



(12)发明专利

(10)授权公告号 CN 109643255 B

(45)授权公告日 2020.09.08

(21)申请号 201780049079.9

(22)申请日 2017.08.08

(65)同一申请的已公布的文献号
申请公布号 CN 109643255 A

(43)申请公布日 2019.04.16

(30)优先权数据

62/374,449 2016.08.12 US

15/668,469 2017.08.03 US

(85)PCT国际申请进入国家阶段日
2019.02.03

(86)PCT国际申请的申请数据
PCT/CN2017/096505 2017.08.08

(87)PCT国际申请的公布数据
W02018/028580 EN 2018.02.15

(73)专利权人 华为技术有限公司

地址 518129 广东省深圳市龙岗区坂田华为总部办公楼

专利权人 芝加哥大学

(72)发明人 陆善 刘浩鹏 李光普
哈里迪·古纳维 田琛 叶枫

(51)Int.Cl.
G06F 11/07(2006.01)

(56)对比文件

US 2014380101 A1,2014.12.25

US 2012278658 A1,2012.11.01

CN 103377128 A,2013.10.30

US 2014359577 A1,2014.12.04

审查员 刘凤霞

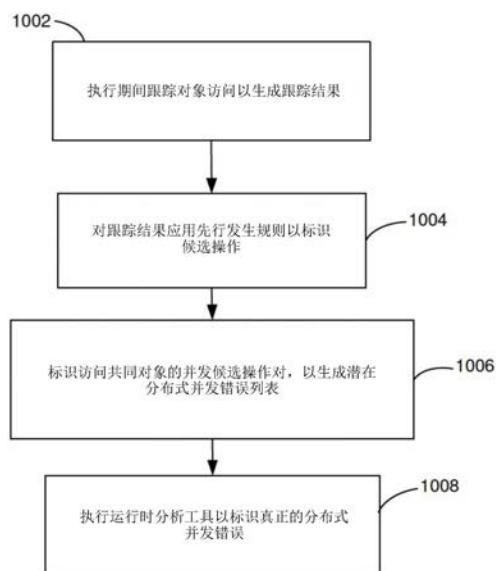
权利要求书4页 说明书19页 附图11页

(54)发明名称

在云系统中自动检测分布式并发错误

(57)摘要

一种在分布式云计算系统中检测分布式并发错误的方法,包括:在涉及进程间通讯的函数中跟踪访问对象的操作,并对所述跟踪操作应用一套先行发生规则。分析所述跟踪操作,以标识访问共同对象的并发操作,生成潜在分布式并发错误(DCbug)列表。删减所述DCbug列表,去除只在本地产生影响且不产生运行时错误的DCbug。



1. 一种在包括多个组件计算机的分布式计算系统中检测分布式并发错误的方法,其特征在于,所述方法包括:

在运行所述分布式计算系统期间跟踪访问对象的操作,以生成跟踪结果;

对所述跟踪结果应用预定的一套先行发生规则,以标识所述跟踪操作中的候选操作,其中每个先行发生规则指示在第二种操作之前发生的第一种操作;

构建所述候选操作的有序图,所述图的每个顶点表示所述候选操作中的一个,每条两个顶点之间的边表示所述两个顶点表示的操作之间的先行发生关系,向每个顶点分配位阵列,所述位阵列中的每一位表示所述图中的对应顶点;

对于每个顶点,遍历所述图,并在遍历所述图期间到达目标顶点时设置所述位阵列中对应于所述目标顶点的一个位;

标识一个或多个并发候选操作对,每个标识的并发候选操作对访问对应的一个共同对象,以生成潜在分布式并发错误列表,其中,将所述第一候选操作标识为与所述第二候选操作并发包括:确定所述第一候选操作的位阵列中没有设置对应于所述第二候选操作的位;以及

使用所述标识的一个或多个并发候选操作对,标识所述多个组件计算机中导致分布式并发错误的冲突存储器访问。

2. 根据权利要求1所述的方法,其特征在于,所述先行发生规则包括关于两个节点之间的消息的消息规则、关于不同节点发起的线程的线程规则、关于不同线程访问的事件的事件规则以及关于不同线程中操作的执行顺序的程序排序规则。

3. 根据权利要求1所述的方法,其特征在于,还包括:

将来自对应的不同线程的、访问所述对应的共同对象并包括至少一个写操作的对应候选操作对标识为所述候选操作。

4. 根据权利要求1所述的方法,其特征在于,跟踪所述访问对象的操作包括仅跟踪远程过程调用(remote procedure call, RPC) 函数、实施套接字操作的函数和事件处理程序函数。

5. 根据权利要求1所述的方法,其特征在于,还包括:分析用于生成所述潜在分布式并发错误列表的每个并发候选操作对,以从所述列表中删除不大可能导致严重故障的并发候选操作对。

6. 根据权利要求5所述的方法,其特征在于,对于每个并发候选操作对,所述并发候选操作对访问的所述对应共同对象位于第一节点中,所述方法还包括:分析所述分布式计算系统中出现所述并发候选操作对的一个或多个部分,以确定所述并发操作乱序执行导致的分布式并发错误在不同于所述第一节点的第二节节点中是否有影响。

7. 根据权利要求1所述的方法,其特征在于,还包括:修改所述分布式计算系统的线程,确定运行所述分布式计算系统期间每个并发操作对中的每个操作的相对时序,以标识真正的动态并发错误。

8. 根据权利要求1所述的方法,其特征在于,还包括:修改所述分布式计算系统的线程,以调整运行所述分布式计算系统期间所述并发操作对所选择的操作的相对时序,引起真正的分布式并发错误,从而确定所述选择的操作的时序灵敏度。

9. 一种非瞬时性计算机可读介质,包括指令,其特征在于,所述指令由处理器执行时,

使得所述处理器：

在运行分布式计算系统期间跟踪所述分布式计算系统中访问对象的操作，以生成跟踪结果；

对所述跟踪结果应用预定的一套先行发生规则，以标识所述跟踪的操作中的候选操作，其中每个先行发生规则指示在第二种操作之前发生的第一种操作；

构建所述候选操作的有序图，所述图的每个顶点表示所述候选操作中的一个，每条两个顶点之间的边表示所述两个顶点表示的操作之间的先行发生关系向每个顶点分配位阵列，所述位阵列中的每一位表示所述图中的对应顶点；

对于每个顶点，遍历所述图，并在遍历所述图期间到达目标顶点时设置所述位阵列中对应于所述目标顶点的一个所述位；

标识一个或多个并发候选操作对，每个标识的并发候选操作对访问对应的一个共同对象，以生成潜在分布式并发错误列表，其中，将所述第一候选操作标识为与所述第二候选操作并发包括：确定所述第一候选操作的位阵列中没有设置对应于所述第二候选操作的位；以及

调整所述并发候选操作对中对应于每个对应潜在分布式并发错误的选择的候选操作的发生顺序，以确认所述分布式并发错误。

10. 根据权利要求9所述的非瞬时性计算机可读介质，其特征在于还包括指令，使得所述处理器：将来自对应的不同线程的、访问所述对应的共同对象并包括至少一个写操作的对应候选操作对标识为所述候选操作。

11. 根据权利要求9所述的非瞬时性计算机可读介质，其特征在于，还包括指令，使得所述处理器：仅跟踪远程过程调用(remote procedure call, RPC) 函数、实施套接字操作的函数和事件处理程序函数。

12. 根据权利要求9所述的非瞬时性计算机可读介质，其特征在于，还包括指令，使得所述处理器：分析用于生成所述潜在分布式并发错误列表的所述并发候选操作对，以从所述列表中删除不大可能导致严重故障的并发候选操作对。

13. 根据权利要求12所述的非瞬时性计算机可读介质，其特征在于，还包括指令，使得所述处理器：响应于所述并发候选操作访问的所述对应共同对象位于第一节点中，分析所述分布式计算系统中出现所述并发候选操作对的一个或多个部分，以确定所述并发候选操作乱序执行导致的分布式并发错误在不同于所述第一节点的第二节点中是否有影响。

14. 一种处理分布式计算系统的组件以标识分布式并发错误的方法，其特征在于，所述方法包括：

通过向所述分布式计算系统中插入跟踪命令生成跟踪分布式计算系统，所述跟踪命令在RPC函数、实施套接字操作的函数和事件处理程序函数中跟踪对对象的访问；

运行所述跟踪分布式计算系统以收集跟踪数据；

分析所述跟踪数据以构建具有顶点和边的图，所述顶点对应于访问所述对象的操作，所述边对应于连接所述操作的先行发生规则；

分析所述图以标识可能导致所述分布式并发错误的候选操作对；

修改所述分布式计算系统以提供具有可调节时序的分布式计算系统；以及

调节时序时多次运行所述具有可调节时序的分布式计算系统，以标识哪些候选操作对

产生真正的分布式并发错误。

15. 根据权利要求14所述的方法,其特征在于,构建所述先行发生图包括构建有向无环图。

16. 根据权利要求14所述的方法,其特征在于,修改所述分布式计算系统包括使用静态字节码分析框架或动态字节码转换框架中的至少一个。

17. 一种在包括多个组件计算机的分布式计算系统中检测分布式并发错误的装置,其特征在于,包括:

处理单元,耦合至分布式计算系统,所述处理单元用于:

在运行所述分布式计算系统期间跟踪所述分布式计算系统中访问对象的操作,以生成跟踪结果;

对所述跟踪结果应用预定的一套先行发生规则,以标识所述跟踪的操作中的候选操作,其中每个先行发生规则指示在第二种操作之前发生的第一种操作;

构建所述候选操作的有序图,所述图的每个顶点表示所述候选操作中的一个,每条两个顶点之间的边表示所述两个顶点表示的操作之间的先行发生关系向每个顶点分配位阵列,所述位阵列中的每一位表示所述图中的对应顶点;

对于每个顶点,遍历所述图,并在遍历所述图期间到达目标顶点时设置所述位阵列中对应于所述目标顶点的一个所述位;

标识一个或多个并发候选操作对,每个标识的并发候选操作对访问对应的一个共同对象,以生成潜在分布式并发错误列表,其中,将所述第一候选操作标识为与所述第二候选操作并发包括:确定所述第一候选操作的位阵列中没有设置对应于所述第二候选操作的位;以及

调整所述并发候选操作对中对应于每个对应潜在分布式并发错误的选择的候选操作的发生顺序,以确认所述分布式并发错误。

18. 根据权利要求17所述的装置,其特征在于,所述处理单元还用于将来自对应的不同线程的、访问所述对应的共同对象并包括至少一个写操作的对应候选操作对标识为所述候选操作。

19. 根据权利要求17所述的装置,其特征在于,所述处理单元还用于仅跟踪远程过程调用(remote procedure call, RPC)函数、实施套接字操作的函数和事件处理程序函数。

20. 根据权利要求17所述的装置,其特征在于,所述处理单元还用于:

分析用于生成所述潜在分布式并发错误列表的所述并发候选操作对,以从所述列表中删除不大可能导致严重故障的并发候选操作对。

21. 根据权利要求20所述的装置,其特征在于,所述处理单元还用于:

响应于所述并发候选操作访问的所述对应共同对象位于第一节点中,分析所述分布式计算系统中出现所述并发候选操作对的一个或多个部分,以确定所述并发候选操作乱序执行导致的分布式并发错误在不同于所述第一节点的第二节节点中是否有影响。

22. 一种处理分布式计算系统的组件以标识分布式并发错误的装置,其特征在于,所述装置包括:

处理单元,用于:

向所述分布式计算系统中插入跟踪命令以生成跟踪分布式计算系统,所述跟踪命令在

RPC函数、实施套接字操作的函数和事件处理程序函数中跟踪对对象的访问；

运行所述跟踪分布式计算系统以收集跟踪数据；

分析所述跟踪数据以构建具有顶点和边的图，所述顶点对应于访问所述对象的操作，所述边对应于连接所述操作的先行发生规则；

分析所述图以标识可能导致所述分布式并发错误的候选操作对；

修改所述分布式计算系统以提供具有可调节时序的分布式计算系统；以及

在调节时序时多次运行所述具有可调节时序的分布式计算系统，以标识哪些候选操作对产生真正的分布式并发错误。

23. 根据权利要求22所述的装置，其特征在于，还包括：

用于修改所述分布式计算系统的静态字节码分析框架或动态字节码转换框架中的至少一个。

在云系统中自动检测分布式并发错误

[0001] 交叉申请

[0002] 本申请要求于2017年8月3日提交的、申请号为15/668,469、发明名称为“在云系统中自动检测分布式并发错误 (Automatically Detecting Distributed Concurrency Errors in Cloud Systems)”的美国非临时专利申请,该非临时专利申请又要求于2016年8月12日提交的、申请号为62/374,449、发明名称为“在云系统中自动检测分布式并发缺陷 (Automatically Detecting Distributed Concurrency Bugs in Cloud Systems)”的美国非临时专利申请,所述两个申请的内容通过引用如全文复制结合在本申请中。

技术领域

[0003] 本公开涉及在计算系统中检测运行错误,确切地说,涉及在跨多个计算系统分布的系统中检测并发错误。

背景技术

[0004] 很多大数据和云计算系统通过分布式云系统实现,所述分布式云系统具有跨多个服务器并行运行的多个程序线程。这些系统包括数据管理系统、多人游戏系统、劳动力合作系统(例如 Sharepoint[®]、Slack[®]和HipChat[®]合作软件)等。这些系统包括软件基础设施,例如横向扩展存储、计算架构、同步业务和集群管理业务。这些分布式云系统的可靠性非常重要。很遗憾,这些系统易受分布式并发错误(缺陷)的影响,分布式并发错误(缺陷)在本文中称为DCbug。由于分布式云系统的状态空间较大,可能难以检测DCbug,并且由于分布式计算和通信的时序,DCbug不确定地出现。

发明内容

[0005] 根据本公开的一个方面,提供一种在包括多个组件计算机的分布式计算系统中检测分布式并发错误的方法,所述方法包括:在运行所述分布式计算系统期间跟踪访问对象的操作,以生成跟踪结果;对所述跟踪结果应用一套先行发生规则,以标识所述跟踪的操作中的候选操作,其中每个先行发生规则指示在第二种操作之前发生的第一种操作;标识访问对应共同对象的对应并发候选操作对,以生成潜在分布式并发错误列表;以及执行运行时分析工具,标识所述多个组件计算机中导致分布式并发错误的冲突存储器访问。

[0006] 可选地,在前述方面中的任一个中,所述先行发生规则包括关于两个节点之间的消息的消息规则、关于不同节点发起的线程的线程规则、关于不同线程访问的事件的事件规则以及关于不同线程中操作的执行顺序的程序排序规则。

[0007] 可选地,在前述方面中的任一个中,所述方法还包括将来自对应的不同线程的、访问所述对应的共同对象并包括至少一个写操作的对应候选操作对标识为所述候选操作。

[0008] 可选地,在前述方面中的任一个中,跟踪所述访问对象的操作包括仅跟踪远程过程调用(remote procedure call, RPC)函数、实施套接字操作的函数和事件处理程序函数。

[0009] 可选地,在前述方面中的任一个中,所述方法还包括:构建所述候选操作的有序

图,所述图的每个顶点表示所述候选操作中的一个,每条两个顶点之间的边表示所述两个顶点表示的操作之间的先行发生关系;以及确定所述图不包括从第一候选操作到第二候选操作的路径后,将所述第一候选操作标识为与所述第二候选操作并发。

[0010] 可选地,在前述方面中的任一个中,所述方法还包括:向每个顶点分配位阵列,所述位阵列中的每一位表示所述图中的对应顶点;对于每个顶点,遍历所述图,并在遍历所述图期间到达目标顶点时设置所述位阵列中对应于所述目标顶点的一个所述位;以及对于所述第一候选操作的位阵列,当对应于所述第二候选操作的位没有设置时,确定所述第一和第二候选操作并发。

[0011] 可选地,在前述方面中的任一个中,所述方法还包括:分析用于生成所述潜在分布式并发错误列表的每个并发候选操作对,以从所述列表中删除不太可能导致严重故障的并发候选操作对。

[0012] 可选地,在前述方面中的任一个中,所述方法还包括:对于每个并发候选操作对,所述并发候选操作对访问的所述对应共同对象位于第一节点中,所述方法还包括:分析所述分布式计算系统中出现所述并发候选操作对的一个或多个部分,以确定所述并发操作乱序执行导致的分布式并发错误在不同于所述第一节点的第二节节点中是否有影响。

[0013] 可选地,在前述方面中的任一个中,所述方法还包括:修改所述分布式计算系统的线程,确定运行所述分布式计算系统期间每个并发操作对中的每个操作的相对时序,以标识真正的动态并发错误。

[0014] 可选地,在前述方面中的任一个中,所述方法还包括:修改所述分布式计算系统的线程,以调整运行所述分布式计算系统期间所述并发操作对所选择的操作的相对时序,引起真正的分布式并发错误,从而确定所述选择的操作的时序灵敏度。

[0015] 根据本公开的另一方面,提供一种计算机可读介质,包括指令,所述指令由处理器执行时,使得所述处理器:在运行分布式计算系统期间跟踪所述分布式计算系统中访问对象的操作,以生成跟踪结果;对所述跟踪结果应用一套先行发生规则,以标识所述跟踪的操作中的候选操作,其中每个先行发生规则指示在第二种操作之前发生的第一种操作;标识访问对应共同对象的对应并发候选操作对,以生成潜在分布式并发错误列表;以及执行运行分析工具,调整所述并发候选操作对中对应于每个对应潜在分布式并发错误的选择的候选操作的发生顺序,以确认所述分布式并发错误。

[0016] 可选地,在前述方面中的任一个中,所述计算机可读介质还包括指令,使得所述处理器:将来自对应的不同线程的、访问所述对应的共同对象并包括至少一个写操作的对应候选操作对标识为所述候选操作。

[0017] 可选地,在前述方面中的任一个中,所述计算机可读介质还包括指令,使得所述处理器:仅跟踪远程过程调用(remote procedure call, RPC) 函数、实施套接字操作的函数和事件处理程序函数。

[0018] 可选地,在前述方面中的任一个中,所述计算机可读介质还包括指令,使得所述处理器:构建所述候选操作的有序图,所述图的每个顶点表示所述候选操作中的一个,每条两个顶点之间的边表示所述两个顶点表示的操作之间的先行发生关系;以及响应于确定所述图中第一候选操作与第二候选操作不相连,将所述第一候选操作标识为与所述第二候选操作并发。

[0019] 可选地,在前述方面中的任一个中,所述计算机可读介质还包括指令,使得所述处理器:向每个顶点分配位阵列,所述位阵列中的每一位表示所述图中的对应顶点;对于每个顶点,遍历所述图,并在遍历所述图期间到达目标顶点时设置所述位阵列中对应于所述目标顶点的一个所述位;以及对于所述第一候选操作的位阵列,当对应于所述第二候选操作的位没有设置时,确定所述第一和第二候选操作并发。

[0020] 可选地,在前述方面中的任一个中,所述计算机可读介质还包括指令,使得所述处理器:分析用于生成所述潜在分布式并发错误列表的所述候选操作并发对,以从所述列表中删除不大可能造成严重故障的候选操作并发对。

[0021] 可选地,在前述方面中的任一个中,所述计算机可读介质还包括指令,使得所述处理器:响应于所述并发候选操作访问的所述对应共同对象位于第一节点中,分析所述分布式计算系统中出现所述并发候选操作对的一个或多个部分,以确定所述并发候选操作乱序执行导致的分布式并发错误在不同于所述第一节点的第二节节点中是否有影响。

[0022] 根据本公开的又一方面,提供一种处理分布式计算系统的组件以标识分布式并发错误的方法,所述方法包括:通过向所述分布式计算系统中插入跟踪命令生成跟踪分布式计算系统,所述跟踪命令在RPC函数、实施套接字操作的函数和事件处理程序函数中跟踪对对象的访问;运行所述跟踪分布式计算系统以收集跟踪数据;分析所述跟踪数据以构建具有顶点和边的图,所述顶点对应于访问所述对象的操作,所述边对应于连接所述操作的先行发生规则;分析所述图以标识可能导致所述分布式并发错误的候选操作对;修改所述分布式计算系统以提供具有可调节时序的分布式计算系统;以及调节时序时多次运行所述具有可调节时序的分布式计算系统,以标识哪些候选操作对产生真正的分布式并发错误。

[0023] 可选地,在前述方面中的任一个中,构建所述先行发生图包括构建有向无环图。

[0024] 可选地,在前述方面中的任一个中,修改所述分布式计算系统包括使用静态字节码分析框架或动态字节码转换框架中的至少一个。

[0025] 根据本公开的又一方面,提供一种装置,所述装置包括处理单元,耦合至分布式计算系统,所述处理单元用于:在运行所述分布式计算系统期间跟踪所述分布式计算系统中访问对象的操作,以生成跟踪结果;对所述跟踪结果应用一套先行发生规则,以标识所述跟踪的操作中的候选操作,其中每个先行发生规则指示在第二种操作之前发生的第一种操作;标识访问对应共同对象的对应并发候选操作对,以生成潜在分布式并发错误列表;以及调整所述并发候选操作对中对应于每个对应潜在分布式并发错误的选择的候选操作的发生顺序,以确认所述分布式并发错误。

[0026] 可选地,在前述方面中的任一个中,所述处理单元还用于将来自对应的不同线程的、访问所述对应的共同对象并包括至少一个写操作的对应候选操作对标识为所述候选操作。

[0027] 可选地,在前述方面中的任一个中,所述处理单元还用于仅跟踪远程过程调用(remote procedure call,RPC)函数、实施套接字操作的函数和事件处理程序函数。

[0028] 可选地,在前述方面中的任一个中,所述处理单元还用于:构建所述候选操作的有序图,所述图的每个顶点表示所述候选操作中的一个,每条两个顶点之间的边表示所述两个顶点表示的操作之间的先行发生关系;以及响应于确定所述图中第一候选操作与第二候选操作不相连,将所述第一候选操作标识为与所述第二候选操作并发。

[0029] 可选地,在前述方面中的任一个中,所述处理单元还用于:向每个顶点分配位阵列,所述位阵列中的每一位表示所述图中的对应顶点;对于每个顶点,遍历所述图,并在遍历所述图期间,在到达目标顶点时,设置所述位阵列中的所述位之一与所述目标顶点对应;以及对于所述第一候选操作的位阵列,当对应于所述第二候选操作的位没有设置时,确定所述第一和第二候选操作并发。

[0030] 可选地,在前述方面中的任一个中,所述处理单元还用于:分析用于生成所述潜在分布式并发错误列表的所述并发候选操作对,以从所述列表中删除不大可能导致严重故障的并发候选操作对。

[0031] 可选地,在前述方面中的任一个中,所述处理单元还用于:响应于所述并发候选操作访问的所述对应共同对象位于第一节点中,分析所述分布式计算系统中出现所述并发候选操作对的一个或多个部分,以确定所述并发候选操作乱序执行导致的分布式并发错误在不同于所述第一节点的第二节中是否有影响。

[0032] 根据本公开的又一方面,提供一种处理分布式计算系统的组件以标识分布式并发错误的装置,所述装置包括处理单元,用于:向所述分布式计算系统中插入跟踪命令以生成跟踪分布式计算系统,所述跟踪命令在RPC函数、实施套接字操作的函数和事件处理程序函数中跟踪对对象的访问;运行所述跟踪分布式计算系统以收集跟踪数据;分析所述跟踪数据以构建具有顶点和边的图,所述顶点对应于访问所述对象的操作,所述边对应于连接所述操作的先行发生规则;分析所述图以标识可能导致所述分布式并发错误的候选操作对;修改所述分布式计算系统以提供具有可调节时序的分布式计算系统;以及在调节时序时多次运行所述具有可调节时序的分布式计算系统,以标识哪些候选操作对产生真正的分布式并发错误。

[0033] 可选地,在前述方面中的任一个中,所述装置还包括:用于修改所述分布式计算系统的静态字节码分析框架或动态字节码转换框架中的至少一个。

[0034] 前述示例中的任何一个可与其它前述示例的任何一个或多个组合以产生在本公开的范围内的新实施例。

附图说明

[0035] 图1是简单的分布式云系统的框图。

[0036] 图2A和2B是可用于说明分布式并发错误的时序图。

[0037] 图3是另一分布式云系统的时序图。

[0038] 图4和图5是展示不同类别的并发错误的状态空间图。

[0039] 图6是根据各实施例的说明异步通信并发规则的状态空间图和时序图。

[0040] 图7是根据各实施例的说明RPC并发规则的状态空间图和时序图。

[0041] 图8是根据各实施例的说明进程间通信并发规则的状态空间图和时序图。

[0042] 图9A和9B是说明用于三个系统之间通信的并发规则的状态空间图和时序图。

[0043] 图10A和10B是根据各实施例的检测并发错误的示例系统的流程图。

[0044] 图11和12是根据各实施例的说明触发运行时并发错误的示例技术的时序图。

[0045] 图13是可用作所描述示例中的任一个的示例服务器的框图。

具体实施方式

[0046] 以下结合附图进行详细描述,所述附图是描述的一部分,并通过图解说明的方式示出可以实施本发明的具体实施例。足够详细地描述这些实施例以使所属领域的技术人员能够实践本发明,并且应理解,可利用其它实施例,并且可在不脱离本发明的范围的情况下作出结构、逻辑和电性变化。因此,以下描述的示例性实施例并不当作限定,本发明的范围由所附权利要求书界定。

[0047] 以下示例描述用于检测DCbug的DCatch系统。DCatch通过分析和监测分布式云系统的运行预测DCbug。DCatch系统使用一套“先行发生”规则,该些“先行发生”规则将现实分布式云系统中使用的多种通信和并发机制模型化。每个示例先行发生规则限制两个动作,使得一个动作发生在另一个之前。基于这套先行发生规则,示例DCatch系统构建运行时跟踪和跟踪分析工具,以高效地标识分布式云系统中的并发和冲突存储器访问。一旦标识这些存储器访问,DCatch系统采用静态和动态工具帮助删减误报并在测试时触发DCbug。

[0048] 由于分布式系统处理的主题很重要,系统的用户期望更高的可靠性,但是,很遗憾由于系统使用的进程间通信软件很复杂,很难保证可靠性。

[0049] 分布式系统中的所有错误类型中,称为DCbug的分布式并发错误是最麻烦的。这些错误是由节点之间不合时宜的互动触发的,并且这些错误可以传播,导致一个节点外的更多错误。先前研究表明DCbug广泛存在于现实分布式系统中,导致多种故障症状,例如数据损坏、系统崩溃和工作挂起。

[0050] 以下内容描述分布式数据管理系统背景下的DCatch系统。但是,在此考虑了DCatch可用于任何分布式计算系统中,包括但不限于:多人游戏系统、劳动力合作系统和提供万维网或基于云的业务的系统。此外,虽然下文所述示例示出服务器为独立实体,但是在此考虑了可以在例如虚拟机的单个机器中实施两个或更多个服务器的情况。

[0051] 图1是简单分布式云系统的框图,该分布式云系统采用三个服务器:网络管理器102、客户端104和应用管理器106。三个服务器通过网络108通信,网络108可以为局域网(local area network,LAN)、广域网(wide area network,WAN)或全局通信网络(例如因特网)。示例网络可以包括有线和无线组件。

[0052] 图2A和2B是可用于说明分布式并发错误的时序图。图2A示出来自 Hadoop® MapReduce™的分布式并发的现实示例。如图所示,客户端104上运行的线程向应用管理器106请求任务,如箭头202所示。然后应用管理器106向网络管理器102中的容器分配任务,如箭头204所示。然后网络管理器从应用管理器106检索任务,如箭头206所示。允许网络管理器访问任务后,客户端104取消任务,如箭头208所示。图2A中所示示例未呈现并发错误,因为网络管理器102上运行的线程在客户端104上运行的线程取消任务之前访问应用管理器106上运行的任务。

[0053] 图2B示出包括分布式并发错误(DCbug)的类似场景。在图2B所示示例中,由节点管理器102、应用管理器106和客户端104之间非预期的时序触发错误。应用管理器106向网络管理器102中的容器分配任务之后,网络管理器容器尝试从应用管理器106检索任务的内容。但是当检索请求传递到应用管理器106时,任务已经在收到来自客户端104的请求时取消。由于无法预测这种时序场景,网络管理器102容器挂起,如循环箭头212指示,并一直等待应用管理器106返回任务。

[0054] DCbug是不确定的,因此很难在跨多个节点的分布式系统的较大状态空间中找到DCbug。

[0055] 只有很少几套方法可以解决DCbug,包括软件模型检查、校验、可校验语言以及记录和重播调试。虽然这些技术很强大,但是也有固有局限性。分布式系统模型检查程序易受状态空间激增的影响,可能需要几小时甚至几天才能完成。校验方法对每个协议需要写数千条验证。还没有采用可校验的语言,因为低级命令式语言仍然由于性能原因非常普遍。记录和重播技术在软件故障之前无法帮助发现错误。此外,由于难以录入跨分布式系统的所有时序相关事件,这些技术对于调试DCbug效果有限。

[0056] 可以使用动态缺陷检测来检测本地并发(local concurrency,LC)。简单来说,动态缺陷检测技术监测并分析存储器访问和同步操作,以将冲突和并发存储器访问标识为本地并发错误(LCbug)怀疑对象。在这种意义上,“冲突”是指多个访问正在使用至少一个写访问接触同一存储单元位置。术语“并发”是指各访问之间不存在先行发生因果关系,导致各访问可以以任何顺序一个接一个发生。这些动态缺陷检测技术不能保证查找到所有缺陷,且经常发生误报。但是,LC技术可以通过开发人员有限的标注或代码变化应用于流行语言实施的现有的大型现实系统。

[0057] 通过理解DCbug指导下文所述的示例DCbug检测工具。DCbug与LCbug的根本原因基本上类似:访问一个机器内同一存储单元位置的并发冲突存储器访问之间的非预期时序。例如上文参考图2B所述,虽然DCbug是由于触发和多个节点之间的错误传播而出现的,但是根本问题还在于客户端104上运行的事件处理程序可以在远程过程调用(remote procedure call,RPC)函数读取同一输入时删除任务。开发人员不希望发生这一系列事件。

[0058] 示例DCbug检测工具将目标系统中的因果关系抽象化为若干先行发生(happens-before,HB)规则。多线程软件中这种HB规则的一个示例是线程创建比线程执行“先行发生”。遵循这些规则构建HB图,表示目标系统中所有存储器访问之间的时序关系;最后基于该HB图标识所有并发冲突存储器访问对。

[0059] DCbug和分布式系统在若干方面与LCbug和单机系统不同,这对DCbug检测造成若干挑战。

[0060] 首先,DCbug比LCbug有更复杂的时序关系。虽然DCbug的根本存储器访问位于同一机器中,但推断DCbug的时序关系还是很复杂,因为访问请求可能来自不同机器。在每个分布式系统内,通过使用一套不同的通信和同步机制,例如RPC、队列等,不仅在线程级执行并发存储器访问,还在节点级和事件级执行。在不同系统上,可以存在通信和同步机制的不同选项,未必总是标准化为多线程软件或Android事件中便携式操作系统接口(portable operating system interface,POSIX)线程库中的项目和/或事件驱动移动App中进程间通信(inter-process communication,IPC)库中的项目。因此,设计现实分布式系统的HB规则是非常重要的。错误或不完整的HB建模可能危及DCbug检测的准确性和覆盖度。

[0061] 检测DCbug的第二个挑战在于大规模的系统 and 错误。分布式系统通常比单机系统的规模大。分布式系统包含更多节点以及总体更多的动态存储器访问。DCbug运行规模也比LCbug大。例如,图2B所示的DCbug在触发和错误传播时涉及三个节点:客户端、应用管理器和网络管理器。较大的系统规模对在大量存储器访问中标识DCbug造成可扩展性挑战。较大的缺陷规模也受益于新技术,从而分析和标识DCbug。

[0062] 第三个挑战与容错性有关。分布式系统可以包括冗余,以容许组件故障。分布式系统的容错性设计有时处理中间错误有时放大错误,导致难以判断哪些错误是真正有害的。

[0063] 基于对以上机会和挑战的理解,下文描述示例DCbug检测工具——DCatch。DCatch的发展有两个阶段:第一,产生用于DCbug的HB模型;第二,设计DCatch的组件。

[0064] 第一步是构建DCatch运行的HB模型。这个模型基于对代表性开源分布式云系统的研究。示例HB模型包括一套覆盖节点间通信、节点内异步事件处理、节点内多线程计算和同步的HB规则。

[0065] 构建HB模型之后,下一步骤是构建DCbug检测工具DCatch,DCatch自定义为处理DCbug检测中的特有问题的。DCatch工具包括四个组件:运行时跟踪、离线跟踪分析、静态缺陷报告删减和DCbug测试和触发。

[0066] 运行时跟踪器组件在系统运行期间跟踪存储器访问、事件队列操作、节点间RPC、套接字通信和其它潜在冲突存储器访问。该组件侧重于节点间通信和计算相关的存储器访问,帮助系统处理DCbug检测中的大规模问题,并允许将DCatch的规模扩大至较大的现实分布式云系统。

[0067] 离线跟踪分析组件分析运行时跟踪,按照HB模型构建所有记录的存储器访问的HB图,并上报所有并发冲突访问(即DCbug候选项)对。这一阶段的关键成果在于构建分布式系统的HB图。

[0068] 静态删减模块分析程序以确定DCbug候选项的本地影响和分布式影响。该组件帮助确定特定DCbug候选项是否有害,从而避免过多误报。

[0069] DCatch缺陷触发模块考虑分布式系统中的不同并发和通信机制的同时运行系统的修改版本,该系统版本根据缺陷报告监测和/或操控分布式执行的时序。该模块帮助触发真正的缺陷,进一步减少误报。

[0070] 如上文所述,DCatch先行发生(Happens-Before, HB)模型基于对多个分布式云数据处理系统的分析。HB模型的目标是抽象化可以应用于各种分布式云系统的一套先行发生规则。每个规则R表示这些系统中一对操作之间的一种因果关系o,规则表示为 $o1 \xrightarrow{R} o2$ 。这些规则基于任意两个操作o1和o2之间的时序关系。具体地,当已知o1必须在o2之前发生时,表示为 $o1 \Rightarrow o2$,一套HB规则可以标识为将o1和o2链接在一起(例如, $o1 \xrightarrow{R^1} o1 \xrightarrow{R^2} o2 \dots o_{k-1} \xrightarrow{R^k} o2$)。如果既不是 $o1 \Rightarrow o2$,也不是 $o2 \Rightarrow o1$,则o1和o2是并发的且因此可以以任何顺序并列执行。这套HB规则足够全面和准确,以允许DCatch容纳例如图3示出的分布式系统中的复杂时序关系,并实现良好的缺陷检测准确性和覆盖度。

[0071] 图3是包括多个节点上运行的多个线程的分布式云系统的时序图。在节点1即301中,线程306上运行的方法对系统变量执行写操作304。在308中,该方法创建新线程310,新线程310执行远程过程调用以启用节点2即302上运行的线程318中的方法314。在框316中,方法314向线程324上运行的事件处理程序320添加事件。协调器将事件322从线程324中移走,并向节点1的线程332中运行的方法328推送通知。方法328对操作304写的变量执行读操作330。由于不同线程和通信模式的数量很多,很难确定在操作304之前执行操作330是否会引起分布式并发错误(DCbug)。

[0072] 以下示例是从涵盖代表性现实云系统的并发和通信机制中得到的,HB规则是从该

云系统中提取的。

[0073] 如上文所述,HB并发规则可以包括用于本地并发错误(LCbug)和分布式并发错误(DCbug)的规则。图4和图5是展示不同类别的并发错误的状态空间图。图4示出分区立方体,其中阴影块402表示被同步的标准操作导致的DCbug,块404表示异步自定义操作导致的DCbug,块406表示异步标准操作导致的DCbug,块408表示同步标准操作导致的DCbug。对于本地并发错误(LCbug),块412表示被同步的标准操作导致的LCbug,块414表示异步自定义操作导致的LCbug,块410表示同步标准操作导致的LCbug。另一个块(图4中未示出)表示异步标准操作导致的LCbug。图5中进一步说明并发错误分区,图5使用与图4相同的编号,以及示出表示异步标准操作导致的LCbug的块502。示例DCatch工具侧重于图5所示的阴影块402、404、406和408(即分布式、同步和异步、自定义和标准操作)。

[0074] 如图6和7所示,每个分布式系统包括多个通过消息互相通信的并行执行的节点。分析进程间通讯会基于不同通信模式产生多个消息相关的HB规则,本文称为Rule-M。

[0075] 图6是说明异步通信并发规则的状态空间图和时序图。图6示出远程过程调用(remote procedure call, RPC),RPC是阴影块408所示的被同步的标准操作。机器A上运行的线程602调用机器B上运行的线程608中实施的RPC函数r(606)。线程602等待(612),直到线程608返回RPC执行结果610。

[0076] 该通信模式指示如下HB规则。在节点1上发起RPC调用r表示为Create(r, n₁),在节点2上开始RPC函数执行表示为Begin(r, n₂),Create(r, n₁)发生在Begin(r, n₂)之前。此外,节点2上RPC函数执行的结束表示为End(r, n₂),从RPC调用节点1中的RPC调用r返回表示为Join(r, n₁),End(r, n₂)发生在Join(r, n₁)之前。

[0077] Rule-M^{RPC}: Create(r, n₁) $\xRightarrow{M^{RPC}}$ Begin(r, n₂); End(r, n₂) $\xRightarrow{M^{RPC}}$ Join(r, n₁)

[0078] 图7是说明RPC并发规则的状态空间图和时序图。图7示出块406所表示的异步套接字通信。节点1(704)中的线程702通过网络套接字向节点2(708)中的线程706发送消息m。不同于RPC,发送方并不阻拦自己,而是可以根据侦听用于传入消息的套接字选择阻拦自己。很明显,发送消息m发生在接收m之前,得到规则M^{SOC}。

[0079] Rule-M^{SOC}: Send(m, n₁) $\xRightarrow{M^{SOC}}$ Recv(m, n₂)

[0080] 除了以上两种基础通信机制之外,以下示例处理节点之间的另外两种高级同步协议。这些通信类型中的每一个都是使用RPC/套接字通信和节点内计算的组合实施的。因此,每个通信类型分配有自己的HB规则。

[0081] 第一种通信是基于推送的通知协议,是图9A中阴影块所示的自定义异步通信。图9B中示出该通信,下文将详细描述。对于基于推送的通知,节点n₁(902)中的线程924利用写操作w将对象s更新到专用协调节点n_c(926)中的线程932,n_c将该更新通知到所有相关节点,例如n₂(901)。n₁更新s表示为Update(s, n₁),更新通知传送到n₂(901)表示为Pushed(s, n₂),很明显,Update(s, n₁)发生在Pushed(s, n₂)之前。例如, HBase[®]节点有时通过ZooKeeper[™]通信。ZooKeeper是维护配置信息,即提供分布式同步和提供分组业务的集中服务。类似于ZooKeeper的协调系统中,其它程序向系统注册密钥,并通知这些程序与密钥相关联的任何变化。HBase是可与Hadoop分布式文件系统(Hadoop distributed file system, HDFS)一起使用的非关系分布式数据库。HBase提供用于存储大量稀疏数据的机制。

[0082] 第一种通信中,节点通过ZooKeeper上某路径向注意到zknode,然后ZooKeeper通知该节点所有从其它节点到zknode的变化。

[0083] $\text{Rule-M}^{\text{push}}: \text{Update}(s, n_1) \xrightarrow{M^{\text{push}}} \text{Pushed}(s, n_2)$

[0084] 应注意,考虑到 $\text{Rule-M}^{\text{rpc}}$ 和 $\text{Rule-M}^{\text{soc}}$,这个规则并不多余。 $\text{Rule-M}^{\text{push}}$ 可以分解成三个因果关系链。

[0085] (1) $\text{Update}(s, n_1) \Rightarrow \text{Recv}(s, n_c);$

[0086] (2) $\text{Recv}(s, n_c) \Rightarrow \text{Send}(s, n_c);$

[0087] (3) $\text{Send}(s, n_c) \Rightarrow \text{Pushed}(s, n_2)$

[0088] 其中 n_c 是包括ZooKeeper协调器的节点926。

[0089] 链(2)可能很难计算,因为它涉及 n_c 中保证对 s 感兴趣的每个节点收到通知的复杂的节点内计算和同步。甚至对于链(1)和(3),也无法保证 $\text{Rule-M}^{\text{rpc}}$ 和 $\text{Rule-M}^{\text{soc}}$ 能计算出链(1)和(3),因为节点 n_1 (902)、 n_2 (901)和 n_c (926)之间的通信通常含有一个以上RPC/套接字消息。

[0090] 第二种通知是基于拉取的通知协议。图8是说明进程间通信并发生规则的状态空间图和时序图。节点 n_2 (808)中的线程806一直轮询节点 n_1 (802)关于 n_1 中的状态对象 s (例如变量Flag的状态)。节点 n_2 不进行下一步直到获知 n_1 已经将 s 更新为具体值。这一HB规则可以抽象为定义 n_1 中的状态 s 发生在在 n_2 上(例如在线程810中)使用该状态之前。这种同步在Hadoop MapReduce和HBase中出现。

[0091] $\text{Rule-M}^{\text{pull}}: \text{Update}(s, n_1) \xrightarrow{M^{\text{pull}}} \text{Pulled}(s, n_2)$

[0092] 同样,考虑到其它HB规则,这个规则并不多余,因为 n_1 中节点内语义很复杂。传统的HB规则无法建立设置 s 和另一线程中的RPC函数读取 s 之间的因果关系,或设置 s 和将 s 串行化为套接字消息之间的因果关系。这个规则类似于单机系统中while循环自定义同步的分布式版本。

[0093] 除了通讯规则,定义节点内并发和通信规则也是有用的。每个节点内可以存在多个线程。图9A和9B是说明用于三个系统之间通信的并发生规则的状态空间图和时序图。如图9B所示,这些线程中的一些(例如918)专用于运行RPC函数实施,一些(例如924)是事件处理线程,一些(例如906和932)是常规线程,例如单机多线程软件中的线程。这些DCbug是由阴影块404所示的异步自定义操作产生的。

[0094] 在父线程中创建线程 t 表示为 $\text{Create}(t)$,开始运行 t 表示为 $\text{Begin}(t)$, $\text{Create}(t)$ 发生在 $\text{Begin}(t)$ 之前。结束线程 t 表示为 $\text{End}(t)$,在另一个线程中加入 t 表示为 $\text{Join}(t)$, $\text{End}(t)$ 发生在 $\text{Join}(t)$ 之前。

[0095] $\text{Rule-T}^{\text{fork}}: \text{Create}(t) \xrightarrow{T^{\text{fork}}} \text{Begin}(t)$

[0096] $\text{Rule-T}^{\text{join}}: \text{End}(t) \xrightarrow{T^{\text{join}}} \text{Join}(t)$

[0097] 理论上,存在另一个有效的线程相关的HB规则:condition-variable-notify发生在对应condition-variable-wait结束之前。但是Condition-variable-notify和condition-variable-wait几乎从不用于与节点间通信和计算相关的代码区。如上所述,示

例系统通过分析节点间通信和计算检测DCbug。

[0098] 最后,如下文所述,虽然示例DCatch系统跟踪锁定/解锁操作,DCatch不处理锁定同步,因为锁定是用于提供相互排除,而非严格排序。但是,如下文所述,DCatch可以使用锁定/解锁操作触发某些DCbug候选项。了解锁定/解锁操作可以有益于在DCatch尝试操控时序和触发DCbug候选项时避免挂起。因此,DCatch跟踪锁定和解锁操作,包括隐式锁定操作(即同步方法和同步语句)和显式锁定操作。

[0099] 很多分布式系统实施异步事件驱动处理,本质上是在线程内创建并发。可以通过任何线程将事件放在队列中。调度器线程通常负责从队列中取出事件,并将事件分配给事件处理thread(s),在事件处理thread(s)中执行预定义事件处理程序。将事件e入队表示为Create(e),开始e的处理程序函数表示为Begin(e),很明显,Create(e)发生在Begin(e)之前。

[0100] Rule-E^{enq}: Create(e) $\xRightarrow{E^{enq}}$ Begin(e)

[0101] 对于同一队列的两个事件e₁和e₂,处理e₁和e₂之间的时序可以取决于队列的若干属性:队列是否为FIFO队列?有多少个调度线程?有多少个处理线程?很多系统中所有队列都是FIFO队列,且每个队列只有一个调度线程。因此,当含有e₁和e₂的队列只有一个处理线程时,将e₁和e₂的处理串行化,否则e₁和e₂的处理是并发的。前一种队列称为单用户队列。Zookeeper中的所有队列和MapReduce中的部分队列是单用户队列。处理这些队列的事件遵循以下HB规则。

[0102] Rule-E^{serial}: End(e₁) $\xRightarrow{E^{serial}}$ Begin(e₂), if Create(e₁) $\Rightarrow$ Create(e₂)

[0103] 其中e₁ $\in$ Q, e₂ $\in$ Q, 且Q是单用户FIFO队列。

[0104] 所述示例还采用用于有序的程序排序规则。根据传统的先行发生模型,在执行一个线程时,如果操作o₁在操作o₂之前出现,则o₁在o₂之前发生。即,一个线程内的执行顺序是确定的。

[0105] Rule-P^{reg}: o₁ $\xRightarrow{p^{reg}}$ o₂

[0106] 执行常规线程时,如果o₁在o₂之前出现。

[0107] 该先行发生规则只适用于不包含任何线程内并发的线程。在分布式系统中,该规则不适用于事件处理线程和RPC线程。在这两种线程中,以上规则只适用于o₁和o₂在同一事件处理程序函数或RPC函数内时。

[0108] Rule-P^{nreg}: o₁ $\xRightarrow{p^{nreg}}$ o₂

[0109] 执行一个事件处理程序或一个RPC函数时,如果o₁在o₂之前出现。

[0110] 以上消息、任务、事件和程序(message, task, event and program, MTEP)规则构成DCatch HB模型。通过形成不同层级的并发,DCatch HB模型实现了对现实分布式云系统中两个操作之间的时序关系的准确模型化。

[0111] 例如,对于图9B中所示的真实示例,根据以下先行发生关系链,可以推断写操作w(904)出现在读操作r(930)之前(即w $\Rightarrow$ r)。

[0112] w $\xRightarrow{p^{reg}}$ Create(t) $\xRightarrow{T^{fork}}$ Begin(t) $\xRightarrow{p^{reg}}$ Create(OpenRegion, HMaster) $\xRightarrow{M^{rpc}}$ Begin(OpenRegion,

[0113]
$$\text{HRS} \xRightarrow{pnreg} \text{Create}(e) \xRightarrow{E^{enq}} \text{Begin}(e) \xRightarrow{pnreg} \text{Update}(\text{RS} \dots \text{OPENED}, \text{HRS}) \xRightarrow{M^{push}}$$

[0114]
$$\text{Pushed}(\text{RS} \dots \text{OPENED}, \text{HMaster}) \xRightarrow{pnreg} r$$

[0115] 参考图9B,该HB关系转换为写操作w(线程906中的904)出现在创建线程910的创建操作908之前。在912中,线程910中的RPC调用发起线程918中的程序OpenRegion 914。在916中,OpenRegion 914将事件e放置到线程924的事件处理程序920中的事件队列922中。协调器926更新事件e,然后将事件e推送到执行读操作r(930)的RS...Opened 928。如虚线箭头923所示,协调器926的操作向线程932中的方法928通知事件处理程序920中的事件922。

[0116] 应注意,示例DCatch系统有意忽略了分布式计算系统中不影响检测DCbug的整体目标的某些因果关系。例如,实际上在将RPC调用分配给RPC线程之前,首先将输入RPC调用放在队列中。但是,Rule-R^{rpc}将属于RPC库实施的这些队列抽象屏蔽。另外,事件入队和开始事件处理之间存在事件调度进程。该事件调度进程在我们的Rule-E^{enq}也抽象屏蔽。此外,如上文所述,示例模型不考虑condition-variable notify-and-wait因果关系,因为该因果关系几乎从不用于分布式系统的节点间通信和计算部分。

[0117] 以下内容描述基于上文所定义的模型和规则的DCatch的四个组件的示例。四个组件包括跟踪、应用HB规则、分类以标识重要潜在DCbug和触发重要DCbug。参考图10A和10B描述示例DCatch系统。图10A是检测并发错误的示例系统的流程图,其示出基础操作的顺序。在框1002中,在执行分布式系统期间DCatch系统跟踪对对象的访问并生成跟踪结果。在框1004中,将先行发生规则应用于跟踪结果,以标识候选操作。在框1006中,标识候选操作并发对,每个对访问对应的共同对象并具有至少一个写操作。这些候选对表示潜在DCbug。在框1008中,DCatch系统使用运行时分析工具以标识并确定哪些潜在DCbug是真正的DCbug。

[0118] 在跟踪组件1002中,DCatch向分布式计算系统插入命令,从而在运行时间生成用于目标分布式系统的每个相关线程的跟踪文件。如下所述,该文件中的跟踪结果允许跟踪分析器1004应用HB规则并标识重要潜在DCbug。在一个示例系统中,使用WALATM、静态Java[®]字节码分析框架和/或JavassistTM、动态Java字节码转换框架实施跟踪组件。考虑了可以使用其它字节码分析软件例如SootTM Java字节码分析框架代替WALA。类似地,可以使用其它动态Java字节码转换框架例如ASMTM框架代替Javaassist。下文参考图10B描述该示例实施方式的细节。

[0119] 第一示例跟踪组件1052确定跟踪哪个操作。在一个示例中,DCatch收集DCbug两个基础组件的信息:存储器访问和HB相关操作。如下所述,示例DCatch还跟踪锁定/解锁操作。

[0120] 可以通过记录(例如录入)所有对程序变量的访问没有经验地执行存储器访问跟踪,程序变量可能在线程或事件处理程序之间共享。但是,该穷尽性的方式可能导致极大的录入和跟踪分析成本。幸而,因为并非全部软件都需要分析,这种过多录入对于DCbug检测是不必要的。确切地说,DCbug由节点间互动触发,其中根本存储器访问主要在与节点间通信和计算相关的代码区。

[0121] 按照这种设计原则,DCatch跟踪组件1052在以下三种函数及其被调用方函数中跟踪所有对堆对象和静态变量的访问:(1)RPC函数;(2)实施套接字操作的函数;(3)事件处理程序函数。前两种函数直接与节点间通信和对应计算相关。因为RPC调用的多个预处理或后处理和套接字发送和接收操作是通过事件队列和事件处理程序执行的,所以考虑第三种函

数。

[0122] 一旦跟踪操作跟踪这些操作,在框1054中,DCatch分析跟踪结果并应用HB规则。按照上述MTEP先行发生规则,示例DCatch系统跟踪允许跟踪分析器推断先行发生关系的操作,如表1所示。

[0123] 表1

操作	T-Rule	E-Rule	M-Rule	P-Rule
Create (t), Join(t)	✓			
Begin (t), End (t)	✓			
Begin (e)		✓		✓
End (e)				✓
Create (e)		✓		
Begin (r,n) End (r,n)			✓	
Create (r,n) Join (r,n)			✓	
Send (m,n ₁) Recv (m,n ₂)			✓	
Update (s,n ₁) Pushed (s,n ₂)			✓	

[0125] 示例DCatch系统可以在运行时间使用Javassist基础设施基于对应的库接口标识这些操作。下文更详细地描述该示例系统的示例实施方式。

[0126] 每个跟踪记录含有三条信息:(1)记录操作的类型;(2)记录操作的调用栈;(3)标识(identifier,ID)。前两条信息是直接的。但是,ID对于不同类型的记录有不同含义。在示例DCatch系统中,ID帮助DCatch跟踪分析器寻找相关跟踪记录。

[0127] 对于存储器访问,ID可以唯一标识该存储器访问接触的变量或对象。在一个示例系统中,对象字段的ID表示为对象内的段偏移和对象哈希码。静态变量的ID表示为变量名称及其对应的名字空间。

[0128] 对于锁定/解锁操作,ID唯一标识锁定对象,允许DCatch的触发模块标识所有锁定重要部分并在合适位置干扰时序。

[0129] 对于HB相关操作,ID允许DCatch跟踪分析,以正确应用HB规则。对于每个线程相关和事件相关操作,ID可以分别是对应的线程对象和事件对象的对象哈希码。每个RPC相关和套接字相关操作可以具有每个RPC调用实例和每个套接字消息的唯一ID。可以通过在运行时使用随机数标记每个RPC调用和每个套接字消息生成这些RPC和套接字相关ID。下文更详细地描述该示例系统的示例实施方式。

[0130] 示例DCatch跟踪分析器标识含有至少一个写操作、接触同一变量或对象并同时出现的每个堆/静态变量访问对。在一个实施方式中,未由HB关系直接或间接连接的操作都视为是并发的。如下所述,可以使用目标分布式系统的先行发生图确定并发。将标识的访问对视为DCbug候选项。

[0131] 示例DCatch跟踪分析包括两个步骤:框1056:构建先行发生图和框1058:标识DCbug候选项。

[0132] 先行发生图是有向无环图(directed acyclic graph,DAG)。在该图中,每个顶点 v 表示示例DCatch跟踪中记录的操作 $o(v)$,包括存储器访问和HB相关操作。该图中的边布置

成当且仅当 $o(v_1)$ 在 $o(v_2)$ 之前发生时 v_1 可以到达 v_2 。

[0133] 为构建这种图,在框1056中,示例DCatch系统首先分析从所有节点中的所有跟踪的进程的所有跟踪的线程收集的所有跟踪文件,并在图中将每条记录记为一个顶点。因为跟踪的函数只有RPC函数、实施套接字操作的函数、事件处理程序函数及其被调用函数,所以减少了待分析的数据量。

[0134] 然后,DCatch根据上述表1中所示的MTEP先行发生规则添加边。以下内容只描述 E^{serial} 和 M^{pull} 规则的应用。其它规则的应用大部分是直接的,因为每条跟踪记录内的ID允许跟踪分析轻易地将相关操作分组在一起。

[0135] 为了应用单用户事件队列规则(E^{serial}),DCatch HB图构建器1056等待,直到已经应用所有其它HB规则,这是应用MTEP HB规则中唯一的排序要求。对于每个处理单用户事件队列的线程,DCatch图构建器1056检查其跟踪结果中记录的每个End(e_i)操作和Begin(e_j)操作对,在基于那些已经添加的HB边发现Create(e_i)在Create(e_j)之前发生后,DCatch图构建器1056添加从End(e_i)操作到Begin(e_j)操作的边。DCatch重复该步骤直到到达固定点。

[0136] 使用程序分析应用规则 M^{pull} 。这里的算法类似于在LCbug检测中如何处理基于循环的自定义同步。对于每个冲突并发读和写 $\{r, w\}$ 操作对,如果(1) r 在RPC函数内执行,(2) 该RPC函数的返回值取决于 r , (3) 在请求该RPC的另一节点中该RPC的返回值是循环 l 的结束条件的一部分,则将 r 视为基于拉取的同步协议。然后再次运行目标软件,只跟踪这种读操作(rs)和基于原始跟踪接触同一对象的所有写操作(ws)。新的跟踪指示哪个写操作 w^* 在循环 l 结束之前为最后一个实例读操作 r 提供值。如果 w^* 和 r 操作来自不同线程,则一个节点中的 w^* 操作发生在另一节点中的远程循环 l 结束之前。这一部分分析与节点内while循环同步分析一起进行。虽然算法是第二次运行软件,但是只造成少量跟踪或跟踪分析开销,因为该算法关注与循环相关的存储器访问。

[0137] 构建先行发生图之后,DCatch时戳和并发框1058可以计算图中每个顶点的向量,并检查每个存储器访问顶点对,以标识访问同一存储器对象的冲突并发访问。即使减少跟踪,该方法仍然比较复杂,因为顶点的数量可能非常多,且每个向量时戳可能具有大量维度,每个维度对应一个事件处理程序和/或RPC函数。

[0138] 为了加速该分析,DCatch使用用于非分布式系统的异步竞争检测算法。简言之,算法首先为跟踪文件中呈现的每个存储器对象构建列表,该列表含有对对象的所有访问。然后,算法列举每个列表中的访问对,其中访问对中至少一个操作是写操作。对于每个这种对,框1058查询先行发生图以查看访问对中的操作是否并发。基础思路是计算先行发生图中每个顶点的可达集合。然后,查询一个顶点的可达集合,以查看其它顶点是否出现在产生的集合中。为了节约存储空间,可以为每个顶点 i 分配位阵列来表示可达集合,如果顶点 i 可以到达顶点 j 则设置第 j 个位。然后算法可以从每个顶点 i 遍历图,为遍历期间碰到的每个顶点 j 设置一个位。构建这些阵列之后,查询可以在固定时间内获得结果。换言之,无需向图中增加时戳就可以确定第一和第二操作之间的并发性。当第一操作对应的顶点的位阵列中没有设置表示第二操作的顶点的位时,算法将第一和第二操作标识为并发。

[0139] 框1058将并发冲突访问对上报为DCbug候选项。候选访问对如果访问同一对象并且其中至少一个访问是写操作,则该候选访问对是冲突的,以及如果顶点位阵列指示两个访问之间不存在先行发生关系,则该候选访问对是冲突的。以下内容将 s 和 t 或 (s, t) 指示为

跟踪分析中标识的并发冲突操作(访问)。但是,并非所有候选项都会导致执行故障。这对于本身比单机系统含有更多冗余和容错性的分布式系统中尤其正确。

[0140] 为了避免过多误报,假设缺陷候选项(s,t),在框1060中,DCatch系统静态地分析目标系统的相关Java字节码,以估计该缺陷候选项的潜在本地(即节点内)影响和分布式(即节点外)影响,并删减不大可能造成严重故障的候选项。

[0141] 示例DCatch系统的DCatch删减框1060进行程序间和节点间影响分析,以更好地适应分布式系统中DCbug的故障传播性质。框1060包括数据结构,该数据结构对故障分类,以标识视为严重故障的故障。数据结构还标识哪种指令视为故障指令。框1060可以检查任何故障指令的执行是否取决于缺陷候选项(s,t)。

[0142] 严重故障的定义可以不同。在一个示例DCatch系统中分析以下几种故障和故障指令:(1)系统中止和结束,其对应的故障指令是调用中止和退出函数(例如System.exit和System.abort);(2)打印出来的或以其它方式输出的严重错误,其对应的故障指令是调用所研究系统中的Log::fatal和Log::error函数;(3)(使用Java Throw语句)抛出无法捕捉到异常,例如RuntimeException;(4)无限循环,其中每个loop-exit指令视为潜在故障指令。最后,如果任何标识的故障指令是在捕捉框内,且如果对应的异常抛出指令可用,则框1060将其视为故障指令。

[0143] 以上所列是可配置的,允许配置示例DCatch删减框1060以检测具有不同类型的影响的DCbug。

[0144] 为了确定标识的DCbug是否是严重故障,DCatch删减框1060对每条缺陷报告(s,t)的程序字节码进行DCatch分析,以查看s或t是否可能对任何故障指令的出现有本地(即节点内)影响或分布式(节点间)影响。

[0145] 删减框1060同时进行用于本地影响分析的程序内和程序间分析。假设方法M中有存储器访问语句s,框1060首先检查M中的任何故障指令是否对s有控制或数据依赖性。框1060对t应用类似检查。如果框1060发现s或t的这种依赖性故障指令,DCatch将对应的缺陷候选项保留在缺陷报告列表中。

[0146] 然后框1060通过M的返回值或M访问的堆/全局对象检查s是否可以影响M的调用程序内的故障指令。应注意,根据DCatch跟踪器和跟踪分析报告调用栈信息,框1060中执行的程序间分析可以遵循s的上报调用栈。

[0147] 为了通过返回值确定影响,框1060检查M的返回值是否对s有控制或数据依赖性。如果有,框1060继续检查调用M的函数中的任何故障指令是否依赖于M的返回值。框1060根据s的调用栈沿着调用链进行类似分析。

[0148] 通过堆/全局变量检查影响可能更为复杂。框1060首先检查方法M中是否存在任何堆写入w对s有数据依赖性 or 控制依赖性。对于向对象o写入的每个这种w,DCatch检查M的调用程序,表示为M',以查看是否存在对o的读取r满足所有以下条:(1)读取r沿着从M的调用站点到故障指令的路径;(2)该故障指令对读取r有控制依赖性 or 数据依赖性。考虑到复杂性和不准确问题(由于混叠等),DCatch只对M的一层调用程序应用该分析,不再往上对呼叫链应用该分析。

[0149] 最后,框1060通过函数调用参数或堆/全局变量检查s是否会影响M的被调用函数(也称为“被调用方被调用方函数”)中的故障站点。该分析只应用于M的一层被调用方函数。

[0150] 除了节点内分析之外,框1060还执行节点间分析。如图2B所示,一个节点内的访问可能导致不同节点内的故障。因此,DCatch还分析RPC函数,以理解对存储器访问的远程影响。

[0151] 具体地,框1060沿着存储器访问s的调用栈发现RPC函数R之后,检查R的返回值是否取决于s。如果是,然后框1060在调用RPC调用R.的不同节点上定位函数 M_r 。在 M_r 中,框1060还检查是否有任何故障指令取决于R的返回值。应注意,考虑到DCatch运行时跟踪,定位 M_r 是简单直接的。

[0152] 理论上,框1060还可以通过套接字分析节点间影响。但是,套接字通信可能不像RPC调用一样结构化,因此没有开发人员标注更难以标识对应的细粒度依赖性信息。

[0153] 最后,对于DCbug候选项(s,t),如果框1060未能找到s和t的任何故障影响,框1060从DCatch缺陷列表中删减该DCbug候选项。在一个示例系统中,在WALA代码分析框架中利用构建程序依赖性图的WALA API完成以上实施方式。

[0154] 到目前为止上报的DCbug候选项因为以下两个原因仍然可能不是真正有害的。第一,一些上报的访问对可能互相不是真正并发,可以通过DCatch未标识的自定义同步固定其执行顺序。第二,一些真正的并发冲突访问对可能是良性的,以不同顺序执行这两个访问可能不会导致任何故障。应注意,上述故障影响分析只是静态估计,因此可能是错误的。此外,即使是那些真正有害的DCbug候选项,在分布式系统中触发这些候选项可能是非常困难的。

[0155] 为帮助删减误报及可靠地显示真正有害的DCbug,DCatch的最后一个组件——测试和触发框1062和1064支持测试分布式系统和触发DCbug。包括两部分:(1)在分布式系统中启用简单时序操控的基础设施;(2)建议如何使用基础设施触发DCbug候选项的分析工具。

[0156] 如图11所示,DCatch系统可以通过向程序中插入睡眠间隔干扰执行时序。图11是触发运行时并发错误的示例技术的时序图。如图11所示,在节点1中的RPC调用1104和节点2中的事件1108的入队操作1106中的一个或两个之前选择性地引入睡眠状态1102。如果节点2中1108和1112的执行顺序可以调换且如果存在错误,则插入的睡眠状态中的每一个都足够长,使得可以调换顺序并触发错误。如果事件1108和1112之间的顺序无法调换,或者如果可以调换但未检测到错误,则可以从DCbug候选项列表中删减这两个操作1108和1112。但是该方法可能不是在复杂系统中检测复杂缺陷的有效方法,因为很难获知睡眠间隔需要多长。有一种更复杂的方法可以运行一个处理器内核中的全部程序并通过线程调度器控制时序。但是,这些方法都对DCbug效果不好,因为DCbug需要操控来自不同节点的操作之间的时序。在一个处理器内核上运行现实大型分布式系统不太现实。

[0157] 一个示例DCatch基础设施包括两个组件:发送协调请求消息的客户端API和消息控制器服务器。下文中,测试的分布式系统称为客户端。

[0158] 考虑分析并发操作对A和B,以上参考图10B描述的测试和触发框1062和1064可以探索在B之前执行A或在A之前执行B。探索的一种方式是框1062将_request API调用放置在A之前,将_confirm API调用放置在A之后,并在B之前和之后放置类似的指令。运行时,_request API可以向控制器服务器发送消息,请求许可继续执行。在框1064中控制器等待来自双方的请求消息到达,然后向一方授予许可,等待对应的_confirm API发送的确认消息,

然后向另一方授予许可。控制器可以记录检查了何种排序,并重启系统若干次(框1066),直到检查完所有请求方(在此示例中仅有两方)之间的所有排序组合。控制器可以记录已经尝试的组合并重启系统若干次直到测试完所有组合。然后该系统在框1068处终止。图12中示出一个这种示例。

[0159] 图12是触发运行时并发错误的示例技术的时序图。在此示例中,_request API框1202插入到RPC调用框1204和1214之前,_confirm API框1204插入到RPC调用框1205和1214之后。控制器1201可以启用RPC调用,从而以任何顺序执行操作1208和操作1212,以确定操作1208和操作1212是否导致错误。其它地或另外地,_request框1202可以插入到入队操作1206和1208之前,_confirm框1204可以插入到入队操作1206和1208之后。插入的_request和_confirm API框使得以不同顺序发起操作1208和1212。如果没有检测到错误,可以从DCbug候选项列表中删减缺陷候选对(1208,1212)。如果在该过程中出现严重错误,可以在列表中保留DCbug候选项。

[0160] 下文所述示例提供该控制器服务器的两种实施方式:一种处于真正的分布模式,通过套接字与不同机器上运行的测试客户端进行通信;另一种处于单机模式,通过文件操作与同一机器上不同进程中运行的测试客户端进行通信。

[0161] 使用上述基础设施,假设有DCbug报告(s,t),剩下的问题是在哪里放置_request和_confirm API。_confirm API可以插入到缺陷报告中的堆访问之后。因此,以下内容关注_request API的放置。

[0162] 如图12中所示,一个方案是将_request放在s和t之前。但是,这种方法方法有时不起作用,要么因为它会导致挂起,要么因为它会由于大量s/t的动态实例产生过多待发送到控制器服务器的_request消息。可以根据以下分析配置一个示例DCatch系统帮助解决这个问题。

[0163] 首先,DCatch系统可以警告由于_request API不合适的放置导致的潜在挂起,并建议一个或多个不会导致挂起的放置位置。具体地,当s和t都在事件处理程序内且它们的事件处理程序对应单用户队列时,DCatch系统可以警告用户会出现挂起并建议将_request API插入到对应事件入队函数中。类似地,如果s和t都位于RPC处理程序内且它们的RPC函数由同一节点中的同一RPC处理线程执行,DCatch可以建议将_request API插入到发起RPC的对应函数中。如果s和t位于同一锁定保护的重要部分内,DCatch可以暗示将_request插入到对应重要部分之前。如上文所述,DCatch可以基于其跟踪结果中的锁定相关记录获取该重要部分信息。

[0164] 第二,DCatch可以在发现s和t的大量动态实例之后发布警告,并建议更好的放置位置。例如,DCbug报告可以含有s和t的调用栈,DCatch系统可以检查运行时跟踪以确定报告是否含有s的对应调用栈的大量动态实例(t的分与此相同)。在这些实例中,DCatch可以检查先行发生图以查找不同节点中导致s的操作o,并检查o是否是放置_request的更好的位置。该分析是有效的,因为很多事件处理程序和RPC函数可能在同一调用栈执行,因此如果没有DCatch系统的这种支持,可能使缺陷触发变得非常复杂。应注意,以上描述的特征对触发DCbug都是唯一的。

[0165] 下文描述DCatch系统的示例实施方式。可以使用Javassist或其它动态Java字节码重写工具实施跟踪HB相关操作,每当加载类时这些工具允许分析和插桩Java字节码。

[0166] 如上文所述,HB相关操作涉及与线程、事件处理、RPC、套接字和节点间通知协议相关的函数。按照`java.lang.Thread`接口可以轻易识别所有线程相关操作。不同系统上略微不同的接口支持其它操作。

[0167] 在一个示例中,在Hadoop和HBase中使用`java.beans.EventHandler`实施事件处理。事件处理程序函数的技术原型是`EventHandler::handle(Event e)`,其中由参数内容确定事件处理动作。Cassandra和Zookeeper使用自己的事件处理接口。实施和调用事件处理程序的方式类似于Hadoop和HBase中的实施和调用。

[0168] 对于RPC,HBase与Hadoop的后续版本共享同一RPC库接口——`VersionedProtocol`。由该接口示例化的类中声明的所有方法都是RPC函数,因此DCatch系统可以轻易标识。Hadoop的后续版本使用略微不同的RPC接口——`ProtoBase`,该接口以与`VersionedProtocol`相同的方式标识RPC函数。

[0169] 对于套接字发送和接收,Cassandra具有超类`IVerbHandler`处理套接字通信,由函数`IVerbHandler::sendOneWay(Message,EndPoint)`进行套接字发送。因此,DCatch系统可以轻易标识所有这种套接字消息发送函数调用,以及对应消息对象。Zookeeper对所有套接字消息使用超类`Record`。每个套接字发送前面是`Record`对象的新的实例,通过`socket::write(Record)`进行发送。因此,也可以轻易标识套接字消息。

[0170] 一个示例DCatch系统首先使用静态Java字节码分析框架——WALA来静态分析目标软件,标识所有RPC/套接字/事件相关函数,并存储`DFunctionList`文件中的分析结果供后续运行时分析使用。然后示例DCatch系统使用Javassist将跟踪函数插入到上文所述的每个堆或静态变量访问之前。具体地,DCatch系统可以使用Javassist插件,每当类C加载到JVM时,Javassist插件执行以下操作:(1)标识C中是`DFunctionList`的一部分的所有方法;(2)对于每个这种方法函数,标识所有`getField/putfield`指令(例如堆访问)和`getstatic/putstatic`指令(例如静态变量访问);(3)对于每个这种指令,将跟踪函数插入到指令之前,跟踪函数产生跟踪记录。

[0171] 示例DCatch系统记录通过套接字通信和通过每个RPC调用发送/接收的每个包的唯一ID。为实现该系统,在套接字发送或RPC调用侧,生成随机数并将随机数与套接字消息或RPC调用一起发送。在接收侧,系统解析随机数并将随机数放置在对应该跟踪记录中。具体地,DCatch系统静态转换目标软件,从而在调用RPC/套接字发送时为每个RPC/套接字发送函数添加一个额外参数并插入代码为每个这种参数生成随机值。

[0172] 如上所述,在已知以下内容时,DCatch系统可以用于任何分布式处理系统:(1)RPC接口是什么;(2)哪些API用于套接字通讯;(3)哪些API用于事件入队/出队/处理程序;(4)事件队列是否是FIFO且是否具有一个或多个处理程序线程。提供以上内容应该是直接且相当简单的,因为只标识相对较少数量的(RPC/事件/套接字)接口或原型,而不是相当大数量的实例函数。现有分布式系统中精确检测DCbug需要以上说明。

[0173] 一旦这些项目已知,DCatch规则容易用于分布式处理系统。为了在分布式处理系统上实施DCatch,首先使用WALA和/或Javassist等静态和/或动态字节码转换/分析框架修改处理系统的组件,以插入命令,该命令用于在RPC函数、实施套接字操作的函数和事件处理程序函数中跟踪对对象的访问。然后在多节点系统上运行修改的系统,以跟踪访问对象的函数。DCatch系统然后分析跟踪,以构建图并确定可能导致DCbug的候选操作对。然后再

次分析处理系统的组件,以删减不延伸到多个节点的潜在DCbug。再次修改系统以插入时延(例如睡眠状态)和/或_request和_confirm API,以调整系统时序。多次执行修改的系统尝试候选操作对的不同组合方式,以确定哪些潜在DCbug真正会出现。

[0174] 在一个实施例中,本文所述的函数或算法可在软件中实施。软件可由存储在计算机可读介质或计算机可读存储设备,如一种或多种基于非瞬时性存储器或其它类型的硬件的存储设备(局域或联网)上的计算机可执行指令组成。另外,这种函数对应于模块,模块可以是软件、硬件、固件或其任何组合。可视需要在一个或多个模块中执行多个函数,且描述的实施例仅仅是示例。可以在计算系统上执行软件,计算系统例如数字信号处理器、ASIC、微处理器、大型主机处理器或计算机系统上运行的其它类型的处理器,计算机系统例如为个人电脑、服务器或其它计算系统,从而将这种计算系统转变为专用编程机器。

[0175] 图13是客户端、服务器和基于云的计算系统资源的计算电路系统的框图,这些资源用于根据示例实施例实施算法和执行方法。分布式计算系统可以包括图13中所示的电路的多个实例,且包括上述DCatch系统。并非所有组件都需于各种实施例中。例如,分布式计算系统的客户端、服务器和网络资源中的每一个可以分别使用一组不同的组件,或者在服务器或大型主机的情况下,使用例如更大存储设备。

[0176] 计算机1300形式的一个示例计算系统可以包括处理单元1302、存储器1303、可移动存储器1310和不可移动存储器1312。处理单元1302可以是单核或多核设备。虽然示例计算系统示出且描述为计算机1300,但在不同实施例中计算系统可呈不同形式。例如,计算系统可以为智能手机、平板电脑、智能手表或包括图13中所示并描述的相同或类似元件的其它计算设备。智能手机、平板电脑和智能手表等设备统称为移动设备或用户设备。进一步,虽然各种数据存储元件示出为计算系统1300的一部分,存储可以还包括或者也包括基于本地服务器的存储器或基于云的通过网络可以访问的存储器,所述网络例如为局域网(local area network,LAN)、个人局域网(personal area network,PAN)、例如因特网的广域网(wide area network,WAN)。

[0177] 存储器1303可以包括易失性存储器1314和非易失性存储器1308。计算机1300可以包括或可以接入包括多种计算机可读介质的计算环境,计算机可读介质例如易失性存储器1314和非易失性存储器1308、可移动存储器1310和不可移动存储器1312。计算机存储装置包括随机存取存储器(random access memory,RAM)、只读存储器(read only memory,ROM)、可擦除可编程只读存储器(erasable programmable read-only memory,EPR0M)和电可擦除可编程只读存储器(electrically erasable programmable read-only memory,EEPROM)、快闪存储器或其它存储器技术、压缩光盘只读存储器(compact disc read-only memory,CD ROM)、数字通用光盘(Digital Versatile Disk,DVD)或其它光盘存储装置、磁带盒、磁带、磁盘存储或其它磁性存储装置,或能够存储计算机可读指令的任何其它介质。

[0178] 计算机1300可以包括或可以接入包括输入接口1306、输出接口1304和通信连接或接口1316。输出1304可以包括也可以充当输入设备的显示器设备,例如触摸屏。输入1306可包括触摸屏、触摸板、鼠标、键盘、相机、一个或多个设备专用按钮、集成在计算系统1300内或经由有线或无线数据连接耦合至计算机系统1300的一个或多个传感器和其它输入设备中的一个或多个。计算机可以在连网环境中运行,使用通信连接连接至一个或多个远程计算机,例如大型主机和/或数据库服务器。远程计算机可以包括个人计算机(personal

computer, PC)、服务器、路由器、网络PC、对等设备或其它公共网络节点等。通信连接可包括局域网(local area network, LAN)、广域网(wide area network, WAN)、蜂窝网、Wi-Fi、蓝牙或其它网络。

[0179] 存储在计算机可读介质上的计算机可读指令可由计算机1300的处理单元1302执行。硬盘驱动器、CD-ROM和RAM是包括例如存储设备的非瞬时性计算机可读介质的物件的一些示例。术语计算机可读介质和存储设备不包括载波, 因为认为载波太瞬时。举例来说, 计算机程序1318可用于使处理单元1302执行一种或多种本文所述的方法或算法。

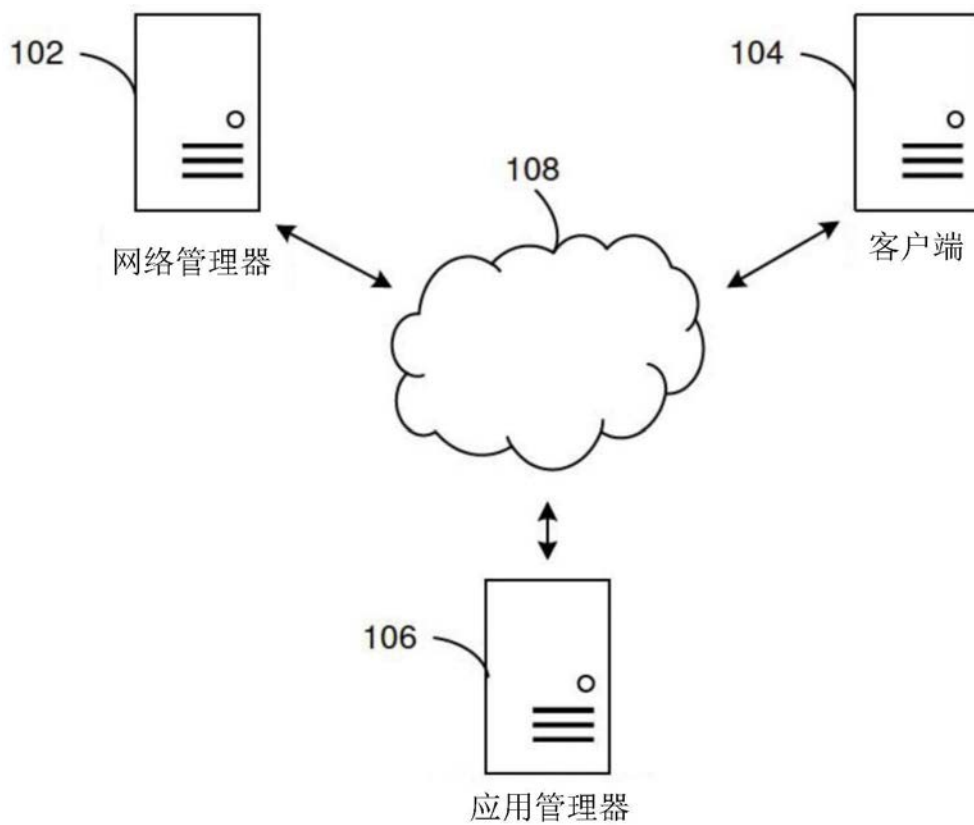


图1

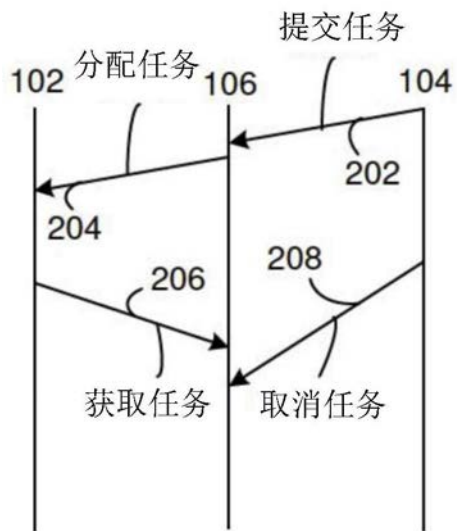


图2A

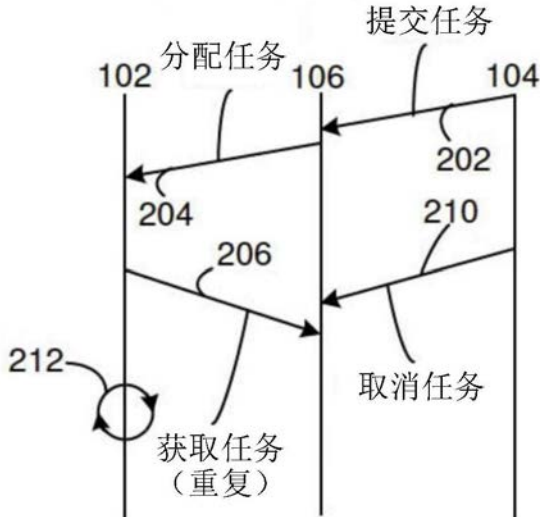


图2B

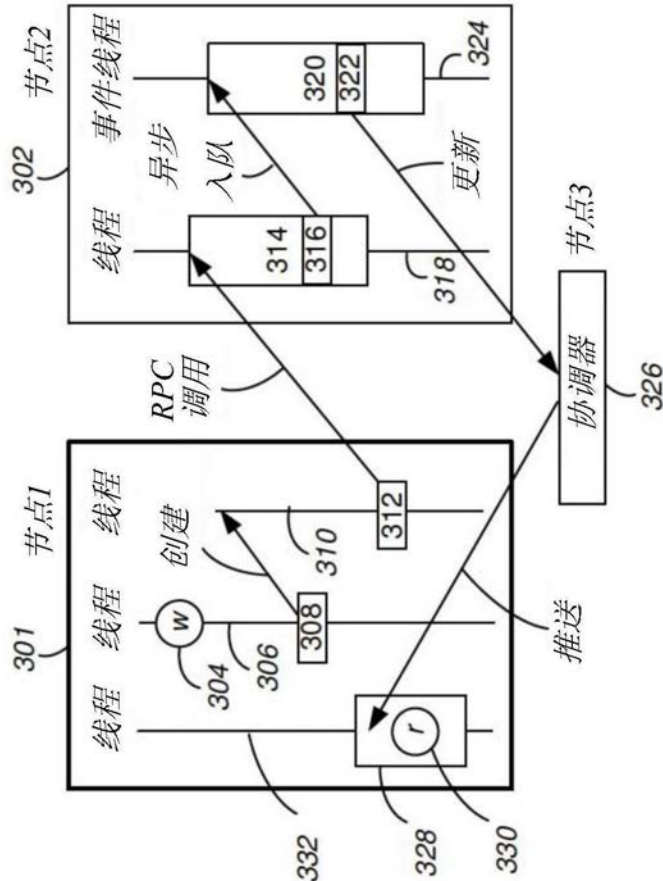


图3

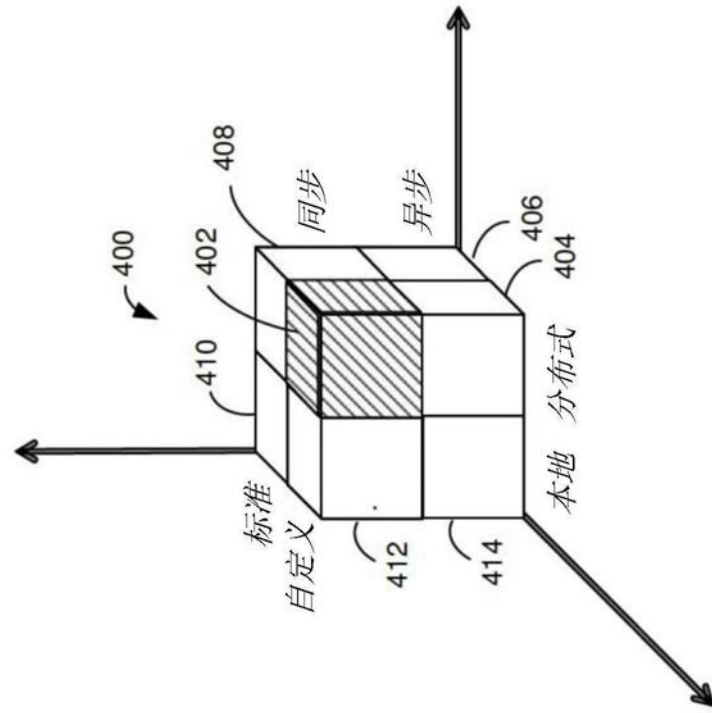


图4

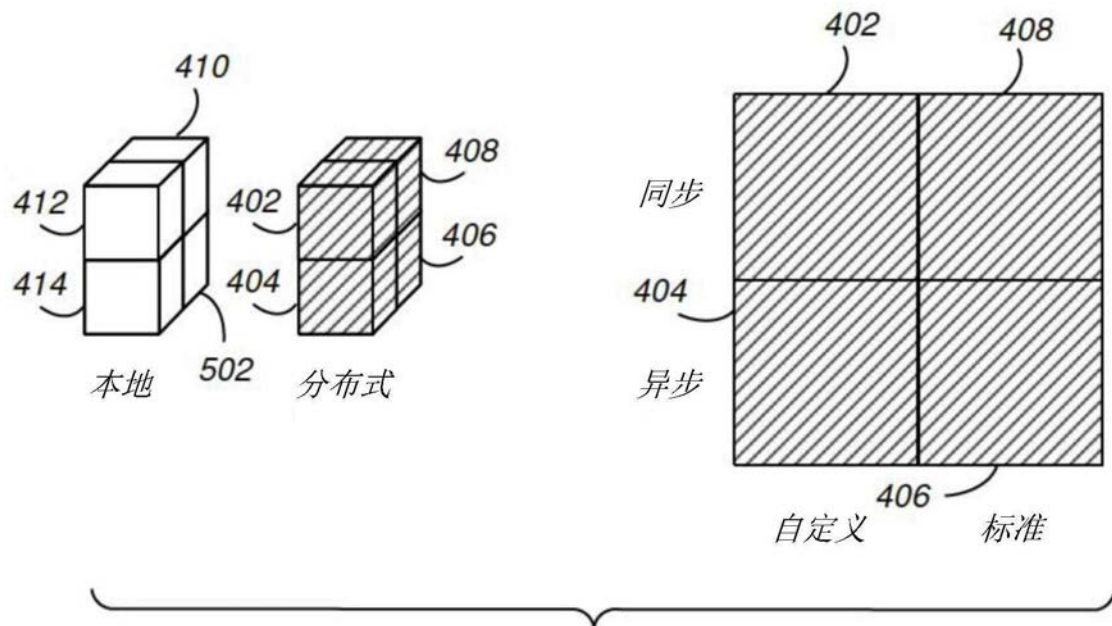


图5

图5

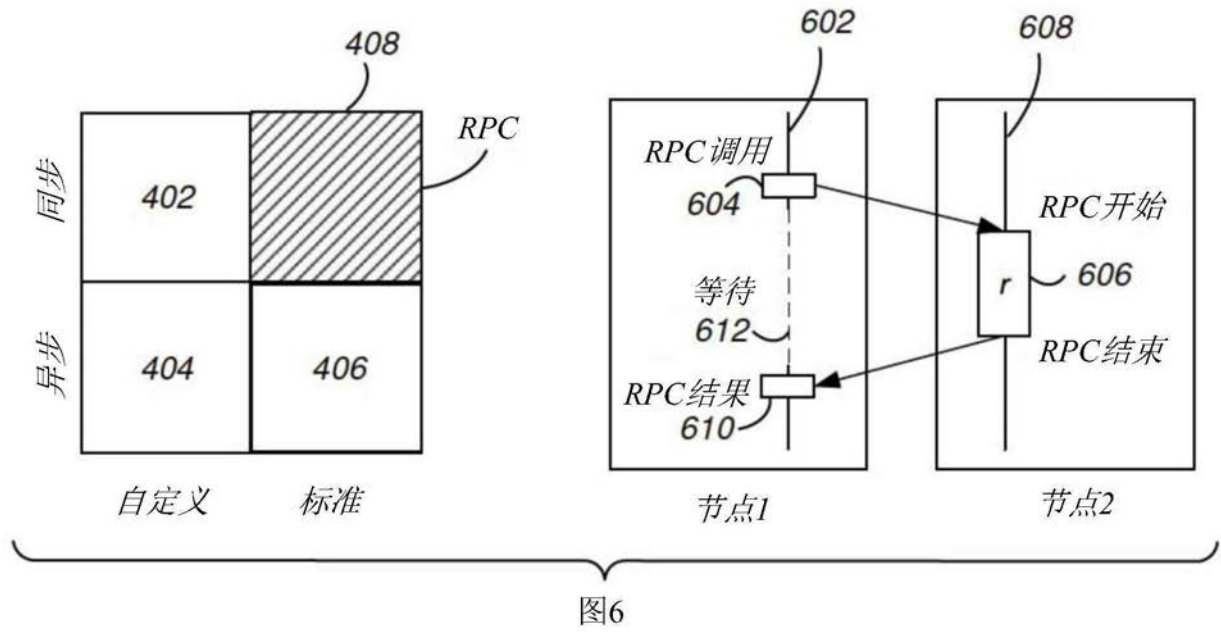


图6

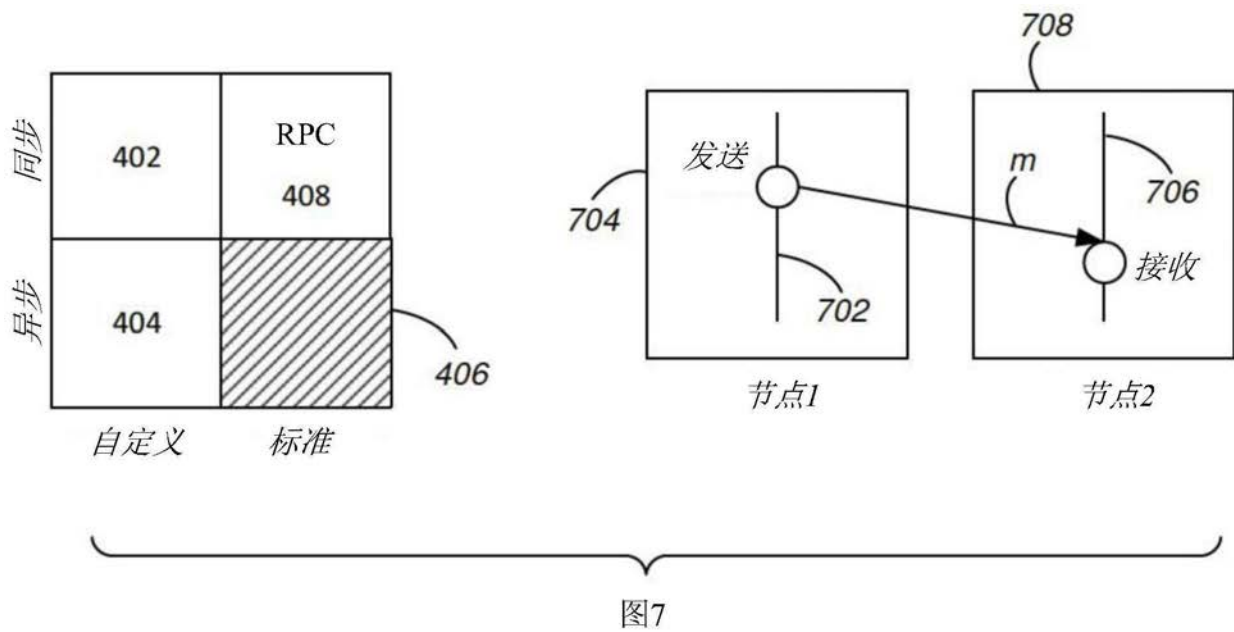
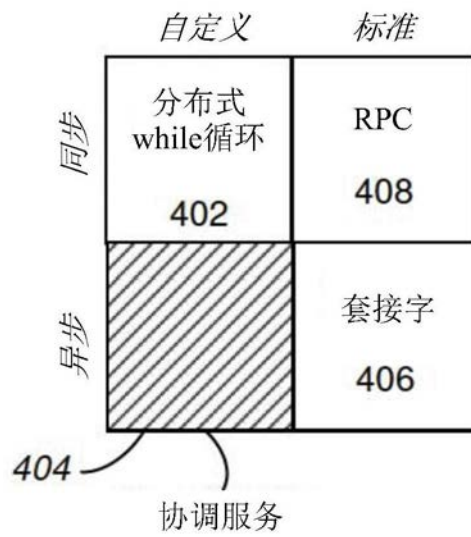
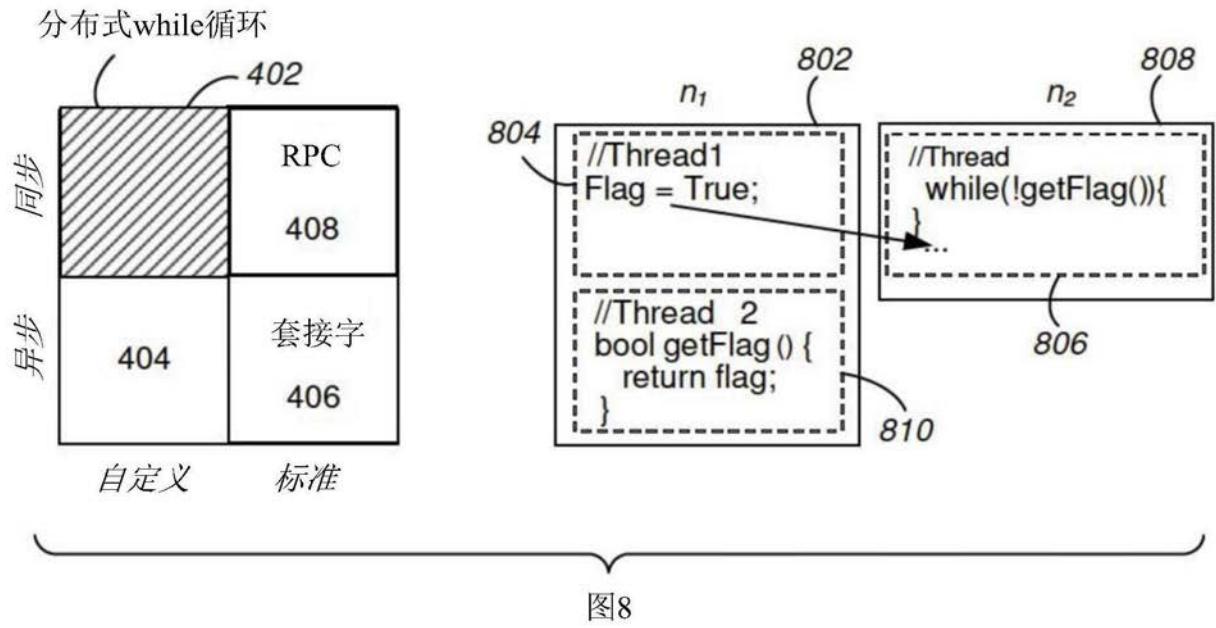


图7



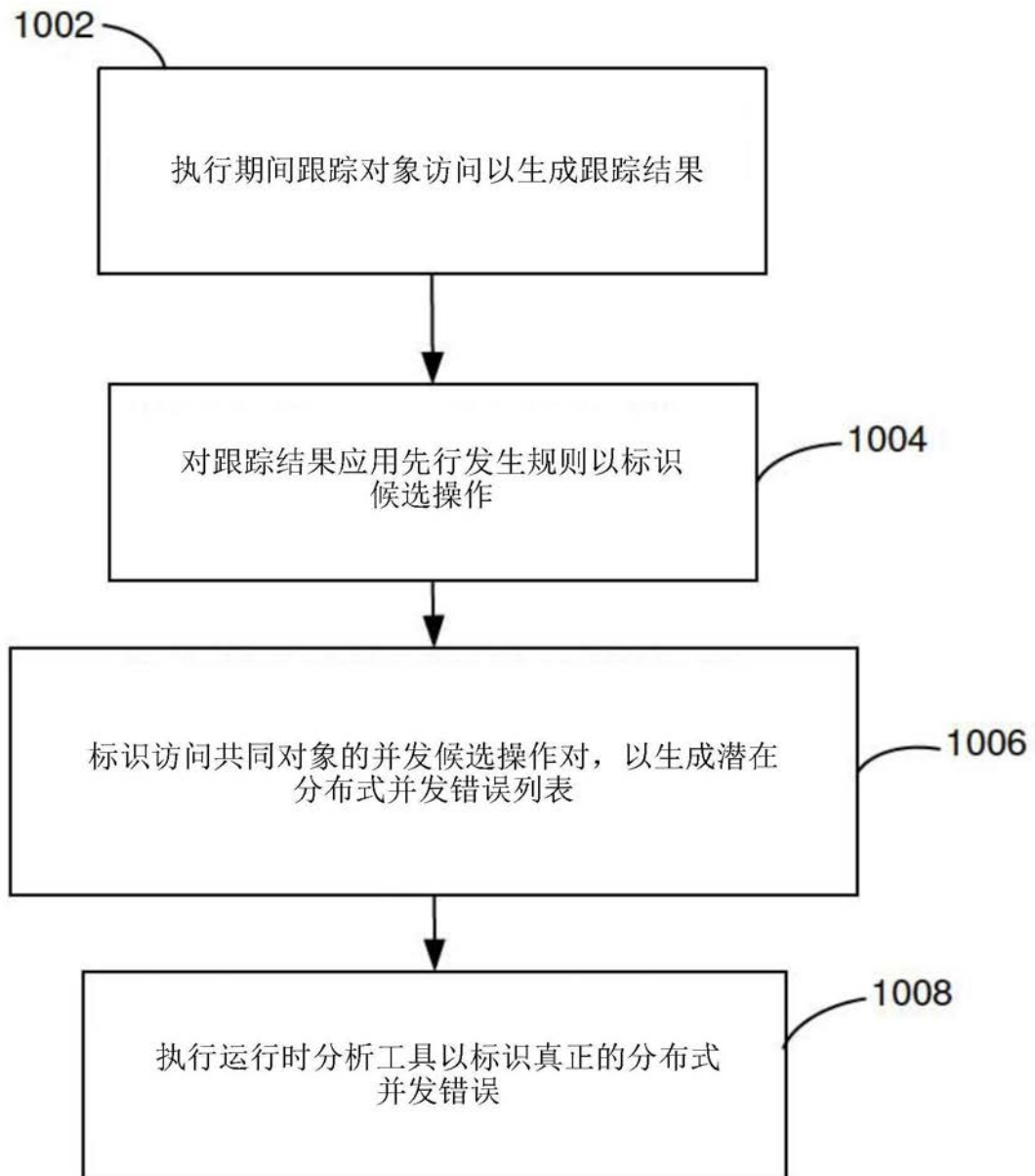


图10A

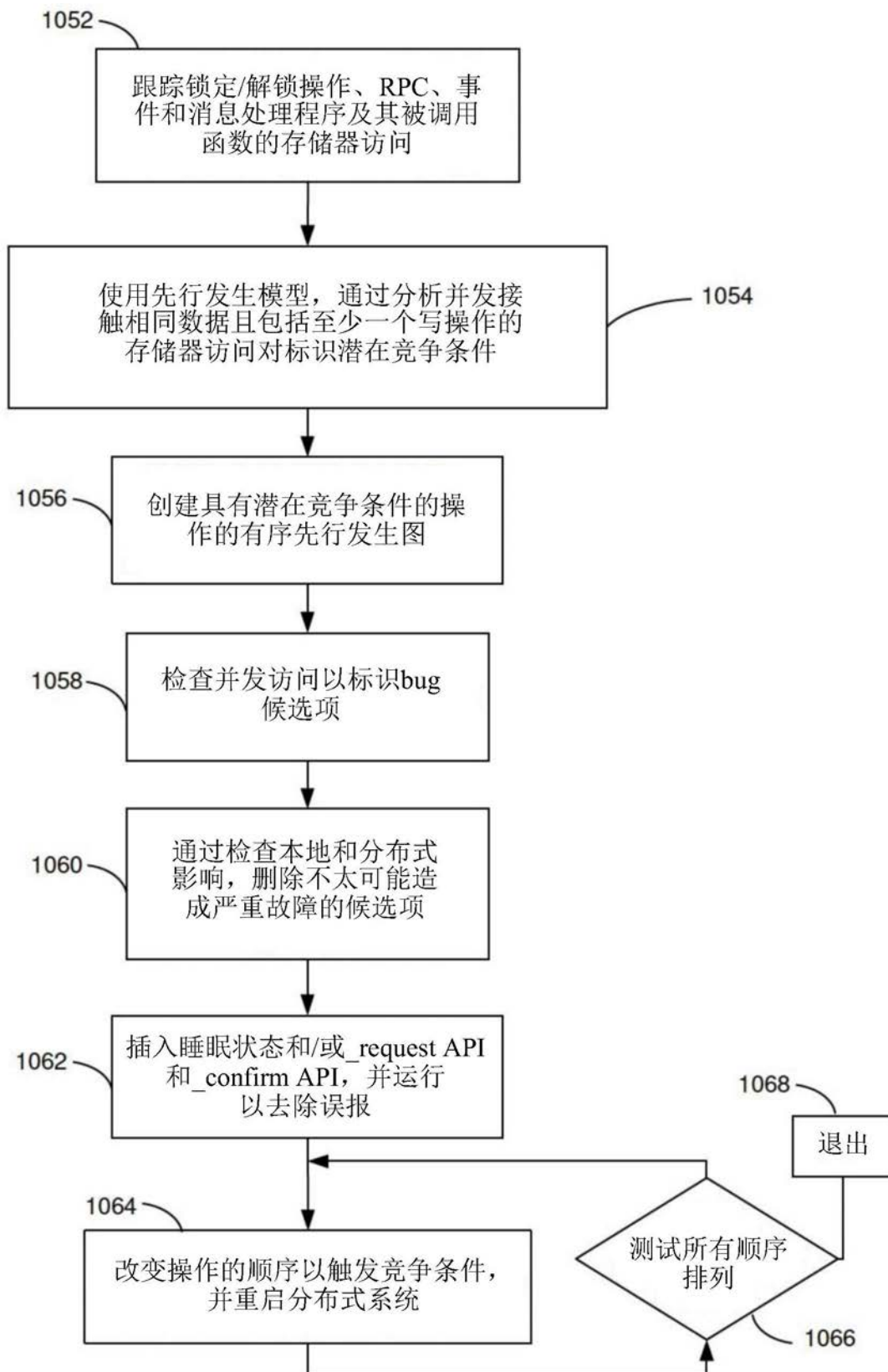


图10B

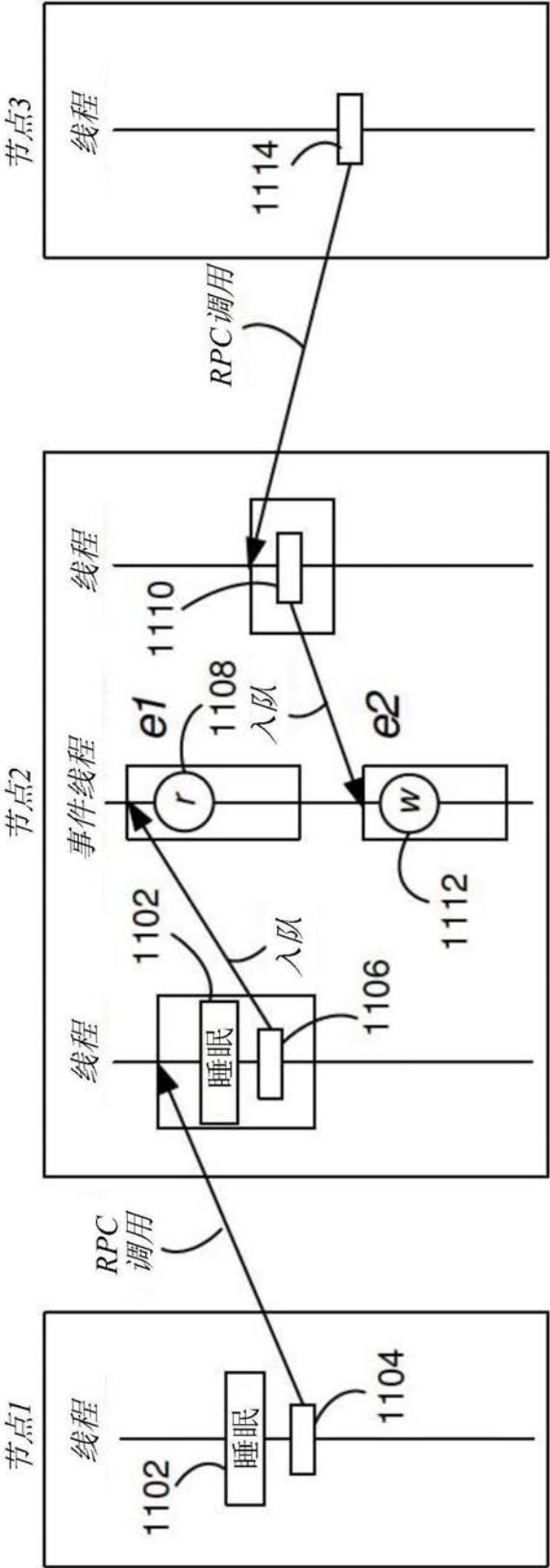


图11

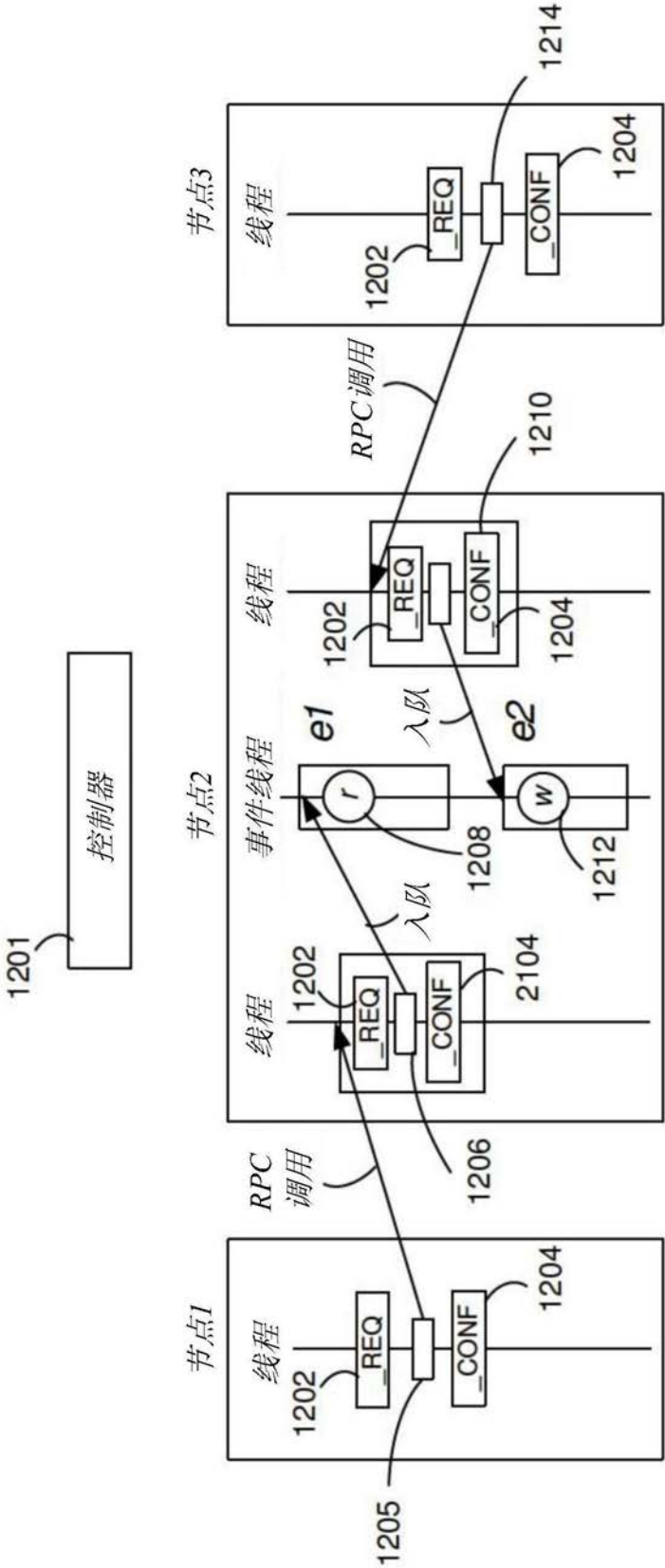


图12

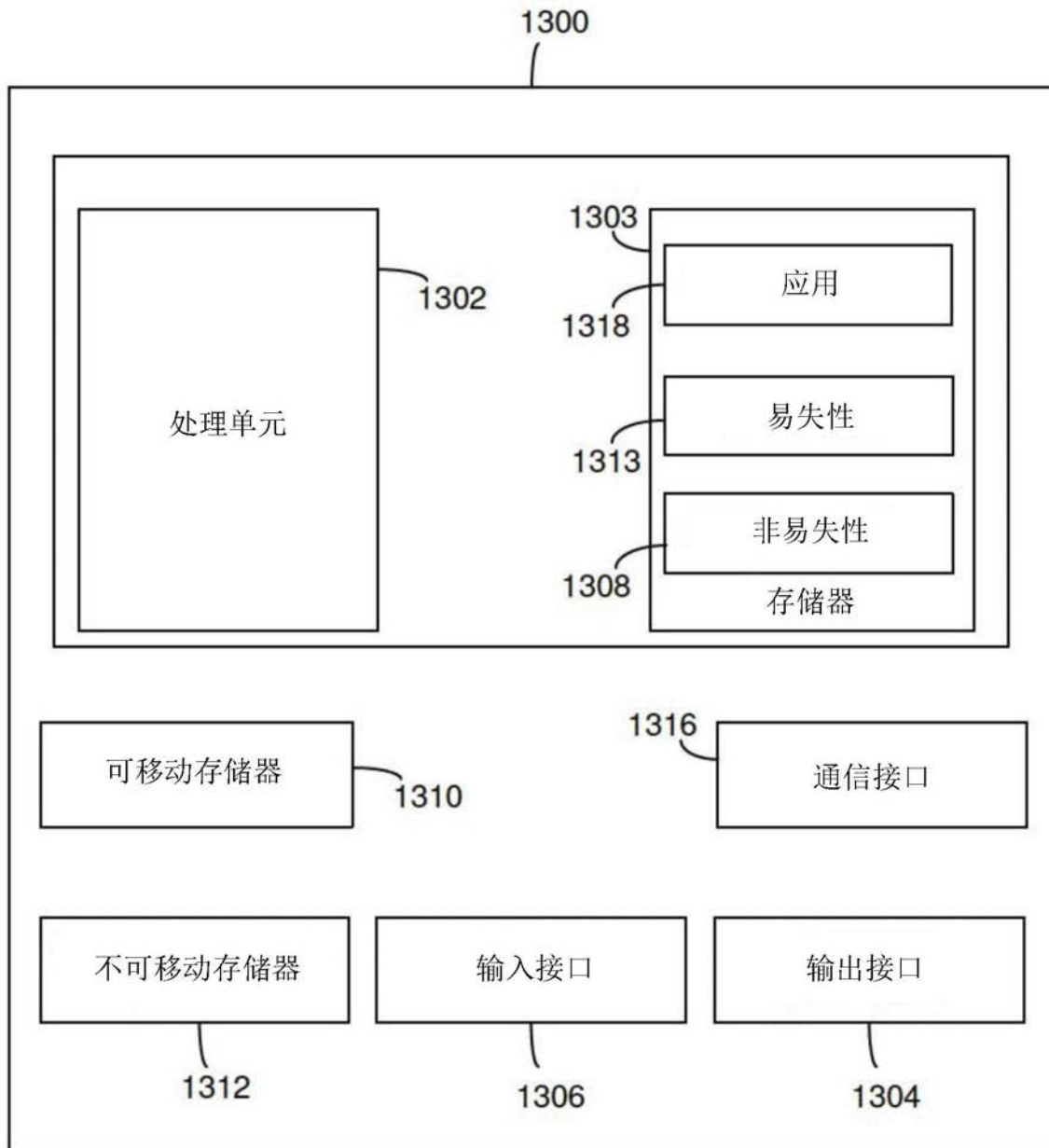


图13