[0128] (4)如果块在未知实例上进入工作状态,那么实例信息被添加到UCP拓扑,并且用于该实例的整个碎片键拓扑以及块放置数据被获取并放置在连接池的碎片键拓扑高速缓存中。

[0129] (5)块UP事件不由连接池处理。

[0130] 根据实施例,当需要分割分片数据库中的块时,服务器将发送看起来如下的块Split(分割)事件:

[0131] VERSION=1.0 event_type=CHUNK chunk=<chunk name>

[0132] instance=<instance name>host=<host>database=<db name>

[0133] db_domain=<db domain name>status=SPLIT timestamp=<timestamp>

[0134] timezone=<timezone>newchunk=<new chunkname>

[0135] [hash=<split boundary hash value>]

[0136] 根据实施例,散列仅在基于自动散列的分片的情况下是适用的,其中边界对应于第一分割块的高值和第二分割块的低值)。连接池(例如,UCP)将需要如下地处理块分割事件:

[0137] (1)在其上接收分割的实例名称将从块放置数据中去除。

[0138] (2)现在利用新版本来更新块名称,因为块现在正经历分割,并且分割之后的新块需要与在其中块尚未分割的复制品上的旧块区分开。

[0139] (3)碎片拓扑表和块放置表两者都利用新的块名称进行更新,保持所有其它数据原封不动。这将确保尚未经历分割的复制品将保持不受分割的影响,并将继续像以前一样处置连接请求。

[0140] (4)当没有处置请求的现有连接并且创建新连接时,如果连接被创建到新分割的块,那么新的块名称及其范围被填充到拓扑高速缓存中,并且在此时分割的块/未分割的块都将可用于处置针对键范围的连接请求。

[0141] (5)该过程继续进行,直到所有复制品都经历了分割,在这种情况下,旧的块名称(更新的版本)将不再具有任何实例,这些旧的块名称然后从块放置表和碎片键拓扑表两者中清除。

[0142] (6)对于自动分片,在步骤(1)中,在接收到块分割之后,具有由散列值指定的新范围的新的块信息在碎片键拓扑表中被更新,新的块放置信息也被更新。

[0143] 对共享池的运行时连接负载平衡(RLB)要求

[0144] 根据实施例,对于具有到分片数据库的连接的池,RLB订阅可以包括:连接池(例如,UCP)将订阅来自GDS集群的RLB通知的全局服务。来自GDS数据库的RLB事件将基于全局服务;因此,当接收到全局服务RLB时,为了检查池当前正在服务的每个块的分布模式,系统可以推导出适用于该块的RLB%,并且然后尝试连接分布。

[0145] 用于分片数据库的块放置数据

[0146] 根据实施例,在连接池(例如UCP)中创建并维护分片元数据信息,该分片元数据具有块名称到服务在其上可用的碎片实例的映射,以及块在该实例上的优先级。根据实施例,对于分片数据库,在内部,所创建的数据结构将看起来如表3所示:

[0147]

块名称	碎片实例列表
CHUNK_1_1	[<Instance:dbs1%1,Priority:0>,<Instance:dbs1%2,Priority:0>]
CHUNK_1_10	[<Instance:dbs1%1,Priority:0>,<Instance:dbs1%2,Priority:0>]
CHUNK_1_100	[<Instance:dbs1%1,Priority:0>,<Instance:dbs1%2,Priority:0>]
CHUNK_1_11	[<Instance:dbs1%1,Priority:0>,<Instance:dbs1%2,Priority:0>]

[0148] 表3

[0149] 用于分片数据库的碎片键到块名称的映射

[0150] 根据实施例,除了服务到碎片列表的映射之外,系统还可以包括碎片键范围/列表或范围到与这些碎片键对应的块的映射。根据实施例,当分片方法被设置为基于散列时,碎片分组可以是基于列表或范围的。在所有其它情况下,不支持碎片组。以下情况作为拓扑数据的示例是可能的,其包括对于基于散列的分片和基于列表的超级分片(表4):

[0151]

<组键列表,碎片键散列范围>	块名称
[gold,silver],{Low:0,High:42949672}	CHUNK_2_1
[gold,silver],{Low:42949672,High:85899344}	CHUNK_2_2
[regular,bronze],{Low:0,High:42949672}	CHUNK_1_100

[0152] 表4

[0153] 对于基于散列的分片和基于范围的超级分片(表5):

[0154]

<组键范围,碎片键散列范围>	块名称
{1-10},{Low:0,High:42949672}	CHUNK_2_1
{1-10},{Low:42949672,High:85899344}	CHUNK_2_2
[10-MAX],{Low:0,High:42949672}	CHUNK_1_100

[0155] 表5

[0156] 对于基于范围的分片,没有碎片分组(表6):

[0157]

碎片键散列范围	块名称
{Low:0,High:42949672}	CHUNK_2_1
{Low:42949672,High:85899344}	CHUNK_2_2
{Low:85899344,High:MAX}	CHUNK_1_100

[0158] 表6

[0159] 对于基于列表的分片,没有碎片分组(表7):

[0160]

碎片键列表	块名称
[IN,AU]	CHUNK_2_1
[US,CAN]	CHUNK_2_2
[DE,UK]	CHUNK_1_100

[0161] 表7

[0162] 碎片元数据高速缓存

[0163] 根据实施例,连接池维护碎片元数据高速缓存,碎片元数据高速缓存包括当从池中创建到分片数据库的不同块和碎片的新连接时获知和收集到的信息,例如:

[0164] (1)适用于连接池(例如,UCP)已创建到其的连接的块的碎片键和碎片组键范围;

[0165] (2)碎片键列信息,诸如数据库编码(如果适用的话);

[0166] (3)分片和碎片分组方法(例如,基于散列的方法、基于列表的方法、基于范围的方法)。

[0167] 在创建连接时构建拓扑

[0168] 根据实施例,当使用碎片创建新连接时,从LOCAL_CHUNK_TYPES表中提取来自LOCAL_CHUNK表的数据和分片元数据(诸如分片类型和数据库编码类型)并且将其存储在客户端侧的元数据高速缓存中。实例上每个块的块优先级也被读取并存储在客户端侧。

[0169] 基于服务器侧Oracle通知服务(Oracle Notification Service,ONS)事件更新拓扑

[0170] 根据实施例,通过处理所有以下数据库高可用性(HA)事件并且更新客户端侧的对应数据结构,元数据高速缓存可以与服务器侧的变化保持同步:

[0171] (1)对于全局服务成员关闭事件或实例关闭事件,从受影响的块的实例列表中移除对应的实例。如果块没有在任何其它实例上处于工作状态,那么对应的块数据需要从存储在拓扑中的块元数据中被清除。

[0172] (2)对于全局服务关闭事件,服务和块相关的数据二者都需要被清理。

[0173] (3)对于块关闭事件,存储在拓扑中的块信息中的对应受影响的条目需要被清理。

[0174] 碎片键查找

[0175] 根据实施例,可以在元数据高速缓存上提供chunk(ShardKey shardKey,ShardKey groupKey)格式的内部方法,并且该方法用于获得与被提供作为该方法的参数的一个或多个碎片键对应的块名称。ShardKey对象可以包括允许比较和检查两个碎片键的相等性的方法。

[0176] 安全性

[0177] 根据实施例,为了解决关于暴露分片数据库中的不同块的内容(碎片键和范围)和位置的安全性问题,系统可以支持使用在服务器上定义的特定“受密码保护的角色”,用于使得用户能够访问表中的数据,诸如LOCAL_CHUNK_TYPE、LOCAL_CHUNK、LOCAL_CHUNK_COLUMNS。使用UCP或JDBC或构建自定义连接池的数据库用户以及希望维护到分片数据库的连接(或池)并且也想构建分片数据库的客户端侧的拓扑快照的数据库用户可能不得不具有为他们提供的这个角色。

[0178] 2.使用高速缓存和碎片拓扑访问分片数据库

[0179] 根据实施例,系统使得能够使用高速缓存和碎片拓扑来访问分片数据库。当创建新连接时,可以在数据库驱动器层处构建碎片拓扑层,碎片拓扑层获知并高速缓存碎片位置的碎片键范围。碎片拓扑层使得来自客户端应用的后续连接请求能够绕过碎片引导器或监听器,并且代替地使用对适当碎片或块的快速键路径访问。图5示出根据实施例的用于使得能够使用高速缓存和碎片拓扑来访问分片数据库的系统。

[0180] 如图5所示,根据实施例并且如上所述,连接到分片数据库的碎片感知的客户端应用可以使用连接池(例如,UCP)来存储或访问共享池内到分片数据库的不同碎片或块的连接。

[0181] 如上面还描述的,根据实施例,分片数据库可以包括第一数据库区域A(DBE),第一数据库区域A包括具有存储为块A1,A2,...An的碎片A的分片数据库实例“DBE 1”;和具有存

储为块B1,B2,...Bn的碎片B的“DBE 2”;并且分片数据库包括第二数据库区域B(DBW),第二数据库区域B包括具有存储为块C1,C2,...Cn的碎片C的分片数据库实例“DBW 1”;和具有存储为块D1,D2,...Dn的碎片D的“DBW 2”。

[0182] 如上面还描述的,根据实施例,每个数据库区域或分片数据库实例组可以与碎片引导器或监听器部件(例如,GSM)相关联。应用可以在连接请求期间向UCP提供碎片键,并且基于该碎片键,连接池可以将连接请求路由到正确的碎片。

[0183] 根据实施例,当创建新连接时,可以在数据库驱动器(例如,JDBC)层处构建碎片拓扑层210,碎片拓扑层210获知并高速缓存每个碎片位置的碎片键范围。后续连接请求可以绕过碎片引导器或监听器,并且代替地使用对适当碎片或块的快速键路径访问。

[0184] 图6进一步示出根据实施例的用于使得能够使用高速缓存和碎片拓扑来访问分片数据库的系统。

[0185] 如图6所示,碎片感知的客户端应用可以做出具有碎片键和超级碎片键(例如,具有碎片键X)的连接请求212。碎片拓扑层使得能够进行对碎片/块的快速键路径访问214,包括高速缓存每个碎片的位置的碎片键范围。

[0186] 图7进一步示出根据实施例的用于使得能够使用高速缓存和碎片拓扑来访问分片数据库的系统。

[0187] 如图7所示,在该示例中,碎片拓扑可以被用于提供碎片感知的客户端应用对(在本示例中)决C2的快速键路径访问,从而绕过碎片引导器或监听器。

[0188] GetConnection API设计

[0189] 根据实施例,在OracleDatasource上定义的示例性API可以包括:

[0190] datasource.getConnectionToShard(<shard_key>)

[0191] 当数据库中有一个层级的分片并且没有涉及超级碎片键时,可以使用该API。可以使用在数据源上设置的默认用户名和密码来获取连接。碎片键是必需的参数,并且利用空的(null)或不正确的碎片键来使用该API将导致向应用抛回异常。

[0192] datasource.getconnectionToShard(<shard_key>,<super_shard_key>,<username>,<password>)

[0193] 当需要为特定一组碎片键或特定用户提取或创建连接时,可以使用该API。碎片键是必需的参数,并且利用空的或不正确的碎片键来使用该API将导致向应用抛回异常。超级碎片键可以为空。

[0194] datasource.getConnectionToShard(<shard_key>,<super_shard_key>,<username>,<password>,<label properties>)

[0195] 当需要为特定一组碎片键或特定用户提取或创建连接并且也与用户定义的标签匹配时,可以使用该API。碎片键是必需的参数,并且利用空的或不正确的碎片键来使用该API将导致向应用抛回异常。超级碎片键可以为空。

[0196] 利用碎片键的连接创建和检索

[0197] 图8示出根据实施例的、描述当连接池用于创建和维护到分片数据库的连接的池时的流程的顺序图。

[0198] 如图8所示,根据实施例,可以使用连接池(例如,UCP)来创建和维护到分片数据库的连接的池,并且支持使用针对PoolDatasource做出的getConnectionToShard API调用。

例如，如图所示，客户端应用230可以利用连接池232(例如，UCP)、数据库驱动器234(例如，JDBC)和数据库拓扑模块236连同碎片引导器或监听器238来访问分片数据库240并支持使用碎片拓扑层。当客户端应用做出(可选地具有一个或多个碎片键的)获取连接请求250时，连接池向数据库拓扑模块做出获取服务拓扑请求252，数据库拓扑模块可以返回实例列表254。

[0199] 根据实施例，如果实例列表为空，那么连接池可以向数据库驱动器做出获取连接请求260，数据库驱动器调用碎片引导器或监听器来创建连接262。当到数据库的连接被创建264时，该连接被返回到客户端应用266、268、270、272，并且碎片拓扑被相应地更新274。

[0200] 替代地，如果实例列表不为空，那么连接池可以在池中的可用连接上针对匹配实例进行查找280，并且向客户端应用返回适当的连接290。

[0201] 根据实施例，如果实例列表具有找到的可用连接的有效列表，那么连接池可以重新目的化现有连接292，并且切换该连接上的虚拟服务294。经重新目的化的连接然后可以返回给客户端应用296、298、300。

[0202] 替代地，如果实例列表没有匹配的连接，那么如上所述，连接池可以向数据库驱动器做出获取连接请求310，数据库驱动器调用碎片引导器或监听器以创建连接312。当到数据库的连接被创建314时，该连接被返回到客户端应用316、318、320、322，并且碎片拓扑再次被相应地更新326。

[0203] 图9示出根据实施例的描述当应用在没有连接池的情况下使用数据库驱动器获取到分片数据库的连接时的流程的顺序图。

[0204] 如图9所示，根据实施例，应用可以在没有连接池的情况下使用诸如JDBC驱动器的数据库驱动器来获取到分片数据库的连接。连接请求可以绕过碎片引导器或监听器，并且使用对适当碎片或块的快速键路径访问。

[0205] 如图9所示，当客户端应用做出具有一个或多个碎片键的获取连接请求330时，数据库驱动器可以直接调用碎片引导器或监听器来创建连接332，而不需要来自连接池的任何输入。当到数据库的连接被创建334时，该连接将被返回到客户端应用336、338、340，并且碎片拓扑被相应地更新342。

[0206] 图10示出根据实施例的用于使得能够使用高速缓存和碎片拓扑来访问分片数据库的过程。

[0207] 如图10所示，在步骤343处，提供了具有多个碎片并且与一个或多个数据库驱动器以及一个或多个连接池相关联的数据库，它们一起提供由客户端应用对存储在数据库处的数据的访问。

[0208] 在步骤345处，数据库驱动器被配置为使得客户端应用能够提供碎片键信息，碎片键信息被用于提供由客户端应用对数据库的适当碎片的访问，数据库驱动器包括碎片拓扑层，碎片拓扑层获知数据库内的碎片的位置的碎片键范围并将其高速缓存，用于在处理连接请求时使用。

[0209] 在步骤347处，连接池可以通过连接的碎片键来识别该连接，并且允许当从客户端应用接收到对相同碎片键的请求时重用连接。

[0210] 在步骤349处，碎片拓扑层使得后续连接请求能够绕过碎片引导器或监听器，并且代替地使用对适当碎片或块的快速键路径访问。

[0211] 通过连接池(例如,UCP)在连接上SetShardKey

[0212] 根据实施例,当客户端应用或用户尝试针对借用的连接执行setShardKey(..)时,在该连接上执行以下操作:

[0213] (1)如果碎片键映射到碎片拓扑表中的给定范围,那么查找针对键范围的虚拟服务名称。如果虚拟服务名称与连接上的虚拟服务名称相同,那么连接被创建到相同的块,并且可以被直接重用。

[0214] (2)如果碎片键映射到新虚拟服务并且该新虚拟服务存在于与连接在其上被创建的实例相同的实例上,那么可以通过将连接上的服务切换到新虚拟服务来重用连接。在连接的虚拟服务被切换之前,需要使用JDBC API关闭连接上的所有打开的工件(artifact)。如果连接上有正在进行的事务,那么操作将失败,并向用户提供异常。

[0215] (3)如果碎片键映射到存在于不同实例上的新虚拟服务,那么连接池(例如,UCP)需要例如通过以下过程来切换代理连接下的物理连接:返回到应用的连接是表示底层物理连接的代理连接对象。当连接需要切换到新的碎片实例时,连接池首先尝试在池中寻找被创建到目标虚拟服务名称的现有连接,否则它创建新的连接。包装在代理中的原始连接在关闭所有打开的工件之后返回到池。如果有正在进行的事务,那么此操作将导致异常。然后,通过使用更改代理对象下的底层物理连接的被称为setDelegate的能力,新的连接与代理连接对象相关联。用户现在可以继续使用代理对象,该代理对象现在指向由setShardKey调用中的新键指定的碎片和块。

[0216] 通过数据库驱动器(例如JDBC)在连接上SetShardKey

[0217] 根据实施例,当用户尝试使用数据库驱动器(例如,JDBC)针对连接对象执行setShardKey时,可以遵循以下过程:

[0218] (1)如果碎片键映射到碎片拓扑表中的给定范围,那么查找针对键范围的虚拟服务名称。如果虚拟服务名称与连接上的虚拟服务名称相同,那么连接被创建到相同的块,并且可以被直接重用。

[0219] (2)如果碎片键映射到新虚拟服务并且该新虚拟服务存在于与连接在其上被创建的实例相同的实例上,那么通过仅仅将连接上的服务切换到新虚拟服务就可以重用连接。在连接的虚拟服务被切换之前,需要使用数据库驱动器(例如,JDBC)API关闭连接上的所有打开的工件。如果连接上有正在进行的事务,那么操作将失败,并向用户提供异常。

[0220] (3)如果碎片键映射到存在于不同实例/碎片上的新虚拟服务,那么将向用户返回对应的错误消息。

[0221] 连接池(例如,UCP)中的连接选择

[0222] 根据实施例,可以使用以下过程在连接池(例如,UCP)中选择连接:

[0223] (1)当在池处接收到具有碎片键信息的getConnection(..)请求时,为了获得与碎片键和超级碎片键对应的服务名称,连接池可以调用由OracleDBTopology模块提供的API:

[0224] dbTopology.getServiceName(<shard_key>,<super_shard_key>)

[0225] (2)如果该块的服务名称在OracleDBTopology中尚不可用,即还没有到该特定块的连接,那么可以使用监听器创建到该碎片的新的连接。在这种情况下,连接串被修改为包括碎片和超级碎片键,并且做出连接创建请求以由监听器处置。在尝试创建连接之前,数据库驱动器(例如,JDBC)需要通过利用这两个键更新URL来处置getConnection(<shard_key

>,<super_shard_key>)。

[0226] (3)如果存在用于碎片键的现有有效映射,并且由上述API返回对应的服务名称,那么连接池将检查是否已经存在到该服务的可用连接(即,已经创建的到相同块或虚拟服务的任何连接);并且如果是这样,那么该连接可以被直接重用。

[0227] (4)如果在连接池中没有用于该服务的可用连接,并且在池中有增长的空间,那么可以考虑是否需要创建新的连接,或者是否可以重新目的化可用连接以获得到期望块的连接。

[0228] (5)如果在(4)中决定重新目的化连接,那么为了选择重新目的化的候选者,可以遵循以下过程:连接池(例如,UCP)使用OracleDBTopology API getServiceTopology(<service_name>)来获取在其上虚拟服务(块)可用的实例的列表。如果虚拟服务拓扑不存在,那么需要中止连接重新目的化,并且系统应当尝试为虚拟服务(块)创建新的连接。如果虚拟服务拓扑可用,那么获取具有与虚拟服务对应的块的实例的列表;并且选择要使用的最佳实例来选择用于重新目的化的候选连接,例如通过使用RLB算法(如果RLB算法在池上被启用的话)。在上一步骤中选择的实例/碎片上,获取所有可用连接的列表,并且然后选择来自此时具有最大数量可用连接的服务的连接,该连接被认为是用于重新目的化的最佳候选者。然后通过切换连接上的服务来重新目的化候选连接,并且该连接被返回给用户。如果切换服务由于任何原因而失败,那么回落到从上面提供的实例的列表中选择实例的随机算法,并且然后重复相同的过程并尝试重新目的化连接。如果由于某种原因使用随机算法重新目的化连接失败,那么如果池中有增长的空间,就尝试创建到由碎片键指定的块的新连接,否则就报告错误。

[0229] 根据实施例,通过使用允许用户传入连接标签以及碎片键的getConnection,可以在使用碎片键从池中检索连接之上或以上支持连接标记。通常的连接选择包括首先通过RLB和亲和度(affinity)的选择,之后是通过标记匹配(或成本)。在分片支持的情况下,系统可以首先通过碎片键(虚拟服务)选择连接,并且然后通过RLB/亲和度选择连接,并且连接标记算法将照常应用。

[0230] 连接池(例如,UCP)中的连接存储和查找

[0231] 根据实施例,一旦创建了连接,连接池(例如,UCP)就将尝试提取与在其上创建该连接的块的内部虚拟服务对应的实际“服务”名称并将它放置在该连接的连接请求标识符中。连接请求标识符将用于从UCP的连接存储中查找和借用具有正确服务名称(块)的连接。当在池中未获得匹配的连接并且连接服务切换是迫切的时,服务名称也将用于确定用于重新目的化的最佳候选连接。

[0232] OracleDBTopology模块

[0233] 图11示出根据实施例的数据库拓扑类设计350。

[0234] 根据实施例,OracleDBTopology模块可以放置在数据库驱动器(例如,JDBC)内,并且包含用于数据库的所有拓扑数据,例如用于池或使用数据库驱动器的应用所引用的所有服务和碎片键的服务到实例的映射以及碎片键到服务的映射。

[0235] 根据实施例,实例OracleDBTopology实例与OracleDataSource实例相关联,并且它们之间存在1:1映射。OracleDBTopology中存在的操作和数据可以由以下各方使用:由数据库驱动器的用户使用以便查找碎片或服务拓扑;由连接池(例如,UCP)使用以基于高速缓

存的拓扑来高效地路由连接请求;由自定义池实现方案使用,自定义池实现方案可以利用该模块来允许池使用拓扑数据并维护到分片数据库的连接的池。

[0236] 根据实施例,当使用利用其中一个getConnection()API的OracleDataSource创建第一个连接时,OracleDBTopology可以被延迟地(lazily)初始化,连接对象用于获取初始化OracleDBTopology模块所需的信息。所创建的连接用于查找表LOCAL_CHUNK_TYPES表。列SHARD_TYPE和SUPER_TYPE被读取以确定分片和超级分片方法。这些值作为属性存储在OracleDBTopology实例中。DEF_VERSION也被提取并放置在客户端侧。

[0237] 服务拓扑

[0238] 图12示出根据实施例的服务拓扑类设计360。

[0239] 根据实施例,ServiceTopologyMap包含虚拟服务名称到服务当前正运行于其上的实例的列表之间的映射,该映射是1:n映射。当创建每个连接以及在OracleDbTopology上调用方法updatedOnCreateConnection()时以及随着创建每个连接以及在OracleDbTopology上调用方法updatedOnCreateConnection(),该映射被逐步地填充。

[0240] 根据实施例,当前连接在其上被创建的实例名称和虚拟服务名称二者都存在于sys_context中。在所创建的每个新连接上,系统可以检查虚拟服务名称是否已经在映射中,并且如果不是这样,那么添加用于虚拟服务名称和实例的新条目。如果虚拟服务名称已经在映射中,并且实例名称也存在于服务的实例列表中,那么不需要进一步的操作;否则将实例添加到实例列表。

[0241] 根据实施例,ShardTopologyMap提供接口并定义可以用于将块键范围或列表映射到虚拟服务名称的数据结构。

[0242] 接口和API

[0243] 根据实施例,下面描述示例性应用程序接口(API),包括数据库驱动器(例如,JDBC)API和连接池(例如UCP)API。根据其它实施例和示例,可以使用其它类型的API或接口。

[0244] 附录A:数据库驱动器(例如JDBC)API

[0245] 根据实施例,下面描述示例性数据库驱动器(例如,JDBC)API。根据其它类型的数据库驱动器,可以使用其它类型的数据库驱动器API或接口。

[0246] 碎片键接口

```
public interface ShardKey extends Comparable<ShardKey>{
  /**
   * Used to compare two shard keys. If the shard keys are
   * compound the corresponding sub-keys in two keys will be
   * compared.
   *
   * @param o
   *          ShardKey to which this Shard key is to be compared.
   * @return -1, 0 or 1 as this ShardKey is less than, equal
   *          to, or greater than the shard key that is passed in
   *          as a method parameter
   * @see java.lang.Comparable#compareTo(java.lang.Object)
   */
  int compareTo(ShardKey o);
}
```

[0247]

[0248] 连接构建器接口

```
/**
 * Builder class for building connection objects with additional parameters
 * other than just the username and password. To use the builder, the
 * corresponding builder method needs to be called for each parameter that needs
 * to be part of the connection request followed by a build() method.
 * The order in which the builder methods are called is not important.
 * However if the same builder attribute is applied more than once, only
 * the most recent value will be considered while building the connection.
 * The builder's build() method can be called only once on a builder object.
 *
 * Example usage :
 *   OracleDataSource ods = new OracleDataSource();
 *   ..//set the required properties on the datasource
 *   Connection conn = ods.createConnectionBuilder()
 *                        .user("user")
 *                        .password("password")
 *                        .proxyClientName("proxy_client")
```

[0249]

```
 *                      .serviceName("service_name")
 *                      .build();
 *
* @param <B>
 *          Type of connection builder
 * @param <S>
 *          Type of connections created using this builder

 */
public interface ConnectionBuilder<B extends ConnectionBuilder<B, S>, S>
    extends Builder<S> {
  /**
   * @param user
   * @return This connection builder object
   */
  B user(String user);

  /**
   * @param password
   * @return This connection builder object
   */
  B password(String password);

  /**
   * @param serviceName
   * @return This connection builder object
   */
  B serviceName(String serviceName);

  /**
   * @param shardKey
   *          Shard Key object that needs to be part of connection request
   * @return This instance of the connection builder.
   */
  B shardKey(ShardKey shardKey);

  /**
   * @param shardGroupKey
   *          Shard Group Key object that needs to be part of connection request
   * @return This instance of the connection builder.
   */
  B shardGroupKey(ShardKey shardGroupKey);
}
```

[0250]

[0251] OracleConnectionBuilder

```
class oracle.jdbc.pool.OracleConnectionBuilder
    implements oracle.jdbc.ConnectionBuilder<OracleConnectionBuilder,Connection>
```

[0252]

```
{
    //Oracle's Implementation class for ConnectionBuilder.
}
```

[0253] OracleDataSource

```
class oracle.jdbc.pool.OracleDataSource
{
```

[0254]

```
   OracleConnectionBuilder createConnectionBuilder() { }
   OracleShardKeyBuilder createShardKeyBuilder();
}
```

[0255] OracleConnection

[0256]

```
interface oracle.jdbc.OracleConnection
{
  /**
   * Used to check the validity of the connection and if the shard keys passed
   * to this method are valid for this connection.
   *
   * @param shardKey
   *          shard key to be validated against this connection
   * @param groupKey
   *          shard group key to be validated against this connection
   * @param timeout
   *          time in seconds before which the validation process is expected to
   *          be completed,else the validation process is aborted.
   * @return true if the connection is valid and the shard keys are valid to be
   *         set on this connection.
   * @throws SQLException
   *         if there is any exception while performing this validation.
   */
  boolean isValid(ShardKey shardKey, ShardKey groupKey, int timeout)
      throws SQLException;

  /**
   * Used to set the shard key and the shard group key on this connection.
   *
   * @param shardKey
   *          shard key to be set on this connection
   * @param groupKey
   *          shard group key to be set on this connection
   * @throws SQLException
   *           if there is an exception while setting the shard keys
   *           on this connection. In this case the connection will continue to
   *           be associated with the shard keys that was set on this connection
   *           before this method was called.
   */
  void setShardKey(ShardKey shardKey, ShardKey groupKey) throws SQLException;}
}
```

[0257]

[0258] 附录B:连接池(例如UCP)API

[0259] 根据实施例,下面描述示例性连接池(例如UCP)API。根据其它类型的连接池,可以使用其它类型的连接池API或接口。

[0260] UCP连接构建器

```
/**
 * UCP's Connection Builder class for building connection objects with additional
parameters
 * other than just the username,password and labels. To use the builder, the
 * corresponding builder method needs to be called for each parameter that needs
 * to be part of the connection request followed by a build() method. The order
 * in which the builder methods are called is not important. However if the same
 * builder attribute is applied more than once, only the most recent value will
 * be considered while building the connection. The builder's build() method can
 * be called only once on a builder object.
 *
 * Example usage :
[0261] * PoolDataSource pds = PoolDataSourceFactory.getPoolDataSource();
 * ..//set the required properties on the datasource
 *
 * Connection conn = pds.createConnectionBuilder()
 *                   .user("user")
 *                   .password("password")
*                    .serviceName("service_name")
 *                   .build();
 *
 * @param <C>
 *          Type of connections created using this builder
 * @param <B>
 *          Type of connection builder
```

```
*/
public interface oracle.ucp.jdbc.UCPConnectionBuilder extends
    oracle.jdbc.ConnectionBuilder<UCPConnectionBuilder, Connection> {

/**
   * Sets the user attribute on the builder
   *
   * @param user
   *          - the database user on whose behalf the connection is being made
   * @return - this builder object
   */
  @Override
  public UCPConnectionBuilder user(String user) ;

/**
   * Sets the password attribute on the builder
   *
   * @param password
   *          - the user's password
   * @return - this builder object
   */
[0262]  @Override
  public UCPConnectionBuilder password(String password) ;

  /**
   * Sets the labels attribute on the builder
   *
   * @param labels
   *          - The requested connection labels.
   * @return - this builder object
   */
  public UCPConnectionBuilder labels(Properties labels) ;

  /**
   * @param serviceName to retrieve the connection
   * @return this connection builder instance
   */
  public UCPConnectionBuilder serviceName(String serviceName);

  /**
   * @param shardKey
   *          Shard Key object that needs to be part of connection request
   * @return This instance of the connection builder.
```

```
     */
    public UCPConnectionBuilder shardKey(ShardKey shardKey) ;

    /**
     * @param shardGroupKey
     *          Shard Group Key object that needs to be part of connection request
     * @return This instance of the connection builder.
     */
    public UCPConnectionBuilder shardGroupKey(ShardKey shardGroupKey);

    /**
     * @return Connection built considering the builder attributes
     * @throws SQLException if there is a failure in building the connection.
     */
   public Connection build() throws SQLException;

    /**
     * Sets the labels attribute on the builder
     *
     * @param labels
     *          - The requested connection labels.
     * @return - this builder object
     */
    ConnectionBuilder labels(Properties labels);
}
```

[0263]

[0264] PoolDataSource接口

```
Interface oracle.ucp.jdbc.PoolDataSource
{
  /**
   * @return UCPConnectionBuilder that can help build Connection with multiple
   *          parameters other than just the user,password and labels.
   */
  public UCPConnectionBuilder createConnectionBuilder();

  /**
   * @return OracleShardKeyBuilder that can help build Shard Keys.
   */
  public OracleShardKeyBuilder createShardKeyBuilder();
}
                                                                              }
```

[0265]

[0266] 本发明的实施例可以使用一个或多个常规的通用或专用数字计算机、计算设备、机器或微处理器来方便地实现，所述数字计算机、计算设备、机器或微处理器包括根据本公开的教导进行编程的一个或多个处理器、存储器和/或计算机可读存储介质。如本领域技术人员将清楚的，基于本公开的教导，适当的软件编码可以由熟练的程序员容易地准备。

[0267] 在一些实施例中，本发明包括计算机程序产品，计算机程序产品是其上/其中存储有指令的非暂态存储介质或(一个或多个)计算机可读介质，指令可以被用来对计算机进行编程，以执行本发明的任何过程。存储介质的示例可以包括但不限于任何类型的盘(包括软盘、光盘、DVD、CD-ROM、微驱动器和磁光盘)、ROM、RAM、EPROM、EEPROM、DRAM、VRAM、闪存存储

器设备，磁卡或光卡、纳米系统(包括分子存储器IC)，或适于存储指令和/或数据的任何类型的介质或设备。

[0268] 本发明的实施例的前述描述是为了说明和描述的目的而提供的。它不旨在是详尽的或者要将本发明限制到所公开的精确形式。许多修改和变化对于本领域技术人员将是清楚的。实施例被选择和描述以便最好地解释本发明的原理及其实际应用，由此使本领域的其他技术人员能够理解用于各种实施例的本发明以及具有适于预期的特定用途的各种修改的本发明。

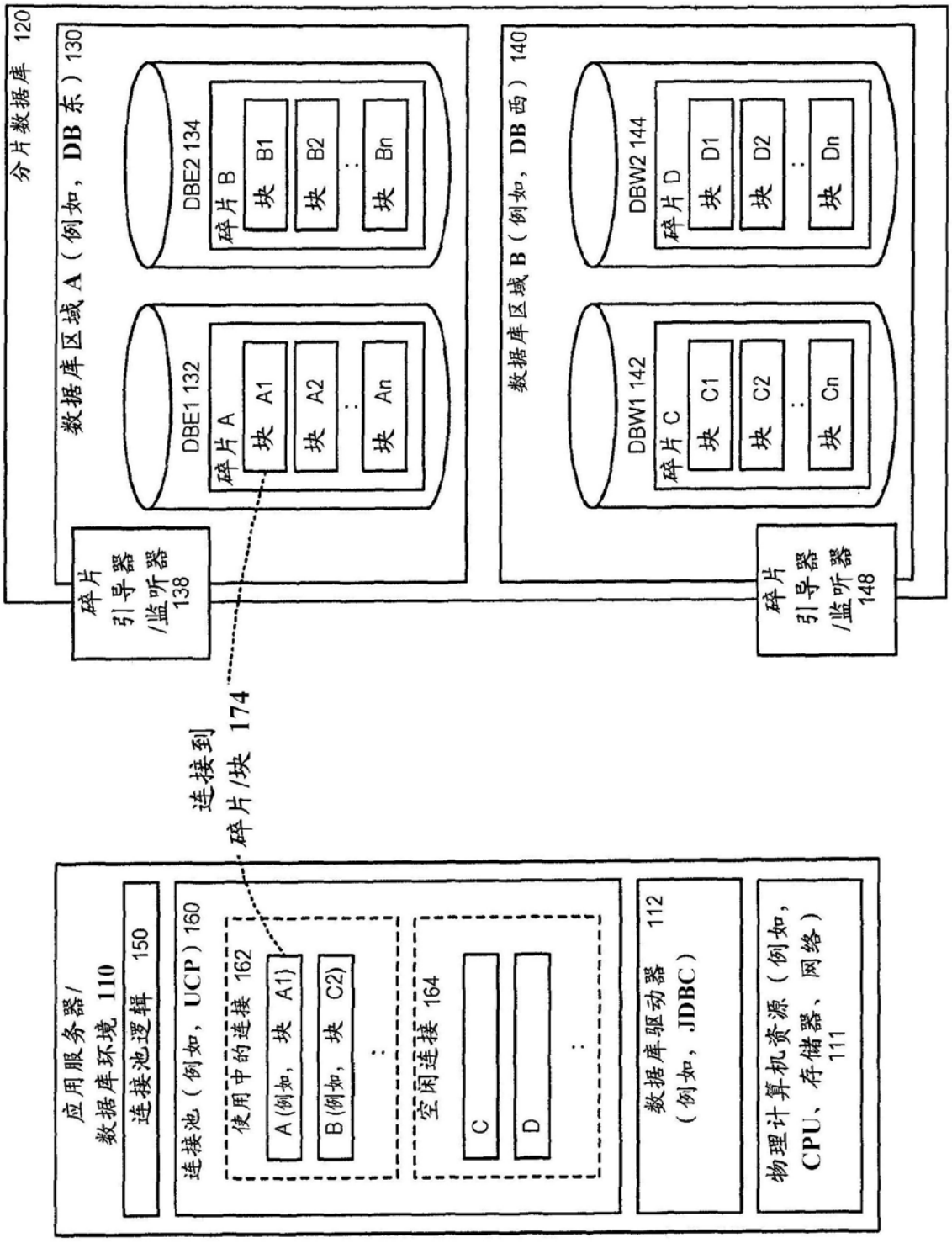

分片数据库 120
数据库区域 A（例如，DB 东）130
DBE1 132
碎片 A
块 A1
块 A2
块 An
DBE2 134
碎片 B
块 B1
块 B2
块 Bn
碎片引导器/监听器 138
数据库区域 B（例如，DB 西）140
DBW1 142
碎片 C
块 C1
块 C2
块 Cn
DBW2 144
碎片 D
块 D1
块 D2
块 Dn
碎片引导器/监听器 148
连接到碎片/块 174
应用服务器/数据库环境 110
连接池逻辑 150
连接池（例如，UCP）160
使用中的连接 162
A（例如，块 A1）
B（例如，块 C2）
空闲连接 164
C
D
数据库驱动器 112
（例如，JDBC）
物理计算机资源（例如，CPU、存储器、网络）111

图1

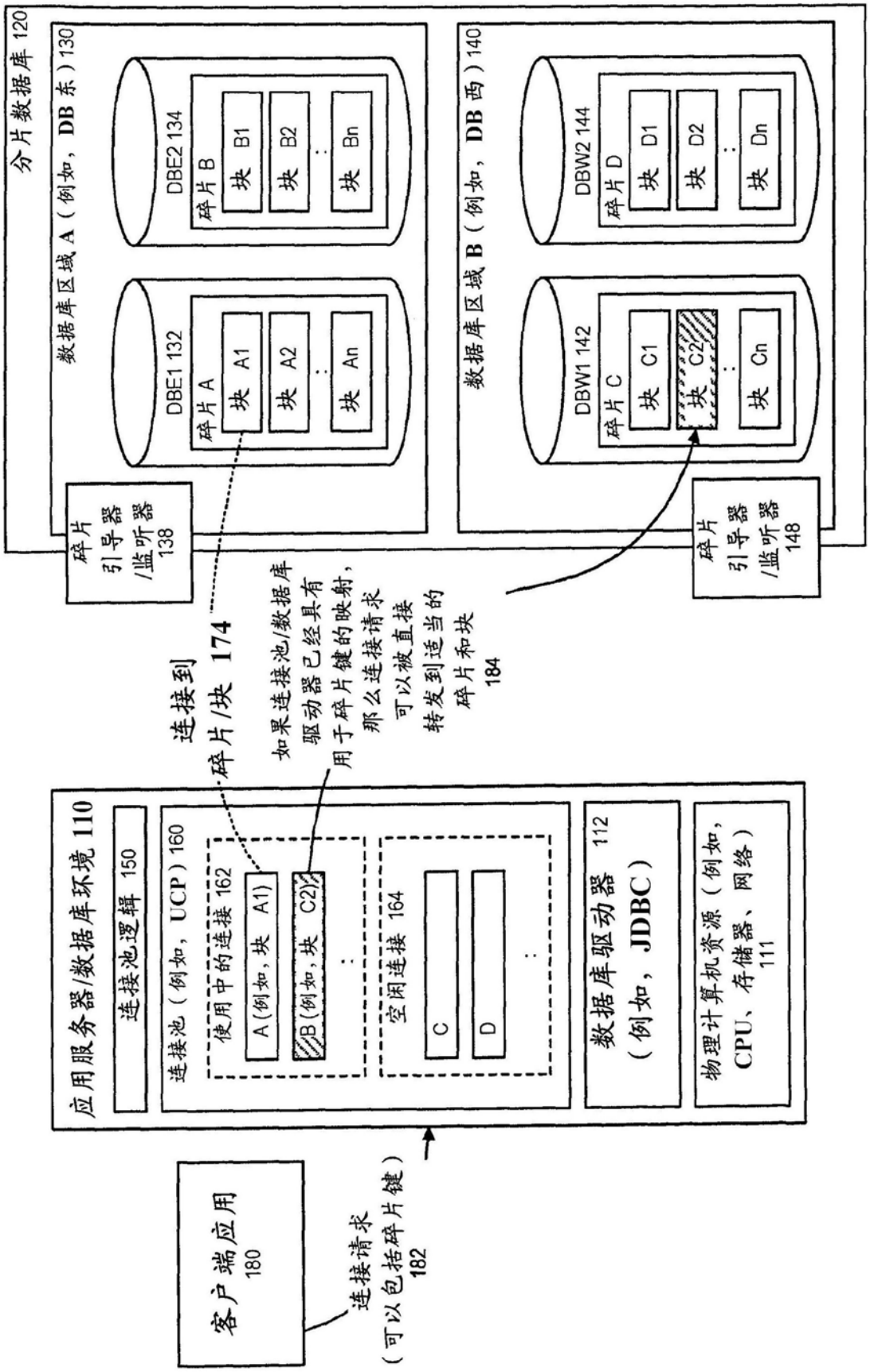

分片数据库 120
数据库区域 A（例如，DB 东）130
DBE1 132
碎片 A
块 A1
块 A2
块 An
DBE2 134
碎片 B
块 B1
块 B2
块 Bn
数据库区域 B（例如，DB 西）140
DBW1 142
碎片 C
块 C1
块 C2
块 Cn
DBW2 144
碎片 D
块 D1
块 D2
块 Dn
碎片引导器/监听器 138
碎片引导器/监听器 148
连接到碎片/块 174
如果连接池/数据库驱动器已经具有用于碎片键的映射，那么连接请求可以被直接转发到适当的碎片和块 184
应用服务器/数据库环境 110
连接池逻辑 150
连接池（例如，UCP）160
使用中的连接 162
A(例如，块 A1)
B(例如，块 C2)
空闲连接 164
C
D
数据库驱动器 112
（例如，JDBC）
物理计算机资源（例如，CPU、存储器、网络）111
客户端应用 180
连接请求（可以包括碎片键）182

图2

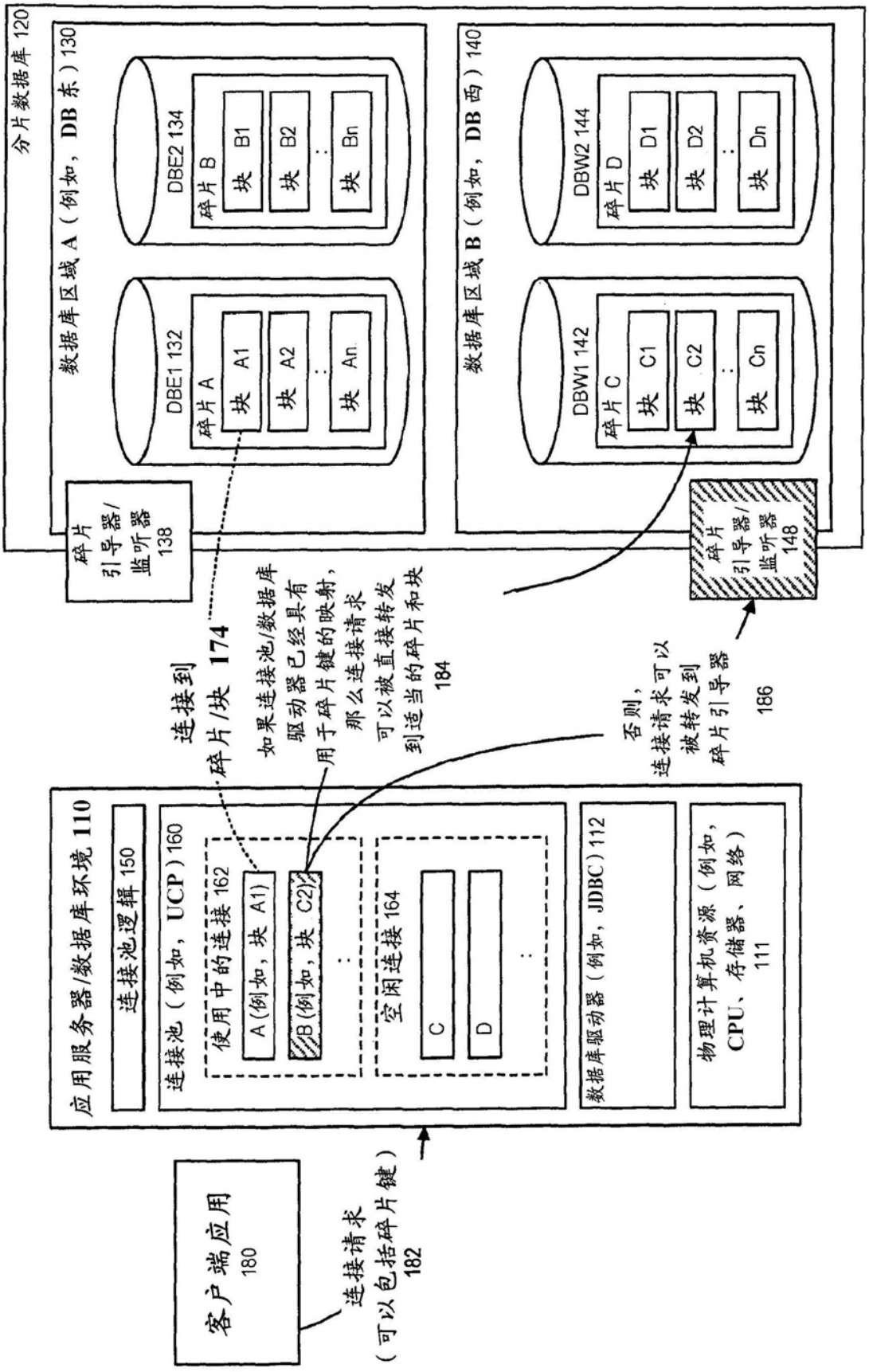

分片数据库 120
数据库区域 A（例如，DB 东）130
DBE1 132
碎片 A
块 A1
块 A2
块 An
DBE2 134
碎片 B
块 B1
块 B2
块 Bn
数据库区域 B（例如，DB 西）140
DBW1 142
碎片 C
块 C1
块 C2
块 Cn
DBW2 144
碎片 D
块 D1
块 D2
块 Dn
碎片引导器/监听器 138
碎片引导器/监听器 148
应用服务器/数据库环境 110
连接池逻辑 150
连接池（例如，UCP）160
使用中的连接 162
A（例如，块 A1）
B（例如，块 C2）
空闲连接 164
C
D
数据库驱动器（例如，JDBC）112
物理计算机资源（例如，CPU、存储器、网络）111
客户端应用 180
连接请求（可以包括碎片键）182
连接到碎片/块 174
如果连接池/数据库驱动器已经具有用于碎片键的映射，那么连接请求可以被直接转发到适当的碎片和块 184
否则，连接请求可以被转发到碎片引导器 186

图3

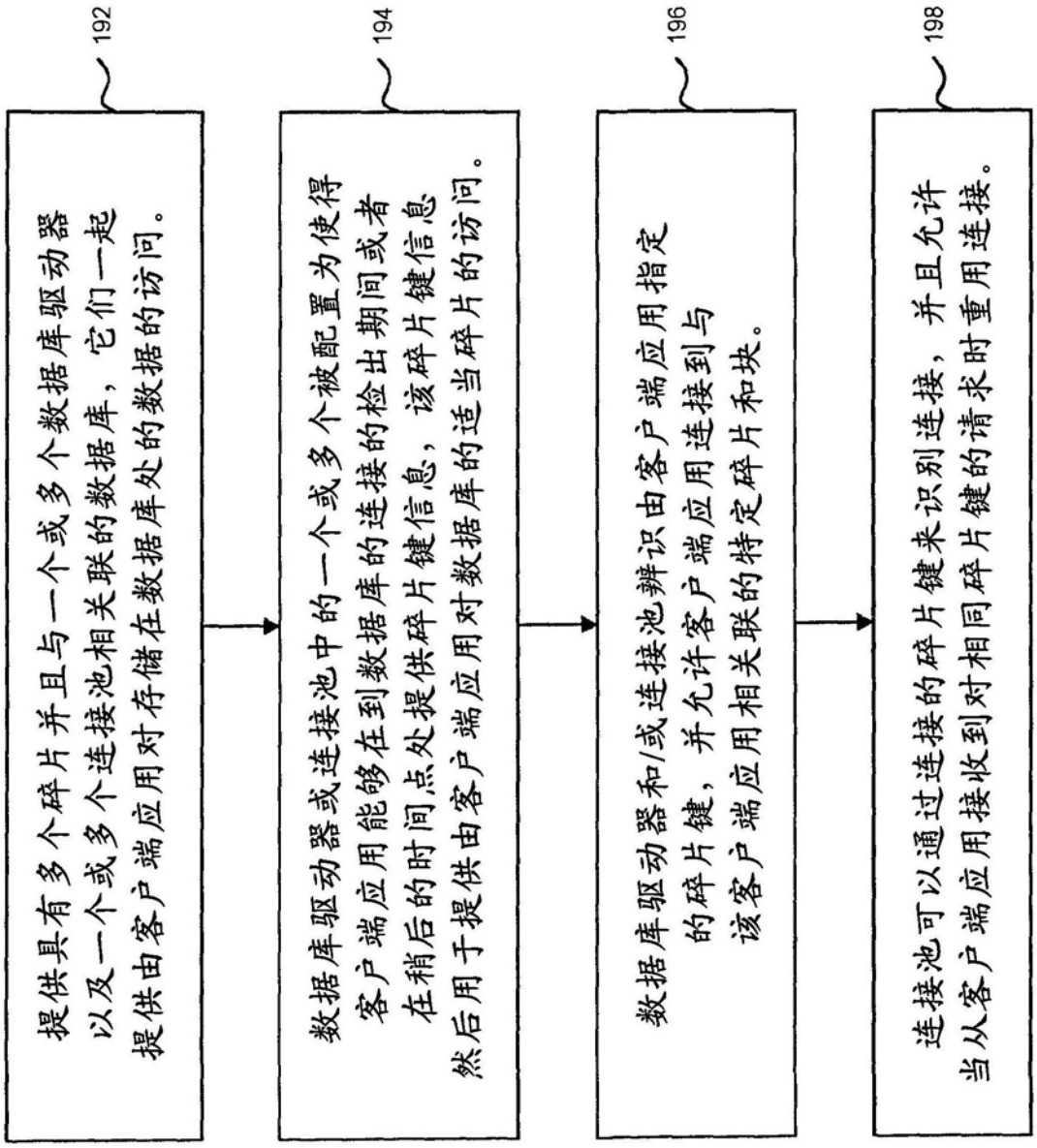

192
提供具有多个碎片并且与一个或多个数据库驱动器以及一个或多个连接池相关联的数据库，它们一起提供由客户端应用对存储在数据库处的数据的访问。
194
数据库驱动器或连接池中的一个或多个被配置为使得客户端应用能够在到数据库的连接的检出期间或者在稍后的时间点处提供碎片键信息，该碎片键信息然后用于提供由客户端应用对数据库的适当碎片的访问。
196
数据库驱动器和/或连接池辨识由客户端应用指定的碎片键，并允许客户端应用连接到与该客户端应用相关联的特定碎片和块。
198
连接池可以通过连接的碎片键来识别连接，并且允许当从客户端应用接收到对相同碎片键的请求时重用连接。

图4

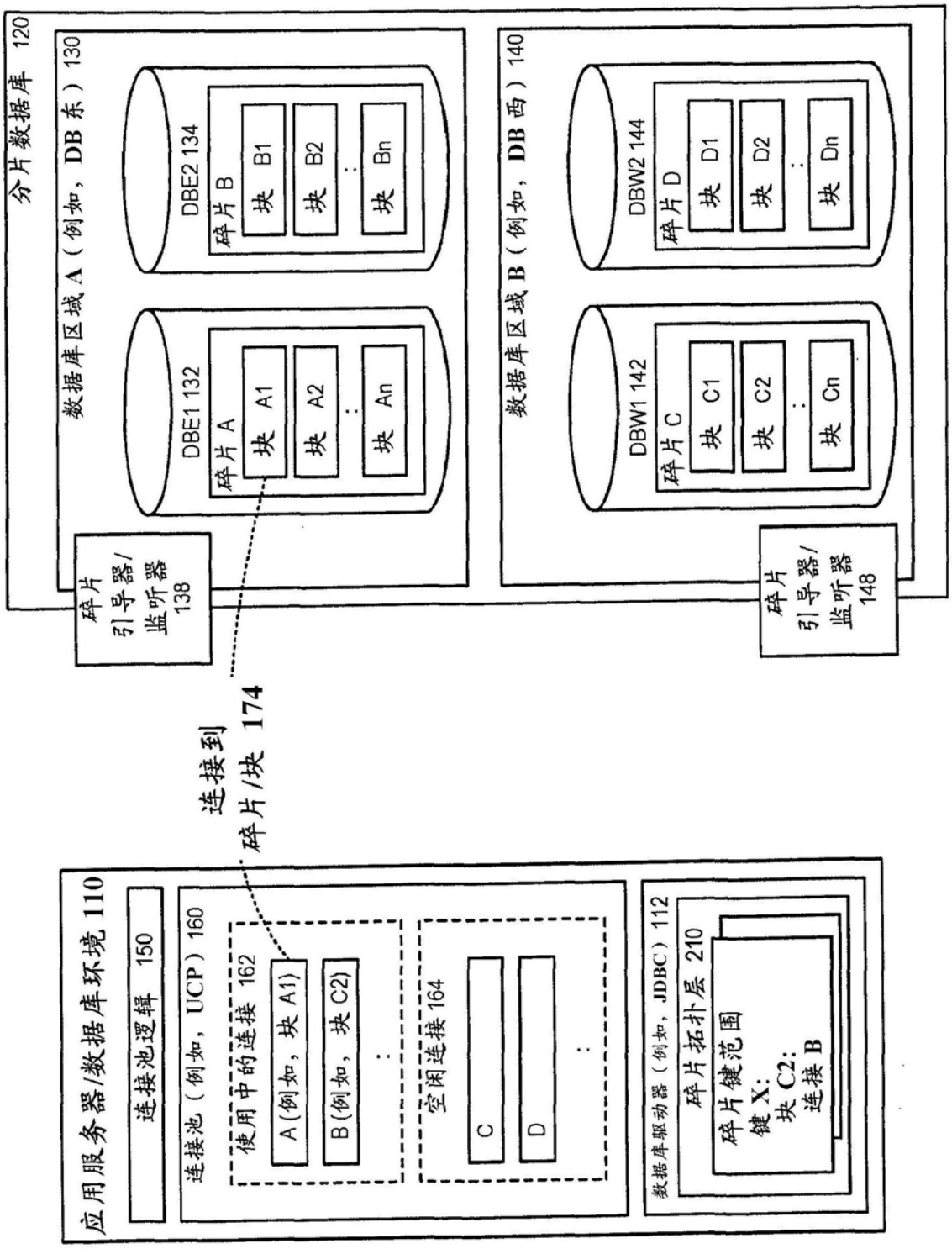

分片数据库 120
数据库区域 A（例如，DB 东）130
DBE1 132
碎片 A
块 A1
块 A2
块 An
DBE2 134
碎片 B
块 B1
块 B2
块 Bn
碎片引导器/监听器 138
数据库区域 B（例如，DB 西）140
DBW1 142
碎片 C
块 C1
块 C2
块 Cn
DBW2 144
碎片 D
块 D1
块 D2
块 Dn
碎片引导器/监听器 148
连接到碎片/块 174
应用服务器/数据库环境 110
连接池逻辑 150
连接池（例如，UCP）160
使用中的连接 162
A（例如，块 A1）
B（例如，块 C2）
空闲连接 164
C
D
数据库驱动器（例如，JDBC）112
碎片拓扑层 210
碎片键范围
键 X:
块 C2:
连接 B

图5

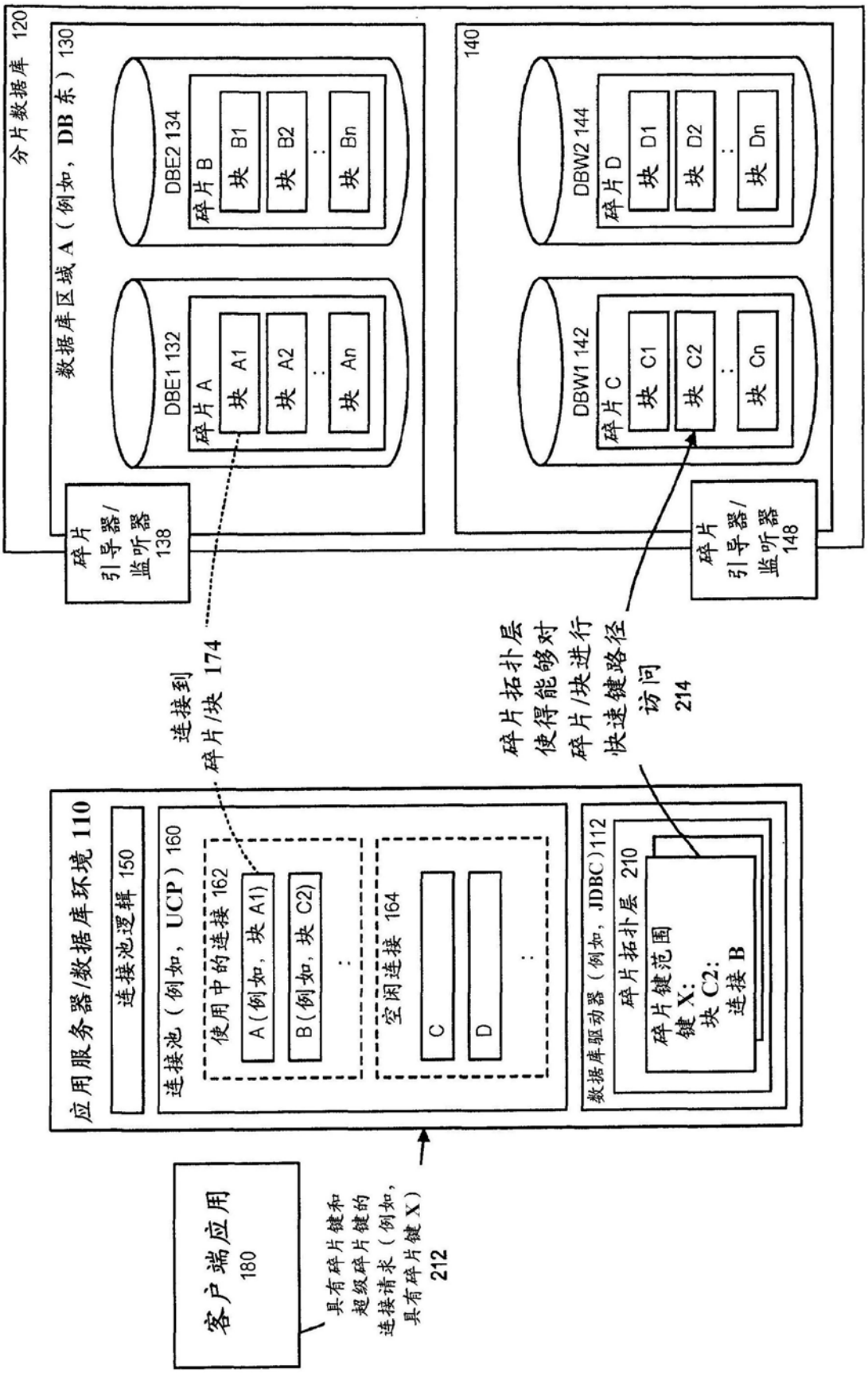

分片数据库 120
数据库区域 A（例如，DB 东）130
DBE1 132
碎片 A
块 A1
块 A2
块 An
DBE2 134
碎片 B
块 B1
块 B2
块 Bn
140
DBW1 142
碎片 C
块 C1
块 C2
块 Cn
DBW2 144
碎片 D
块 D1
块 D2
块 Dn
碎片引导器/监听器 138
碎片引导器/监听器 148
连接到碎片/块 174
碎片拓扑层使得能够对碎片/块进行快速键路径访问 214
应用服务器/数据库环境 110
连接池逻辑 150
连接池（例如，UCP）160
使用中的连接 162
A（例如，块 A1）
B（例如，块 C2）
空闲连接 164
C
D
数据库驱动器（例如，JDBC）112
碎片拓扑层 210
碎片键范围
键 X:
块 C2:
连接 B
客户端应用 180
具有碎片键和超级碎片键的连接请求（例如，具有碎片键 X）212

图6

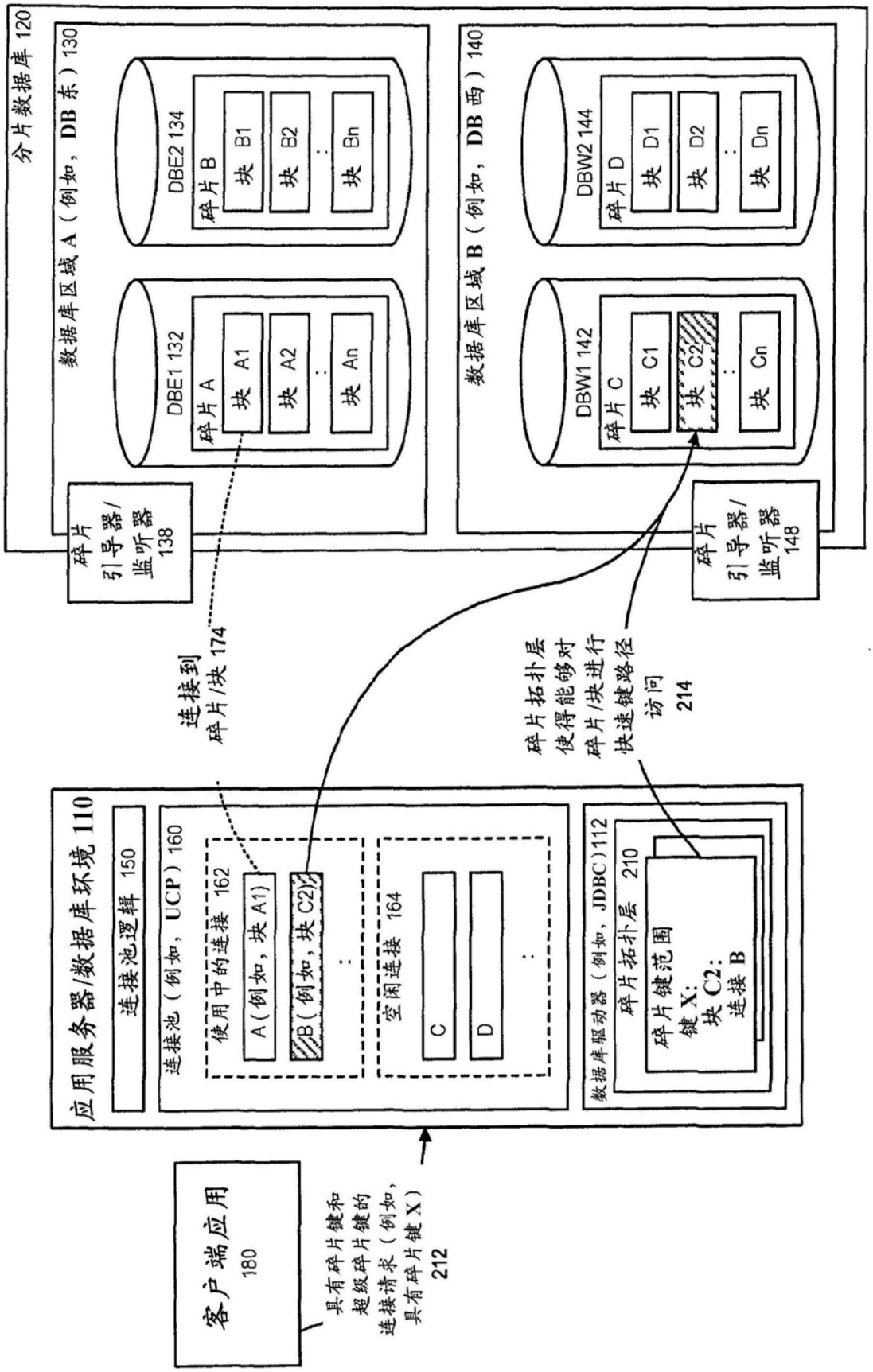

分片数据库 120
数据库区域 A（例如，DB 东）130
DBE1 132
碎片 A
块 A1
块 A2
块 An
DBE2 134
碎片 B
块 B1
块 B2
块 Bn
数据库区域 B（例如，DB 西）140
DBW1 142
碎片 C
块 C1
块 C2
块 Cn
DBW2 144
碎片 D
块 D1
块 D2
块 Dn
碎片引导器/监听器 138
碎片引导器/监听器 148
连接到碎片/块 174
碎片拓扑层使得能够对碎片/块进行快速键路径访问 214
应用服务器/数据库环境 110
连接池逻辑 150
连接池（例如，UCP）160
使用中的连接 162
A（例如，块 A1）
B（例如，块 C2）
空闲连接 164
C
D
数据库驱动器（例如，JDBC）112
碎片拓扑层 210
碎片键范围
键 X:
块 C2:
连接 B
客户端应用 180
具有碎片键和超级碎片键的连接请求（例如，具有碎片键 X）212

图7

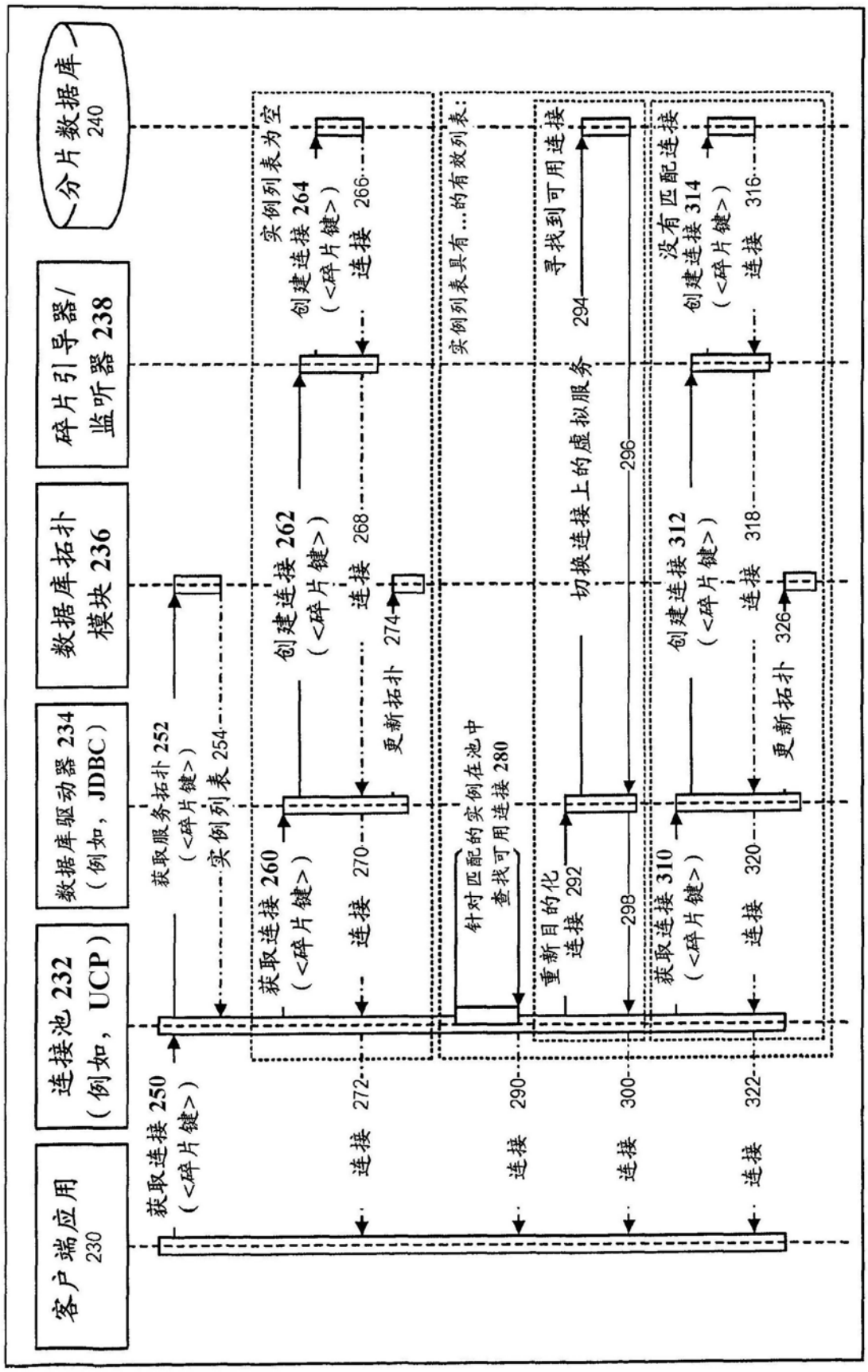

客户端应用 230
连接池 232 (例如，UCP)
数据库驱动器 234 (例如，JDBC)
数据库拓扑模块 236
碎片引导器/监听器 238
分片数据库 240
获取连接 250 (<碎片键>)
连接 272
获取服务拓扑 252 (<碎片键>)
实例列表 254
获取连接 260 (<碎片键>)
连接 270
创建连接 262 (<碎片键>)
连接 268
更新拓扑 274
实例列表为空
创建连接 264 (<碎片键>)
连接 266
实例列表具有..的有效列表:
针对匹配的实例在池中查找可用连接 280
连接 290
重新目的化连接 292
连接 298
切换连接上的虚拟服务 296
寻找到可用连接
连接 300
获取连接 310 (<碎片键>)
连接 320
创建连接 312 (<碎片键>)
连接 318
更新拓扑 326
没有匹配连接
创建连接 314 (<碎片键>)
连接 316
连接 322

图8

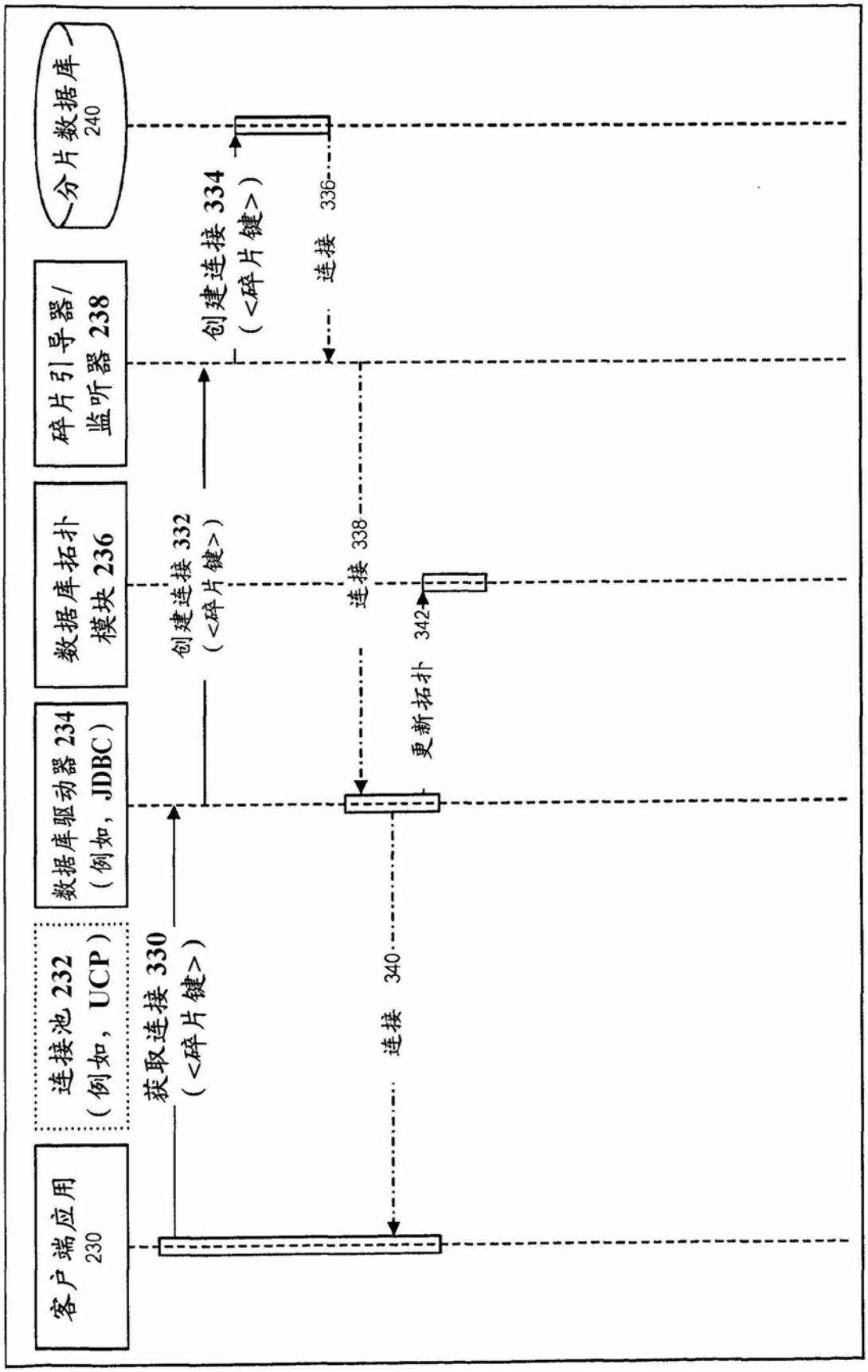

客户端应用 230
连接池 232 (例如，UCP)
数据库驱动器 234 (例如，JDBC)
数据库拓扑模块 236
碎片引导器/监听器 238
分片数据库 240
获取连接 330 (<碎片键>)
创建连接 332 (<碎片键>)
创建连接 334 (<碎片键>)
连接 336
连接 338
连接 340
更新拓扑 342

图9

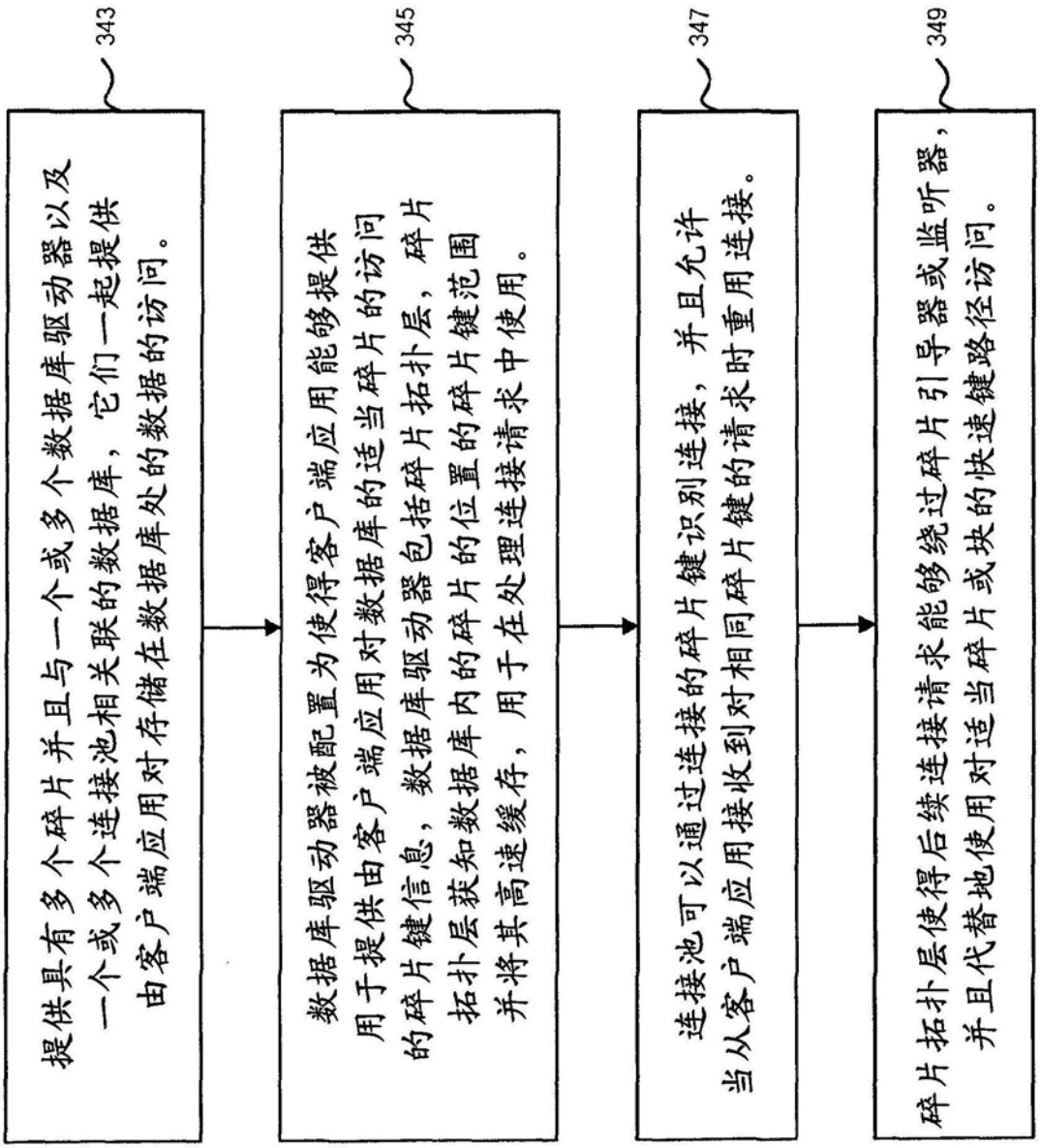

343
提供具有多个碎片并且与一个或多个数据库驱动器以及一个或多个连接池相关联的数据库，它们一起提供由客户端应用对存储在数据库处的数据的访问。
345
数据库驱动器被配置为使得客户端应用能够提供用于提供由客户端应用对数据库的适当碎片的访问的碎片键信息，数据库驱动器包括碎片拓扑层，碎片拓扑层获知数据库内的碎片的位置的碎片键范围并将其高速缓存，用于在处理连接请求中使用。
347
连接池可以通过连接的碎片键识别连接，并且允许当从客户端应用接收到对相同碎片键的请求时重用连接。
349
碎片拓扑层使得后续连接请求能够绕过碎片引导器或监听器，并且代替地使用对适当碎片或块的快速键路径访问。

图10

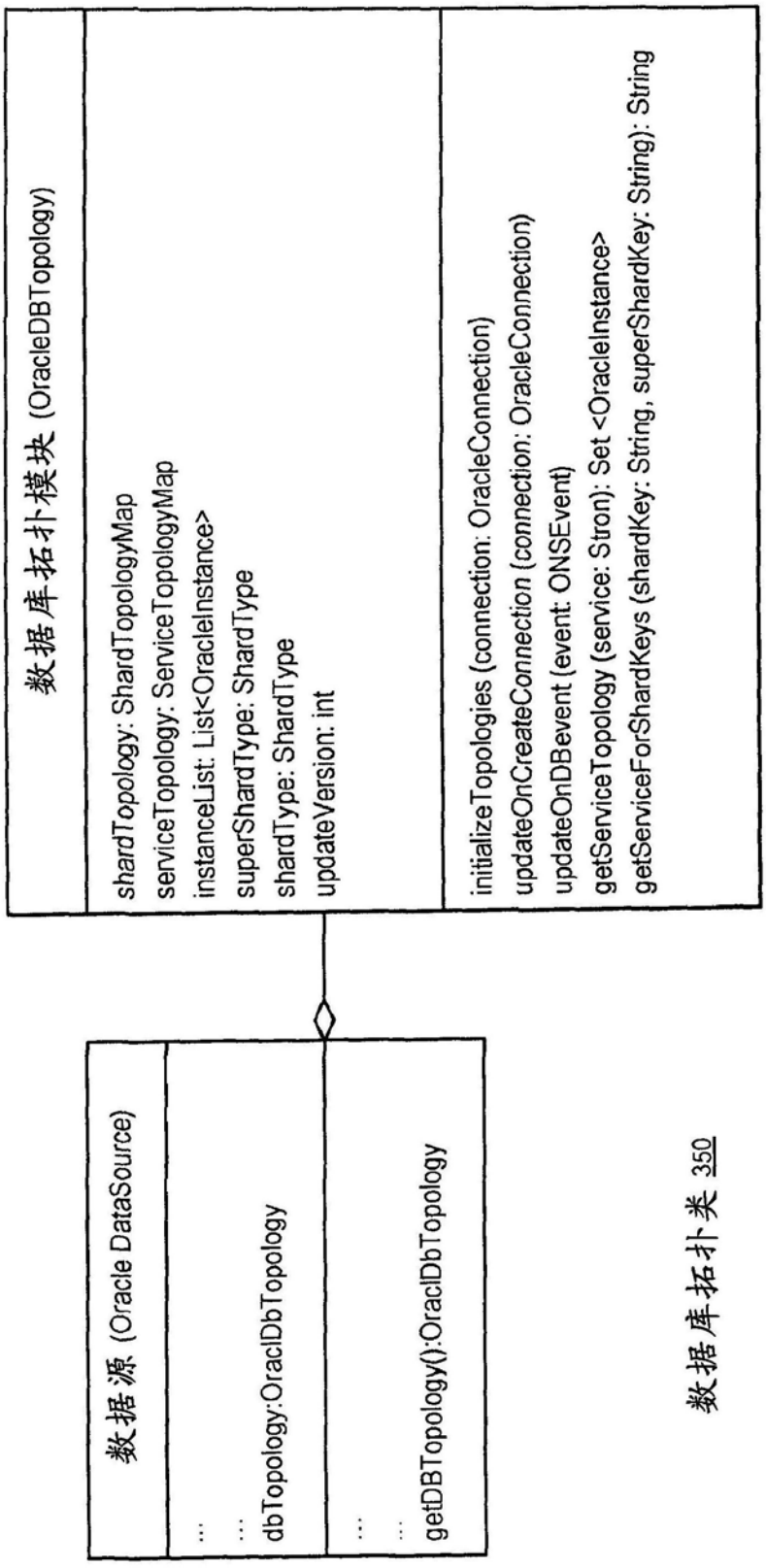

数据库拓扑模块 (OracleDBTopology)
shardTopology: ShardTopologyMap
serviceTopology: ServiceTopologyMap
instanceList: List<OracleInstance>
superShardType: ShardType
shardType: ShardType
updateVersion: int
initializeTopologies (connection: OracleConnection)
updateOnCreateConnection (connection: OracleConnection)
updateOnDBEvent (event: ONSEvent)
getServiceTopology (service: Stron): Set <OracleInstance>
getServiceForShardKeys (shardKey: String, superShardKey: String): String
数据源 (Oracle DataSource)
...
...
dbTopology:OraclDbTopology
...
...
getDBTopology():OraclDbTopology
数据库拓扑类 350

图11

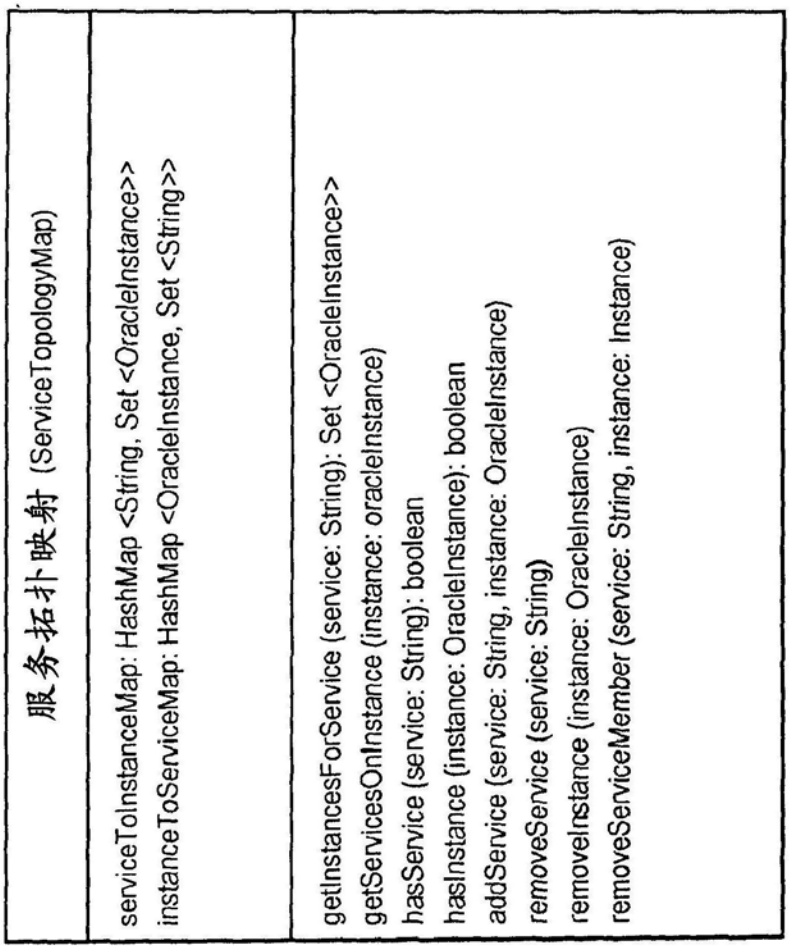

服务拓扑映射 (ServiceTopologyMap)
serviceToInstanceMap: HashMap <String, Set <OracleInstance>>
instanceToServiceMap: HashMap <OracleInstance, Set <String>>
getInstancesForService (service: String): Set <OracleInstance>>
getServicesOnInstance (instance: oracleInstance)
hasService (service: String): boolean
hasInstance (instance: OracleInstance): boolean
addService (service: String, instance: OracleInstance)
removeService (service: String)
removeInstance (instance: OracleInstance)
removeServiceMember (service: String, instance: Instance)
服务拓扑类 360

图12